BLUETOOTH CHAT APPLICATION USING ANDROID

Mini Project Report Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

In partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

A.SUSHEN	**14BD1A05AX**
G.VIKAS	**14BD1A05B7**
S.RAHUL	**14BD1A05BU**
V.PAVAN KUMAR	**14BD1A05CH**

Under the Esteemed Guidance of

Mrs. K L S Sirisha
(Asst/ Associate Professor, Department of CSE)

DEPARTMENT OF COMPUTER SCIENCE ANDENGINEERING
KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(Approved by AICTE, Affiliated to JNTUH)
Narayanaguda, Hyderabad

Academic Year: 2017-18

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this bonafide record of the project report titled **" *BLUETOOTH CHAT APPLICATION USING ANDROID* "** which is being presented as the Mini Project report by

A.SUSHEN	**14BD1A05AX**
G.VIKAS	**14BD1A05B7**
S.RAHUL	**14BD1A05BU**
V.PAVAN KUMAR	**14BD1A05CH**

In partial fulfilment for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** affiliated to the **Jawaharlal Nehru Technology University, Hyderabad.**

Internal Guide
(Mrs. K.L.S SIRISHA)

Head of the Department
(Dr. J.V.S Srinivas)

External Examiner

DECLARATION

We hereby declare that the results embodied in this dissertation entitled **" *BLUETOOTH CHAT APPLICATION USING ANDROID* "** has been carried out by us together during the academic year 2017-2018 as a partial fulfilment of the award of the B. Tech degree in Computer Science and Engineering from JNTUH. We have not submitted this report to any other university or organization for award of any other degree.

A.SUSHEN	**14BD1A05AX**
G.VIKAS	**14BD1A05B7**
S.RAHUL	**14BD1A05BU**
V.PAVAN KUMAR	**14BD1A05CH**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance have been a source of inspiration throughout the course of the project work. We are glad to have the opportunity of expressing our gratitude to all of them.

We are thankful to Mrs. Deepa Ganu, Director Academics, Keshav Memorial Institute of Technology. We thank her for her kind support to do our project.

We are thankful to Dr. J.V.S Srinivas, Professor and Head of the Department of Computer Science and Engineering, Keshav Memorial Institute of Technology. We thank him for his support and guidance throughout the project. He is a source of inspiration for innovative ideas and his kind support is well known to all his students and colleagues.

We are thankful to our Guide Mrs K L S Sirisha, Asst. Professor of Computer Science and Engineering Dept., Keshav Memorial Institute of Technology, for his support and guidance throughout the project.

We also like to thank the Computer Science and Engineering Branch Coordinator, Dr.K.Bhargavi, Associate Professor, CSE Dept., Keshav Memorial Institute of Technology. We thank her for providing us timely advice throughout the project work.

We would like to express our sincere gratitude to Project Co-ordinator for his and support, guidelines and encouragement.

We also express my sincere gratitude to the Management and Principal of Keshav Memorial Institute of Technology for their encouragement, facilities provided and support.

Finally, We would like to make a special mention of all our family members and friends who helped us for the successful completion of the project report.

A.SUSHEN	14BD1A05AX
G.VIKAS	14BD1A05B7
S.RAHUL	14BD1A05BU
V.PAVAN KUMAR	14BD1A05CH

DESCRIPTION

PAGE NO

ABSTRACT

I

CONTENTS

CHAPTER 1 INTRODUCTION 1-2

CHAPTER 2 BLUETOOTH 3-26

2.1 NAME AND LOGO 4
2.2 COMMUNICATION AND CONNECTION 4
2.3 USES 5
2.4 LIST OF APPLICATIONS 5
2.5 BLUETOOTH VS. WI-FI (IEEE 802.11) 7
2.6 DEVICES 8
2.7 SPECIFICATIONS AND FEATURES 9
2.8 TECHNICAL INFORMATION 17
2.9 PAIRING AND BONDING 21
2.10 SECURITY CONCERNS 24
2.11 SECURITY 25

CHAPTER 3 RFCOMM 27-33
3.1 INTRODUCTION 27
3.1.1 OVERVIEW 28
3.1.2 DEVICE TYPES 28
3.2 RFCOMM SERVICE OVERVIEW 31
3.2.1 NULL MODEM EMULATION 31
3.3 MULTIPLE EMULATED SERIAL PORTS 32
3.3.1 MULTIPLE EMULATED SERIAL PORTS BETWEEN TWO DEVICES 32

3.3.2 MULTIPLE EMULATED SERIAL PORTS AND MULTIPLE BT DEVICES 33

CHAPTER 4 SERVICE INTERFACE DESCRIPTION 34-35

4.1 SERVICE DEFINITION MODEL 34
4.2 FRAME TYPES TABLE 4.1 SHOWS THE TS 7.10 FRAME TYPES THAT ARE SUPPORTED IN RFCOMM. 35
4.3 COMMANDS 35
4.4 CONVERGENCE LAYERS 35

CHAPTER 5 TS 07.10 ADAPTATIONS FOR RFCOMM 36-39

5.1 MEDIA ADAPTATION 36
5.1.1 FCS CALCULATION 36
5.2 TS 07.10 MULTIPLEXER START-UP AND CLOSEDOWN PROCEDURE 36
5.2.1 START-UP PROCEDURE 37
5.2.2 CLOSE-DOWN PROCEDURE 37
5.2.3 LINK LOSS HANDLING 37
5.3 SYSTEM PARAMETERS 37
5.4 MULTIPLEXER CONTROL COMMANDS 38
5.4.1 REMOTE LINE STATUS COMMAND (RLS) 39

CHAPTER 6 FLOW CONTROL 40-42

6.1 L2CAP FLOW CONTROL IN OVERVIEW 40
6.2 WIRED SERIAL PORT FLOW CONTROL 40
6.3 RFCOMM FLOW CONTROL 40
6.4 PORT EMULATION ENTITY SERIAL

FLOW CONTROL 41

CHAPTER 7 INTERACTION WITH OTHER ENTITIES 43-46

7.1 PORT EMULATION
AND PORT PROXY ENTITIES 43
7.1.1 PORT EMULATION ENTITY 43
7.1.2 PORT PROXY ENTITY 43
7.2 SERVICE REGISTRATION AND DISCOVERY 44
7.3 LOWER LAYER DEPENDENCIES 45
7.3.1 RELIABILITY 45
7.3.2 LOW POWER MODES 46

CHAPTER 8 ANDRIOD STUDIO 47

8.1 PROJECT STRUCTURE 48
8.1.1 USER INTERFACE 50
8.2 TOOL WINDOWS 51
8.3 CODE COMPLETION 52
8.4 FIND SAMPLE CODE 53
8.5 NAVIGATION 53

8.6 STYLE AND FORMATTING 54
8.7 VERSION CONTROL BASICS 54
8.8 GRADLE BUILD SYSTEM 55
8.8.1 BUILD VARIANTS 56
8.8.2 MULTIPLE APK SUPPORT 56
8.8.3 RESOURCE SHRINKING 56
8.8.4 MANAGING DEPENDENCIES 56
8.9 DEBUG AND PROFILE TOOLS 57
8.9.1 INLINE DEBUGGING 57
8.9.2 PERFORMANCE PROFILERS 58
8.9.3 HEAP DUMP 58
8.9.4 MEMORY PROFILER 58
8.9.5 DATA FILE ACCESS 58
8.9.6 CODE INSPECTIONS 59
8.9.7 ANNOTATIONS IN ANDROID STUDIO 59

CHAPTER 9 IMPLEMENTATION 60-63

CHAPTER 10 TESTING 64-70

10.1 INTRODUCTION TO TESTING 64
10.2 TYPES OF TESTING 65
10.3 TEST CASES 66

CHAPTER 11 OUTPUT 71

CHAPTER 12 CONCLUSION 72

LIST OF FIGURES

Figure 1:	BLUETOOTH CHAT LOGO	1
Figure 2:	BLUETOOTH PHONE HEADSET	5
Figure 3:	BLUETOOTH USE DONGLE	8
Figure 4:	RFCOMM COMMUNICATION SEGMENT	28
Figure 5:	NULL MODEM EMULATOR	32
Figure 6:	MULTIPLE EMULATED SERIAL PORTS	33
Figure 7:	EMULATING SERIAL PORTS COMING FROM 2 BT DEVICE	33
Figure 8:	RFCOMM REFERENCE MODEL	34
Figure 9:	THE PROJECT FILE IN ANDRIOD VIEW	48
Figure 10:	THE PROJECT FILE IN PROBLEM VIEW SHOWING A LAYOUT FILE WITH A PROBLEM	49

ABSTRACT

<u>Android Studio:</u> Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development.

It's a known fact that people are into "social media" life more than the social one, so they prefer to chat on an application more than speaking directly (pun). This application will focus on one such aspect where in you can chat with your peer in the available Bluetooth range.

The technology this application will use is:

<u>RFCOMM tools</u> – This is the asset to give a peer to peer connection via Bluetooth and enabling to share data between two devices. The application we are going to show will allow you to connect to a device that's already paired and gives you another option to connect in an insecure way without pairing. Although there are breaches that are involved in an RFCOMM enabled network, since this is a peer to peer connection, it will prompt you an option to connect and only then you will be connected to a device. The elaborated version of this application will be presented to you and will focus on the deployment of the application.

1 INTRODUCTION

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices, and building personal area networks (PANs). Invented by telecom vendor Ericsson in 1994, it was originally conceived as a wireless alternative to RS-232 data cables. It can connect several devices, overcoming problems of synchronization.

According to this, we can "build" a local are network (LAN) by connecting devices over Bluetooth. The Android platform includes support for the Bluetooth network stack, which allows a device to wirelessly exchange data with other Bluetooth devices. The application framework provides access to the Bluetooth functionality through the Android Bluetooth APIs. These APIs let applications wirelessly connect to other Bluetooth devices, enabling point-to-point and multipoint wireless features so we absolutely able to transferring data to other devices in the network circle.

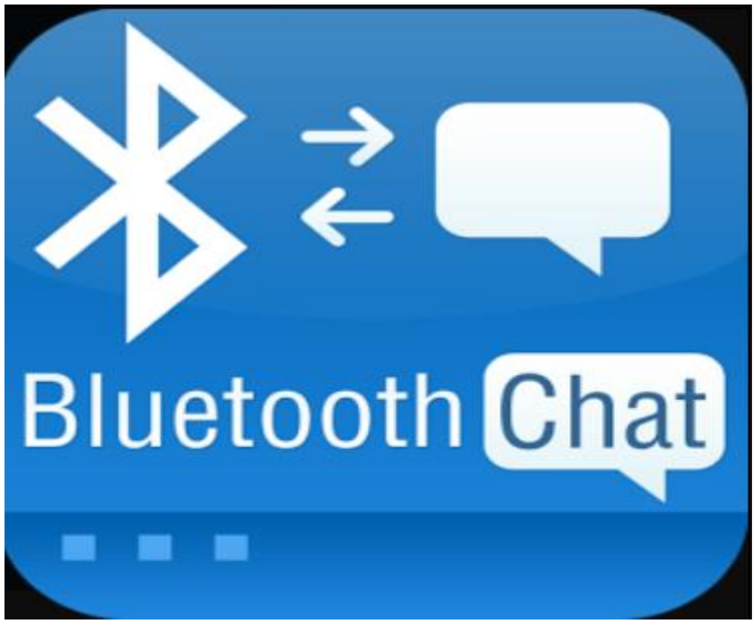


FIGURE 1 BLUETOOTH CHAT LOGO

This application allows two Android devices to carry out two-way text chat over Bluetooth. It demonstrates all the fundamental Bluetooth API capabilites, such as:

- Scanning for other Bluetooth devices
- Querying the local Bluetooth adapter for paired Bluetooth devices
- Establishing RFCOMM channels/sockets
- Connecting to a remote device
- Transfering data over Bluetooth

2 BLUETOOTH

Developed by	Bluetooth Special Interest Group
Industry	Personal area networks
Compatible hardware	Personal computers
	Smartphones
	Gaming consoles Wireless Audio Devices
Physical range	Typically less than 10 m (33 ft), up to 100 m (330 ft)
	Bluetooth 5.0: 40–400 m (100–1,000 ft)

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices and building personal area networks (PANs). Invented by telecom vendor Ericsson in 1994, it was originally conceived as a wireless alternative to RS-232 data cables.

The development of the "short-link" radio technology, later named Bluetooth, was initiated in 1989 by Nils Rydbeck, CTO at Ericsson Mobile in Lund, Sweden, and by Johan Ullman. The idea of this name was proposed in 1997 by Jim Kardach of Intel who developed a system that would allow mobile phones to communicate with computers.

2.1 Name and logo

Name

The name " Bluetooth " is an Anglicised version of the Scandinavian *Blåtand*/*Blåtann* (Old Norse *blátǫnn*), the epithet of the tenth - century king Harald Bluetooth who united dissonant Danish tribes into a single kingdom. The implication is that Bluetooth unites communications protocols.

The idea of this name was proposed in 1997 by Jim Kardach of Intel who developed a system that would allow mobile phones to communicate with computers. At the time of this proposal

he was reading Frans G. Bengtsson's historical novel *The Long Ships* about Vikings and King Harald Bluetooth.

2.2 Communication and Connection

A master BR/EDR Bluetooth device can communicate with a maximum of seven devices in a piconet (an ad-hoc computer network using Bluetooth technology), though not all devices reach this maximum. The devices can switch roles, by agreement, and the slave can become the master (for example, a headset initiating a connection to a phone necessarily begins as master—as initiator of the connection—but may subsequently operate as slave).

The Bluetooth Core Specification provides for the connection of two or more piconets to form a scatternet, in which certain devices simultaneously play the master role in one piconet and the slave role in another.

At any given time, data can be transferred between the master and one other device (except for the little-used broadcast mode). The master chooses which slave device to address; typically, it switches rapidly from one device to another in a round-robin fashion. Since it is the master that chooses which slave to address, whereas a slave is (in theory) supposed to listen in each receive slot, being a master is a lighter burden than being a slave.

2.3 Uses

Bluetooth is a standard wire-replacement communications protocol primarily designed for low power consumption , with a short range based on low-cost transceiver microchips in each device. Because the devices use a radio (broadcast) communications system, they do not have to be in visual line of sight of each other ; however, a quasi optical wireless path must be viable. Range is power-class-dependent, but effective ranges vary in practice. See the table "Ranges of Bluetooth Devices by Class".

Officially Class 3 radios have a range of up to 1 metre (3 ft), Class 2, most commonly found in mobile devices, 10 metres (33 ft), and Class 1, primarily for industrial use cases, 100 metres (300 ft). Bluetooth Marketing qualifies that Class 1 range is in most cases 20–30 metres (66–98 ft), and Class 2 range 5–10 metres (16–33 ft).

2.4 List of applications

A typical Bluetooth mobile phone headset

FIGURE 2 BLUETOOTH PHONE HEADSET

- Wireless control of and communication between a mobile phone and a handsfree headset. This was one of the earliest applications to become popular.

- Wireless control of and communication between a mobile phone and a Bluetooth compatible car stereo system. Wireless control of and communication with iOS and Android device phones, tablets and portable wireless speakers.
- Wireless Bluetooth headset and Intercom. Idiomatically, a headset is sometimes called "a Bluetooth".
- Wireless streaming of audio to headphones with or without communication capabilities. • Wireless streaming of data collected by Bluetooth-enabled fitness devices to phone or PC.
- Wireless networking between PCs in a confined space and where little bandwidth is required.
- Wireless communication with PC input and output devices, the most common being the mouse, keyboard and printer.
- Transfer of files, contact details, calendar appointments, and reminders between devices with OBEX.
- Replacement of previous wired RS-232 serial communications in test equipment, GPS receivers, medical equipment, bar code scanners, and traffic control devices.
- For controls where infrared was often used.
- For low bandwidth applications where higher USB bandwidth is not required and cablefree connection desired.
- Sending small advertisements from Bluetooth-enabled advertising hoardings to other, discoverable, Bluetooth devices.
- Wireless bridge between two Industrial Ethernet (e.g., PROFINET) networks.
- Seventh and eighth generation game consoles such as Nintendo's Wii, and Sony's PlayStation 3 use Bluetooth for their respective wireless controllers.
- Dial-up internet access on personal computers or PDAs using a data-capable mobile phone as a wireless modem.
- Short-range transmission of health sensor data from medical devices to mobile phone, set-top box or dedicated telehealth devices.
- Allowing a DECT phone to ring and answer calls on behalf of a nearby mobile phone.
- Real-time location systems (RTLS) are used to track and identify the location of objects in real time using "Nodes" or "tags" attached to, or embedded in, the objects tracked, and

"Readers" that receive and process the wireless signals from these tags to determine their locations.

- Personal security application on mobile phones for prevention of theft or loss of items. The protected item has a Bluetooth marker (e.g., a tag) that is in constant communication with the phone. If the connection is broken (the marker is out of range of the phone) then an alarm is raised. This can also be used as a man overboard alarm. A product using this technology has been available since 2009.
- Calgary, Alberta, Canada's Roads Traffic division uses data collected from travelers' Bluetooth devices to predict travel times and road congestion for motorists.
- Wireless transmission of audio (a more reliable alternative to FM transmitters)
- Live video streaming to the visual cortical implant device by Nabeel Fattah in Newcastle university 2017

2.5 Bluetooth vs. Wi-Fi (IEEE 802.11)

Bluetooth and Wi-Fi (Wi-Fi is the brand name for products using IEEE 802.11 standards) have some similar applications: setting up networks, printing, or transferring files. Wi-Fi is intended

as a replacement for high-speed cabling for general local area network access in work areas or home. This category of applications is sometimes called wireless local area networks (WLAN). Bluetooth was intended for portable equipment and its applications. The category of applications is outlined as the wireless personal area network (WPAN). Bluetooth is a replacement for cabling in a variety of personally carried applications in any setting, and also works for fixed location applications such as smart energy functionality in the home (thermostats, etc.).

Wi-Fi and Bluetooth are to some extent complementary in their applications and usage. Wi-Fi is usually access point - centered, with an asymmetrical client-server connection with all traffic routed through the access point, while Bluetooth is usually symmetrical, between two Bluetooth devices. Bluetooth serves well in simple applications where two devices need to connect with minimal configuration like a button press, as in headsets and remote controls, while Wi-Fi suits better in applications where some degree of client configuration is possible and high speeds are required, especially for network access through an access node. However, Bluetooth access points do exist, and ad-hoc connections are possible with Wi-Fi though not as simply as with Bluetooth. Wi-Fi Direct was recently developed to add a more Bluetooth-like ad-hoc functionality to Wi-Fi.

2.6 Devices

A Bluetooth USB dongle with a 100 m range.

FIGURE 3 BLUETOOTH USB DONGLE

Bluetooth exists in many products, such as telephones, speakers, tablets, media players, robotics systems, handheld, laptops, console gaming equipment as well as some high definition headsets, modems, and watches. The technology is useful when transferring information between two or more devices that are near each other in low-bandwidth situations. Bluetooth is commonly used to transfer sound data with telephones (i.e., with a Bluetooth headset) or byte data with hand-held computers (transferring files).

Bluetooth protocols simplify the discovery and setup of services between devices. Bluetooth devices can advertise all of the services they provide. This makes using services easier, because more of the security, network address and permission configuration can be automated than with many other network types.

2.7 Specifications and features

The specifications were formalized by the Bluetooth Special Interest Group (SIG). The SIG was formally announced on 20 May 1998. Today it has a membership of over 30,000 companies worldwide. It was established by Ericsson, IBM, Intel, Toshiba and Nokia, and later joined by many other companies.

All versions of the Bluetooth standards support downward compatibility.That lets the latest standard cover all older versions.

The Bluetooth Core Specification Working Group (CSWG) produces mainly 4 kinds of specifications:

- The Bluetooth Core Specification, release cycle is typically a few years in between
- Core Specification Addendum (CSA), release cycle can be as tight as a few times per year
- Core Specification Supplements (CSS), can be released very quickly
- Errata (Available with a user account: Errata login)

Bluetooth 1.0 and 1.0B

Versions 1.0 and 1.0B had many problems, and manufacturers had difficulty making their products interoperable. Versions 1.0 and 1.0B also included mandatory Bluetooth hardware device address (BD_ADDR) transmission in the Connecting process (rendering anonymity impossible at the protocol level), which was a major setback for certain services planned for use in Bluetooth environments.

Bluetooth 1.1

- Ratified as IEEE Standard 802.15.1–2002
- Many errors found in the v1.0B specifications were fixed.
- Added possibility of non-encrypted channels.
- Received Signal Strength Indicator (RSSI).

Bluetooth 1.2

Major enhancements include:

- Faster Connection and Discovery
- Adaptive frequency-hopping spread spectrum *(AFH)*, which improves resistance to radio frequency interference by avoiding the use of crowded frequencies in the hopping sequence. • Higher transmission speeds in practice than in v1.1, up to 721 kbit/s.[47]
- Extended Synchronous Connections (eSCO), which improve voice quality of audio links by allowing retransmissions of corrupted packets, and may optionally increase audio latency to provide better concurrent data transfer.
- Host Controller Interface (HCI) operation with three-wire UART.
- Ratified as IEEE Standard 802.15.1–2005[48]
- Introduced Flow Control and Retransmission Modes for L2CAP.

Bluetooth 2.0 + EDR

This version of the Bluetooth Core Specification was released in 2004. The main difference is the introduction of an Enhanced Data Rate (EDR) for faster data transfer. The bit rate of EDR is 3 Mbit/s, although the maximum data transfer rate (allowing for inter-packet time and acknowledgements) is 2.1 Mbit/s. EDR uses a combination of GFSK and Phase Shift Keying modulation (PSK) with two variants, π/4-DQPSK and 8DPSK. EDR can provide a lower power consumption through a reduced duty cycle.

The specification is published as Bluetooth v2.0 + *EDR*, which implies that EDR is an optional feature. Aside from EDR, the v2.0 specification contains other minor improvements, and products may claim compliance to "Bluetooth v2.0" without supporting the higher data rate. At least one commercial device states "Bluetooth v2.0 without EDR" on its data sheet.

Bluetooth 2.1 + EDR

Bluetooth Core Specification Version 2.1 + EDR was adopted by the Bluetooth SIG on 26 July 2007.

The headline feature of v2.1 is secure simple pairing (SSP): this improves the pairing experience for Bluetooth devices, while increasing the use and strength of security. See the section on Pairing below for more details.

Version 2.1 allows various other improvements, including "Extended inquiry response" (EIR), which provides more information during the inquiry procedure to allow better filtering of devices before connection; and sniff subrating, which reduces the power consumption in low-power mode.

Bluetooth 3.0 + HS

Version 3.0 + HS of the Bluetooth Core Specification was adopted by the Bluetooth SIG on 21 April 2009. Bluetooth v3.0 + HS provides theoretical data transfer speeds of up to 24 Mbit/s, though not over the Bluetooth link itself. Instead, the Bluetooth link is used for negotiation and establishment, and the high data rate traffic is carried over a colocated 802.11 link.

The main new feature is AMP (Alternative MAC/PHY), the addition of 802.11 as a high-speed transport. The high-speed part of the specification is not mandatory, and hence only devices that display the "+HS" logo actually support Bluetooth over 802.11 high-speed data transfer. A Bluetooth v3.0 device without the "+HS" suffix is only required to support features introduced in Core Specification Version 3.0 or earlier Core Specification Addendum 1.

L2CAP Enhanced modes

Enhanced Retransmission Mode (ERTM) implements reliable L2CAP channel, while Streaming Mode (SM) implements unreliable channel with no retransmission or flow control. Introduced in Core Specification Addendum 1.

Alternative MAC/PHY

Enables the use of alternative MAC and PHYs for transporting Bluetooth profile data. The Bluetooth radio is still used for device discovery, initial connection and profile configuration. However, when large quantities of data must be sent, the high-speed alternative MAC PHY 802.11 (typically associated with Wi-Fi) transports the data. This means that Bluetooth uses proven low power connection models when the system is idle, and the faster radio when it must send large quantities of data. AMP links require enhanced L2CAP modes.

Unicast Connectionless Data

Permits sending service data without establishing an explicit L2CAP channel. It is intended for use by applications that require low latency between user action and reconnection/transmission of data. This is only appropriate for small amounts of data.

Enhanced Power Control

Updates the power control feature to remove the open loop power control, and also to clarify ambiguities in power control introduced by the new modulation schemes added for EDR. Enhanced power control removes the ambiguities by specifying the behavior that is expected. The feature also adds closed loop power control, meaning RSSI filtering can start as the response is received. Additionally, a "go straight to maximum power" request has been introduced. This is expected to deal with the headset link loss issue typically observed when a user puts their phone into a pocket on the opposite side to the headset.

Ultra-wideband

The high-speed (AMP) feature of Bluetooth v3.0 was originally intended for UWB, but the WiMedia Alliance, the body responsible for the flavor of UWB intended for Bluetooth, announced in March 2009 that it was disbanding, and ultimately UWB was omitted from the Core v3.0 specification.

On 16 March 2009, the WiMedia Alliance announced it was entering into technology transfer agreements for the WiMedia Ultra-wideband (UWB) specifications. WiMedia has transferred all current and future specifications, including work on future high-speed and power-optimized implementations, to the Bluetooth Special Interest Group (SIG), Wireless USB Promoter Group and the USB Implementers Forum. After successful completion of the technology transfer, marketing, and related administrative items, the WiMedia Alliance ceased operations.

In October 2009 the Bluetooth Special Interest Group suspended development of UWB as part of the alternative MAC/PHY, Bluetooth v3.0 + HS solution. A small, but significant, number of former WiMedia members had not and would not sign up to the necessary agreements for the IP transfer. The Bluetooth SIG is now in the process of evaluating other options for its longer term roadmap.

Bluetooth 4.0 + LE

Main article: Bluetooth Low Energy

The Bluetooth SIG completed the Bluetooth Core Specification version 4.0 (called Bluetooth Smart) and has been adopted as of 30 June 2010. It includes *Classic Bluetooth*, *Bluetooth high speed* and *Bluetooth Low Energy* protocols. Bluetooth high speed is based on Wi-Fi, and Classic Bluetooth consists of legacy Bluetooth protocols.

Bluetooth Low Energy, previously known as Wibree, is a subset of Bluetooth v4.0 with an entirely new protocol stack for rapid build-up of simple links. As an alternative to the Bluetooth standard protocols that were introduced in Bluetooth v1.0 to v3.0, it is aimed at very low power applications running off a coin cell. Chip designs allow for two types of implementation, dual mode, single-mode and enhanced past versions. The provisional names Wibree and Bluetooth ULP (Ultra Low Power) were abandoned and the BLE name was used for a while. In late 2011, new logos "Bluetooth Smart Ready" for hosts and "Bluetooth Smart" for sensors were introduced as the general-public face of BLE.

Compared to *Classic Bluetooth*, Bluetooth Low Energy is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range. In terms of lengthening the battery life of Bluetooth devices, BLE represents a significant progression.

- In a single-mode implementation, only the low energy protocol stack is implemented. Dialog Semiconductor STMicroelectronics, AMICCOM, CSR, Nordic Semiconductor and Texas Instruments have released single mode Bluetooth Low Energy solutions.
- In a dual-mode implementation, Bluetooth Smart functionality is integrated into an existing Classic Bluetooth controller. As of March 2011, the following semiconductor companies have announced the availability of chips meeting the standard: Qualcomm Atheros, CSR, Broadcom and Texas Instruments. The compliant architecture shares all of Classic Bluetooth’s existing radio and functionality resulting in a negligible cost increase compared to Classic Bluetooth.
- Cost-reduced single-mode chips, which enable highly integrated and compact devices, feature a lightweight Link Layer providing ultra-low power idle mode operation, simple

- device discovery, and reliable point-to-multipoint data transfer with advanced power-save and secure encrypted connections at the lowest possible cost.

General improvements in version 4.0 include the changes necessary to facilitate BLE modes, as well the Generic Attribute Profile (GATT) and Security Manager (SM) services with AES Encryption.

Core Specification Addendum 2 was unveiled in December 2011; it contains improvements to the audio Host Controller Interface and to the High Speed (802.11) Protocol Adaptation Layer.

Core Specification Addendum 3 revision 2 has an adoption date of 24 July 2012.

Core Specification Addendum 4 has an adoption date of 12 February 2013.

Bluetooth 4.1

The Bluetooth SIG announced formal adoption of the Bluetooth v4.1 specification on 4 December 2013. This specification is an incremental software update to Bluetooth Specification v4.0, and not a hardware update. The update incorporates Bluetooth Core Specification Addenda (CSA 1, 2, 3 & 4) and adds new features that improve consumer usability. These include increased co-existence support for LTE, bulk data exchange rates—and aid developer innovation by allowing devices to support multiple roles simultaneously.

New features of this specification include:

- Mobile Wireless Service Coexistence Signaling
- Train Nudging and Generalized Interlaced Scanning
- Low Duty Cycle Directed Advertising
- L2CAP Connection Oriented and Dedicated Channels with Credit Based Flow Control
- Dual Mode and Topology
- LE Link Layer Topology
- 802.11n PAL
- Audio Architecture Updates for Wide Band Speech
- Fast Data Advertising Interval
- Limited Discovery Time

Notice that some features were already available in a Core Specification Addendum (CSA) before the release of v4.1.

Bluetooth 4.2

Released on December 2, 2014, it introduces features for the Internet of Things.

The major areas of improvement are:

- Low Energy Secure Connection with Data Packet Length Extension
- Link Layer Privacy with Extended Scanner Filter Policies
- Internet Protocol Support Profile (IPSP) version 6 ready for Bluetooth Smart things to support connected home
- Older Bluetooth hardware may receive 4.2 features such as Data Packet Length Extension and improved privacy via firmware updates.

Bluetooth 5

The Bluetooth SIG officially unveiled Bluetooth 5 during a media event in London on 16 June 2016. Its new features are mainly focused on emerging Internet of Things technology. The Samsung Galaxy S8 launched with Bluetooth 5 support in April 2017. In September 2017, the iPhone 8, 8 Plus and iPhone X launched with Bluetooth 5 support as well. Apple also integrated 'Bluetooth 5.0' in their new HomePod offering released on February 9, 2018. Marketing drops the point number; so that it is just "Bluetooth 5" (not 5.0 or LE like Bluetooth 4.0). The change is for the sake of "Simplifying our marketing, communicating user benefits more effectively and making it easier to signal significant technology updates to the market."

Bluetooth 5 provides, for BLE, options that can double the speed (2 Mbit/s burst) at the expense of range, or up to fourfold the range at the expense of data rate, and eightfold the data broadcasting capacity of transmissions, by increasing the packet lengths. The increase in transmissions could be important for Internet of Things devices, where many nodes connect throughout a whole house. Bluetooth 5 adds functionality for connectionless services such as location-relevant navigation of low-energy Bluetooth connections.

The major areas of improvement are:

- Slot Availability Mask (SAM)
- 2 Mbit/s PHY for LE

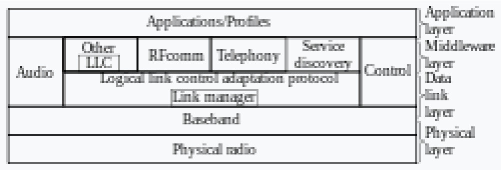
- LE Long Range
- High Duty Cycle Non-Connectable Advertising
- LE Advertising Extensions
- LE Channel Selection Algorithm #2

Features Added in CSA5- Integrated in v5.0:

- Higher Output Power

2.8 Technical information

Bluetooth protocol stack



Bluetooth Protocol Stack

Bluetooth is defined as a layer protocol architecture consisting of core protocols, cable replacement protocols, telephony control protocols, and adopted protocols. Mandatory protocols for all Bluetooth stacks are LMP, L2CAP and SDP. In addition, devices that communicate with Bluetooth almost universally can use these protocols: HCI and RFCOMM.

LMP

The *Link Management Protocol* (LMP) is used for set-up and control of the radio link between two devices (e.g., creating, modifying, and releasing logical transports and logical links, querying device abilities, "parking" devices, and power control.) LMP is carried over the ACL-C logical link of the default ACL logical transport, or over PSB. Implemented on the controller.

L2CAP

The *Logical Link Control and Adaptation Protocol* (L2CAP) is used to multiplex multiple logical connections between two devices using different higher level protocols. Provides segmentation and reassembly of on-air packets.

In *Basic* mode, L2CAP provides packets with a payload configurable up to 64 kB, with 672 bytes as the default MTU, and 48 bytes as the minimum mandatory supported MTU.

In Retransmission and Flow Control modes, L2CAP can be configured either for isochronous data or reliable data per channel by performing retransmissions and CRC checks.

Bluetooth Core Specification Addendum 1 adds two additional L2CAP modes to the core specification. These modes effectively deprecate original Retransmission and Flow Control modes:

- **Enhanced Retransmission Mode** (ERTM): This mode is an improved version of the original retransmission mode. This mode provides a reliable L2CAP channel.
- **Streaming Mode** (SM): This is a very simple mode, with no retransmission or flow control. This mode provides an unreliable L2CAP channel.

Reliability in any of these modes is optionally and/or additionally guaranteed by the lower layer Bluetooth BDR/EDR air interface by configuring the number of retransmissions and flush timeout (time after which the radio flushes packets). In-order sequencing is guaranteed by the lower layer.

Only L2CAP channels configured in ERTM or SM may be operated over AMP logical links.

SDP

The Service Discovery Protocol (SDP) allows a device to discover services offered by other devices, and their associated parameters. For example, when you use a mobile phone with a Bluetooth headset, the phone uses SDP to determine which Bluetooth profiles the headset can use (Headset Profile, Hands Free Profile, Advanced Audio Distribution Profile (A2DP) etc.) and the protocol multiplexer settings needed for the phone to connect to the headset using each of them. Each service is identified by a Universally Unique Identifier(UUID), with official services (Bluetooth profiles) assigned a short form UUID (16 bits rather than the full 128).

RFCOMM

Radio Frequency Communications (RFCOMM) is a cable replacement protocol used for generating a virtual serial data stream. RFCOMM provides for binary data transport and emulates EIA-232 (formerly RS-232) control signals over the Bluetooth baseband layer, i.e. it is a serial port emulation.

RFCOMM provides a simple reliable data stream to the user, similar to TCP. It is used directly by many telephony related profiles as a carrier for AT commands, as well as being a transport layer for OBEX over Bluetooth.

Many Bluetooth applications use RFCOMM because of its widespread support and publicly available API on most operating systems. Additionally, applications that used a serial port to communicate can be quickly ported to use RFCOMM.

BNEP

The Bluetooth Network Encapsulation Protocol (BNEP) is used for transferring another protocol stack's data via an L2CAP channel. Its main purpose is the transmission of IP packets in the Personal Area Networking Profile. BNEP performs a similar function to SNAP in Wireless LAN.

AVCTP

The Audio/Video Control Transport Protocol (AVCTP) is used by the remote control profile to transfer AV/C commands over an L2CAP channel. The music control buttons on a stereo headset use this protocol to control the music player.

AVDTP

The Audio/Video Distribution Transport Protocol (AVDTP) is used by the advanced audio distribution profile to stream music to stereo headsets over an L2CAP channel intended for video distribution profile in the Bluetooth transmission.

TCS

The Telephony Control Protocol – Binary (TCS BIN) is the bit-oriented protocol that defines the call control signaling for the establishment of voice and data calls between Bluetooth devices. Additionally, "TCS BIN defines mobility management procedures for handling groups of Bluetooth TCS devices."

TCS-BIN is only used by the cordless telephony profile, which failed to attract implementers. As such it is only of historical interest.

Adopted protocols

Adopted protocols are defined by other standards-making organizations and incorporated into Bluetooth's protocol stack, allowing Bluetooth to code protocols only when necessary. The adopted protocols include:

Point-to-Point Protocol (PPP)

Internet standard protocol for transporting IP datagrams over a point-to-point link.

TCP/IP/UDP

Foundation Protocols for TCP/IP protocol suite

Object Exchange Protocol (OBEX)

Session-layer protocol for the exchange of objects, providing a model for object and operation representation

Wireless Application Environment/Wireless Application Protocol (WAE/WAP)

WAE specifies an application framework for wireless devices and WAP is an open standard to provide mobile users access to telephony and information services. **Baseband error correction**

Depending on packet type, individual packets may be protected by error correction, either 1/3 rate forward error correction (FEC) or 2/3 rate. In addition, packets with CRC will be retransmitted until acknowledged by automatic repeat request (ARQ).

Setting up connections

Any Bluetooth device in discoverable mode transmits the following information on demand:

- Device name
- Device class
- List of services

- Technical information (for example: device features, manufacturer, Bluetooth specification used, clock offset)

Any device may perform an inquiry to find other devices to connect to, and any device can be configured to respond to such inquiries. However, if the device trying to connect knows the address of the device, it always responds to direct connection requests and transmits the information shown in the list above if requested. Use of a device's services may require pairing or acceptance by its owner, but the connection itself can be initiated by any device and held until it goes out of range. Some devices can be connected to only one device at a time, and connecting to them prevents them from connecting to other devices and appearing in inquiries until they disconnect from the other device.

Every device has a unique 48-bit address. However, these addresses are generally not shown in inquiries. Instead, friendly Bluetooth names are used, which can be set by the user. This name appears when another user scans for devices and in lists of paired devices.

Most cellular phones have the Bluetooth name set to the manufacturer and model of the phone by default. Most cellular phones and laptops show only the Bluetooth names and special programs are required to get additional information about remote devices. This can be confusing as, for example, there could be several cellular phones in range named T610(see Bluejacking).

2.9 Pairing and bonding

Motivation

Many services offered over Bluetooth can expose private data or let a connecting party control the Bluetooth device. Security reasons make it necessary to recognize specific devices, and thus enable control over which devices can connect to a given Bluetooth device. At the same time, it is useful for Bluetooth devices to be able to establish a connection without user intervention (for example, as soon as in range).

To resolve this conflict, Bluetooth uses a process called *bonding*, and a bond is generated through a process called *pairing*. The pairing process is triggered either by a specific request from a user

to generate a bond (for example, the user explicitly requests to "Add a Bluetooth device"), or it is triggered automatically when connecting to a service where (for the first time) the identity of a device is required for security purposes. These two cases are referred to as dedicated bonding and general bonding respectively.

Pairing often involves some level of user interaction. This user interaction confirms the identity of the devices. When pairing successfully completes, a bond forms between the two devices, enabling those two devices to connect to each other in the future without repeating the pairing process to confirm device identities. When desired, the user can remove the bonding relationship.

Implementation

During pairing, the two devices establish a relationship by creating a shared secret known as a *link key*. If both devices store the same link key, they are said to be *paired* or *bonded*. A device that wants to communicate only with a bonded device can cryptographically authenticate the identity of the other device, ensuring it is the same device it previously paired with. Once a link key is generated, an authenticated Asynchronous Connection-Less (ACL) link between the devices may be encrypted to protect exchanged data against eavesdropping. Users can delete link keys from either device, which removes the bond between the devices—so it is possible for one device to have a stored link key for a device it is no longer paired with.

Bluetooth services generally require either encryption or authentication and as such require pairing before they let a remote device connect.

Pairing mechanisms

Pairing mechanisms changed significantly with the introduction of Secure Simple Pairing in Bluetooth v2.1. The following summarizes the pairing mechanisms:

- Legacy pairing: This is the only method available in Bluetooth v2.0 and before. Each device must enter a PIN code; pairing is only successful if both devices enter the same PIN code. Any 16-byte UTF-8 string may be used as a PIN code; however, not all devices may be capable of entering all possible PIN codes.
- Limited input devices: The obvious example of this class of device is a Bluetooth Handsfree headset, which generally have few inputs. These devices usually have a *fixed PIN*, for example "0000" or "1234", that are hard-coded into the device.
- Numeric input devices: Mobile phones are classic examples of these devices.

They allow a user to enter a numeric value up to 16 digits in length.

- Alpha-numeric input devices: PCs and smartphones are examples of these devices. They allow a user to enter full UTF-8 text as a PIN code. If pairing with a less capable device the user must be aware of the input limitations on the other device; there is no mechanism available for a capable device to determine how it should limit the available input a user may use.

- Secure Simple Pairing (SSP): This is required by Bluetooth v2.1, although a Bluetooth v2.1 device may only use legacy pairing to interoperate with a v2.0 or earlier device. Secure Simple Pairing uses a form of public key cryptography, and some types can help protect against man in the middle, or MITM attacks. SSP has the following authentication mechanisms:
- Just works: As the name implies, this method just works, with no user interaction. However, a device may prompt the user to confirm the pairing process. This method is typically used by headsets with very limited IO capabilities, and is more secure than the fixed PIN mechanism this limited set of devices uses for legacy pairing. This method provides no man-in-the-middle (MITM) protection.
- Numeric comparison: If both devices have a display, and at least one can accept a binary yes/no user input, they may use Numeric Comparison. This method displays a 6digit numeric code on each device. The user should compare the numbers to ensure they are identical. If the comparison succeeds, the user(s) should confirm pairing on the device(s) that can accept an input. This method provides MITM protection, assuming the user confirms on both devices and actually performs the comparison properly.

- Passkey Entry: This method may be used between a device with a display and a device with numeric keypad entry (such as a keyboard), or two devices with numeric keypad entry. In the first case, the display presents a 6-digit numeric code to the user, who then enters the code on the keypad. In the second case, the user of each device enters the same 6-digit number. Both of these cases provide MITM protection.

- Out of band (OOB): This method uses an external means of communication, such as near-field communication (NFC) to exchange some information used in the pairing process. Pairing is completed using the Bluetooth radio, but requires information from the OOB mechanism. This provides only the level of MITM protection that is present in the OOB mechanism.
-

SSP is considered simple for the following reasons:

- In most cases, it does not require a user to generate a passkey.
- For use cases not requiring MITM protection, user interaction can be eliminated.
- For *numeric comparison*, MITM protection can be achieved with a simple equality comparison by the user.
- Using OOB with NFC enables pairing when devices simply get close, rather than requiring a lengthy discovery process.

2.10 Security concerns

Prior to Bluetooth v2.1, encryption is not required and can be turned off at any time. Moreover, the encryption key is only good for approximately 23.5 hours; using a single encryption key longer than this time allows simple XOR attacks to retrieve the encryption key.

- Turning off encryption is required for several normal operations, so it is problematic to detect if encryption is disabled for a valid reason or for a security attack.

Bluetooth v2.1 addresses this in the following ways:

- Encryption is required for all non-SDP (Service Discovery Protocol) connections
- A new Encryption Pause and Resume feature is used for all normal operations that require that encryption be disabled. This enables easy identification of normal operation from security attacks.
- The encryption key must be refreshed before it expires.

Link keys may be stored on the device file system, not on the Bluetooth chip itself. Many Bluetooth chip manufacturers let link keys be stored on the device—however, if the device is removable, this means that the link key moves with the device.

2.11 Security

Overview

See also: Mobile security § Attacks based on communication networks.

Bluetooth implements confidentiality, authentication and key derivation with custom algorithms based on the SAFER+ block cipher. Bluetooth key generation is generally based on a Bluetooth PIN, which must be entered into both devices. This procedure might be modified if one of the devices has a fixed PIN (e.g., for headsets or similar devices with a restricted user interface). During pairing, an initialization key or master key is generated, using the E22 algorithm. The E0 stream cipher is used for encrypting packets, granting confidentiality, and is based on a shared cryptographic secret, namely a previously generated link key or master key. Those keys, used for subsequent encryption of data sent via the air interface, rely on the Bluetooth PIN, which has been entered into one or both devices.

An overview of Bluetooth vulnerabilities exploits was published in 2007 by Andreas Becker.

In September 2008, the National Institute of Standards and Technology (NIST) published a Guide to Bluetooth Security as a reference for organizations. It describes Bluetooth security capabilities and how to secure Bluetooth technologies effectively. While Bluetooth has its benefits, it is susceptible to denial-of-service attacks, eavesdropping, man-in-the-middle attacks, message modification, and resource misappropriation. Users and organizations must evaluate their acceptable level of risk and incorporate security into the lifecycle of Bluetooth devices. To help mitigate risks, included in the NIST document are security checklists with guidelines and recommendations for creating and maintaining secure Bluetooth piconets, headsets, and smart card readers.

Bluetooth v2.1 – finalized in 2007 with consumer devices first appearing in 2009 – makes significant changes to Bluetooth's security, including pairing. See the pairing mechanisms section for more about these changes.

Bluejacking

Bluejacking is the sending of either a picture or a message from one user to an unsuspecting user through Bluetooth wireless technology. Common applications include short messages.

Health concerns

Main article: Wireless electronic devices and health

Bluetooth uses the microwave radio frequency spectrum in the 2.402 GHz to 2.480 GHz range,] which is non-ionizing radiation, of similar bandwidth to the one used by wireless and mobile phones. No specific demonstration of harm has been demonstrated up to date, even if wireless transmission has been included by IARC in the possible carcinogen list. Maximum power output from a Bluetooth radio is 100 mW for class 1, 2.5 mW for class 2, and 1 mW for class 3 devices. Even the maximum power output of class 1 is a lower level than the lowestpowered mobile phones. UMTS and W-CDMA output 250 mW, GSM1800/1900 outputs 1000 mW, and GSM850/900 outputs 2000 mW.

Bluetooth award programs

The Bluetooth Innovation World Cup, a marketing initiative of the Bluetooth Special Interest Group (SIG), was an international competition that encouraged the development of innovations for applications leveraging Bluetooth technology in sports, fitness and health care products. The aim of the competition was to stimulate new markets.

The Bluetooth Innovation World Cup morphed into the Bluetooth Breakthrough Awards in 2013. The Breakthrough Awards. Bluetooth program highlights the most innovative products and applications available today, prototypes coming soon, and student-led projects in the making.

3 RFCOMM

Acronym Definition

RFCOMM	Radio Frequency Communication*(s)*

3.1 Introduction

The RFCOMM protocol provides emulation of serial ports over the L2CAP protocol. The protocol is based on the ETSI standard TS 07.10. This document does not contain a complete specification. Instead, references are made to the relevant parts of the TS 07.10 standard. Only a subset of the TS 07.10 standard is used, and some adaptations of the protocol are specified in this document.

RFCOMM is a simple transport protocol, which provides emulation of RS232 serial ports over the L2CAP protocol.The protocol is based on the ETSI standard TS 07.10. Only a subset of the TS 07.10 standard is used and an RFCOMM - specific extension is added, in the form of a mandatory credit based flow control scheme.

The RFCOMM protocol supports up to 60 simultaneous connections between two BT devices. The number of connections that can be used simultaneously in a BT device is implementation specific. For the purposes of RFCOMM, a complete communication path involves two applications running on different devices (the communication endpoints) with a communication segment between them.

3.1.1 OVERVIEW

RFCOMM is a simple transport protocol, with additional provisions for emulating the 9 circuits of RS-232 (EIATIA-232-E) serial ports. The RFCOMM protocol supports up to 60 simultaneous connections between two BT devices. The number of connections that can be used simultaneously in a BT device is implementation-specific.

3.1.2 DEVICE TYPES

For the purposes of RFCOMM, a complete communication path involves two applications running on different devices (the communication endpoints) with a communication segment between them. Figure shows the complete communication path.(In this context, the term application may mean other things than end-user application; e.g. higher layer protocols or other services acting on behalf of end-user applications.)

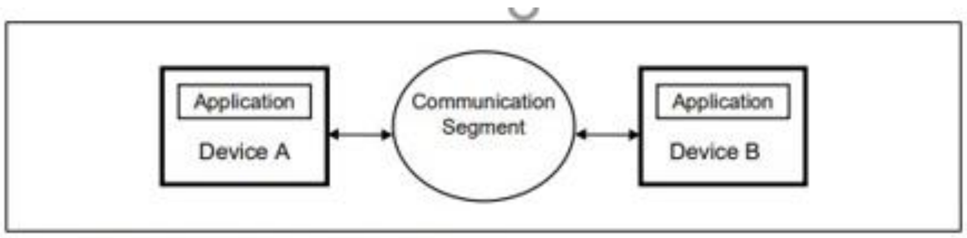
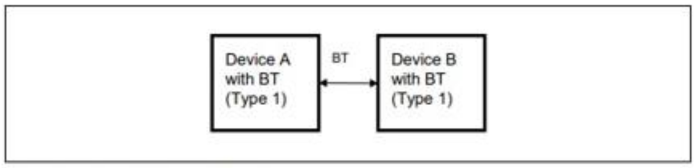


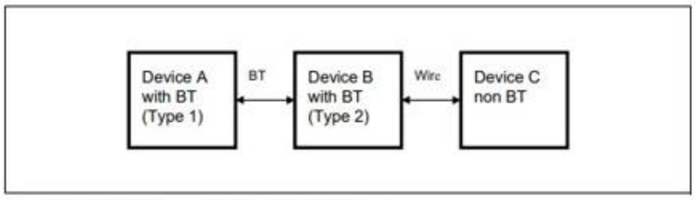
FIGURE 4 RFCOMM Communication Segment

Segment RFCOMM is intended to cover applications that make use of the serial ports of the devices in which they reside In the simple configuration, the communication segment is BT link from one device to another (direct connect), see Figure. Where the communication segment is another network, BT is used for the path between the device and a network connection device like a modem.

RFCOMM is only concerned with the connection between the devices in the direct connect case, or between the device and a mode in the network case. RFCOMM can support other configurations, such as modules that communicate via BT on one side and provide a wired interface on the other side, as shown in Figure. These devices are not really modems but offer a similar service. They are therefore not explicitly discussed here.

Basically two device types exist that RFCOMM must accommodate. Type 1 devices are communication end points such as computers and printers. Type 2 devices are those that are part of the communication segment; e.g. modems. Though RFCOMM does not make a distinction between these two device types in the protocol, accommodating both types of devices impacts the RFCOMM protocol.





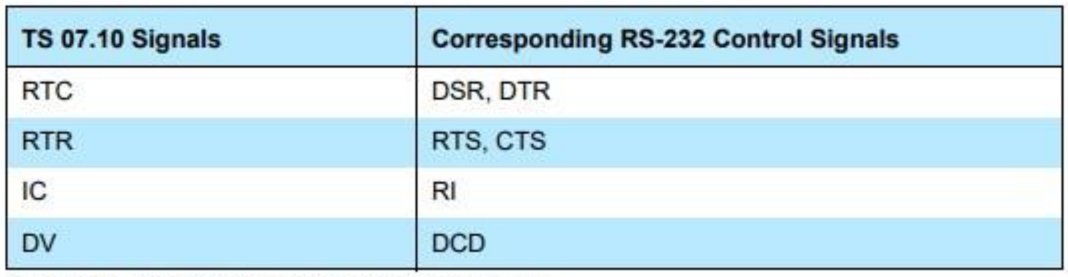
The information transferred between two RFCOMM entities has been defined to support both type 1 and type 2 devices. Some information is only needed by type 2 devices while other information is intended to be used by both. In the protocol, no distinction is made between type 1 and type 2. It is therefore up to the RFCOMM implementers to determine if the information passed in the RFCOMM protocol is of use to the implementation. Since the device is not aware of the type of the other device in the communication path, each must pass on all available information specified by the protocol. 1.3 BYTE ORDERING This document uses the same byte ordering as the TS 07.10 specification; i.e. all binary numbers are in Least Significant Bit to Most Significant Bit order, reading from left to right.

3.2 RFCOMM SERVICE OVERVIEW

RFCOMM emulates RS-232 (EIATIA-232-E) serial ports. The emulation includes transfer of the state of the non-data circuits. RFCOMM has a built-in scheme for null modem emulation. In the event that a baud rate is set for a particular port through the RFCOMM service interface, that will not affect the actual data throughput in RFCOMM; i.e. RFCOMM does not incur artificial rate limitation or pacing. However, if either device is a type 2 device (relays data onto other media), or if data pacing is done above the RFCOMM service interface in either or both ends, actual throughput will, on an average, reflect the baud rate setting. RFCOMM supports emulation of multiple serial ports between two devices and also emulation of serial ports between multiple devices.

3.2.1 NULL MODEM EMULATION

RFCOMM is based on TS 07.10. When it comes to transfer of the states of the non-data circuits, TS 07.10 does not distinguish between DTE and DCE devices. The RS-232 control signals are sent as a number of DTE/DCE independent signals, see Table 2.2.



TS 07.10 Signals	Corresponding RS-232 Control Signals
RTC	DSR, DTR
RTR	RTS, CTS
IC	RI
DV	DCD

Table 2.2: TS 07.10 Serial Port Control Signals

The way in which TS 07.10 transfers the RS-232 control signals creates an implicit null modem when two devices of the same kind are connected together. Figure 2.1 shows the null modem that is created when two DTE are connected via RFCOMM. No single null-modem cable wiring

scheme works in all cases; however the null modem scheme provided in RFCOMM should work in most cases.

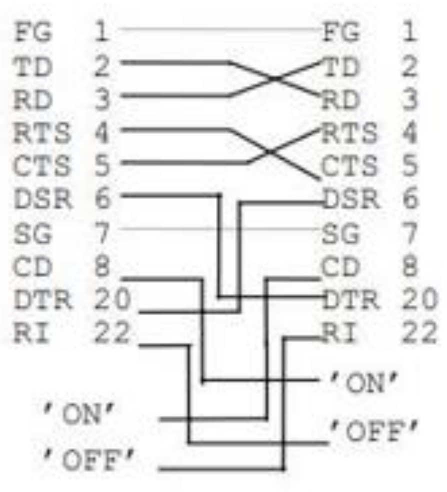


FIGURE 5 : NULL MODEM EMULATION

3.3 MULTIPLE EMULATED SERIAL PORTS

3.3.1 MULTIPLE EMULATED SERIAL PORTS BETWEEN TWO DEVICES

Two BT devices using RFCOMM in their communication may open multiple emulated serial ports. RFCOMM supports up to 60 open emulated ports; however the number of ports that can be used in a device is implementationspecific. A Data Link Connection Identifier (DLCI) [1] identifies an ongoing connection between a client and a server application. The DLCI is represented by 6 bits, but its usable value range is 2…61; in TS 07.10, DLCI 0 is the dedicated control channel, DLCI 1 is unusable due to the concept of Server Channels, and DLCI 62-63 is reserved. The DLCI is unique within one RFCOMM session between two devices. (This is explained further in Section 2.3.2) To account for the fact that both client and server applications may reside on both sides of an RFCOMM session, with clients on either side making connections independent of each other, the DLCI value space is divided between the two communicating devices using the concept of RFCOMM server channels. This is further described in Section 5.4.

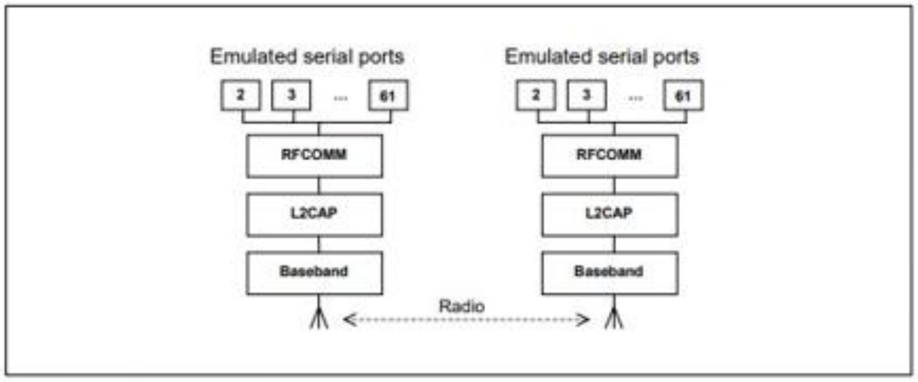


FIGURE 6 Multiple Emulated Serial Ports

3.3.2 MULTIPLE EMULATED SERIAL PORTS AND MULTIPLE BT DEVICES

If a BT device supports multiple emulated serial ports and the connections are allowed to have endpoints in different BT devices, then the RFCOMM entity must be able to run multiple TS 07.10 multiplexer sessions, see Figure 2.3. Note that each multiplexer session is using its own L2CAP channel ID (CID). The ability to run multiple sessions of the TS 07.10 multiplexer is optional for RFCOMM.

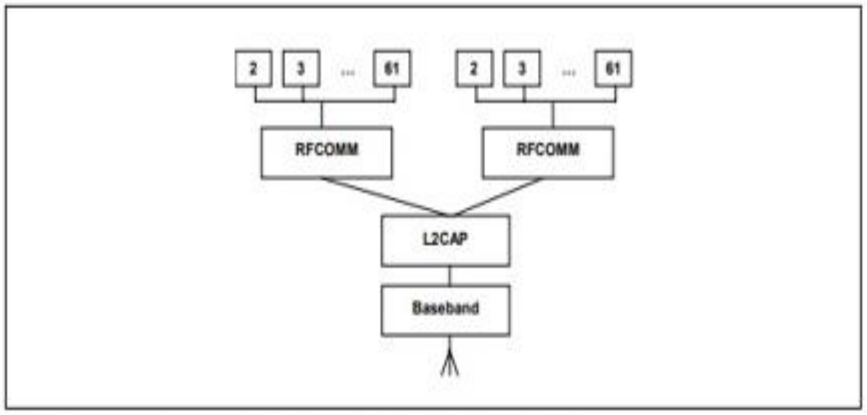


FIGURE 7 : Emulating serial ports coming from two BT devices

4 SERVICE INTERFACE DESCRIPTION

RFCOMM is intended to define a protocol that can be used to emulate serial ports. In most systems, RFCOMM will be part of a port driver which includes a serial port emulation entity.

4.1 SERVICE DEFINITION MODEL

The figure below shows a model of how RFCOMM fits into a typical system. This figure represents the RFCOMM reference model.

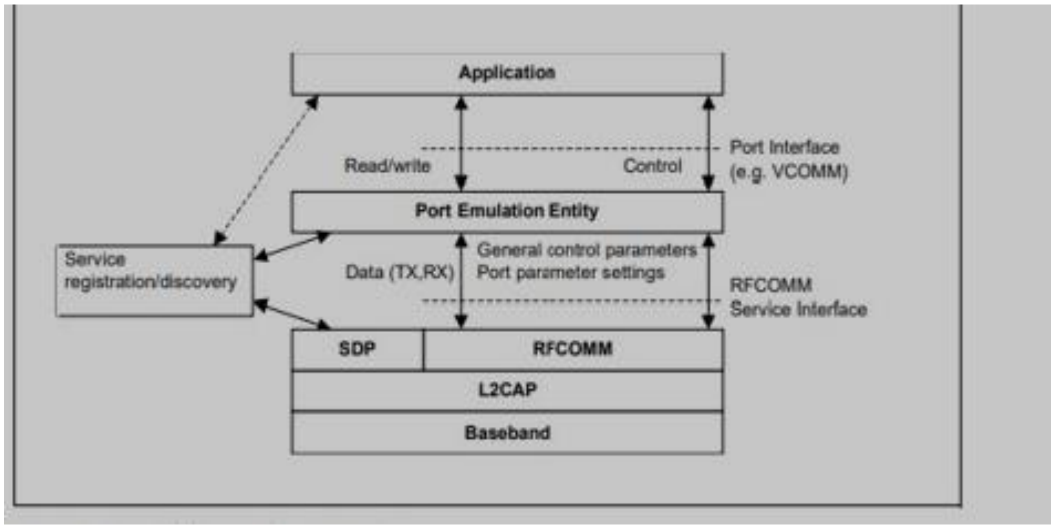
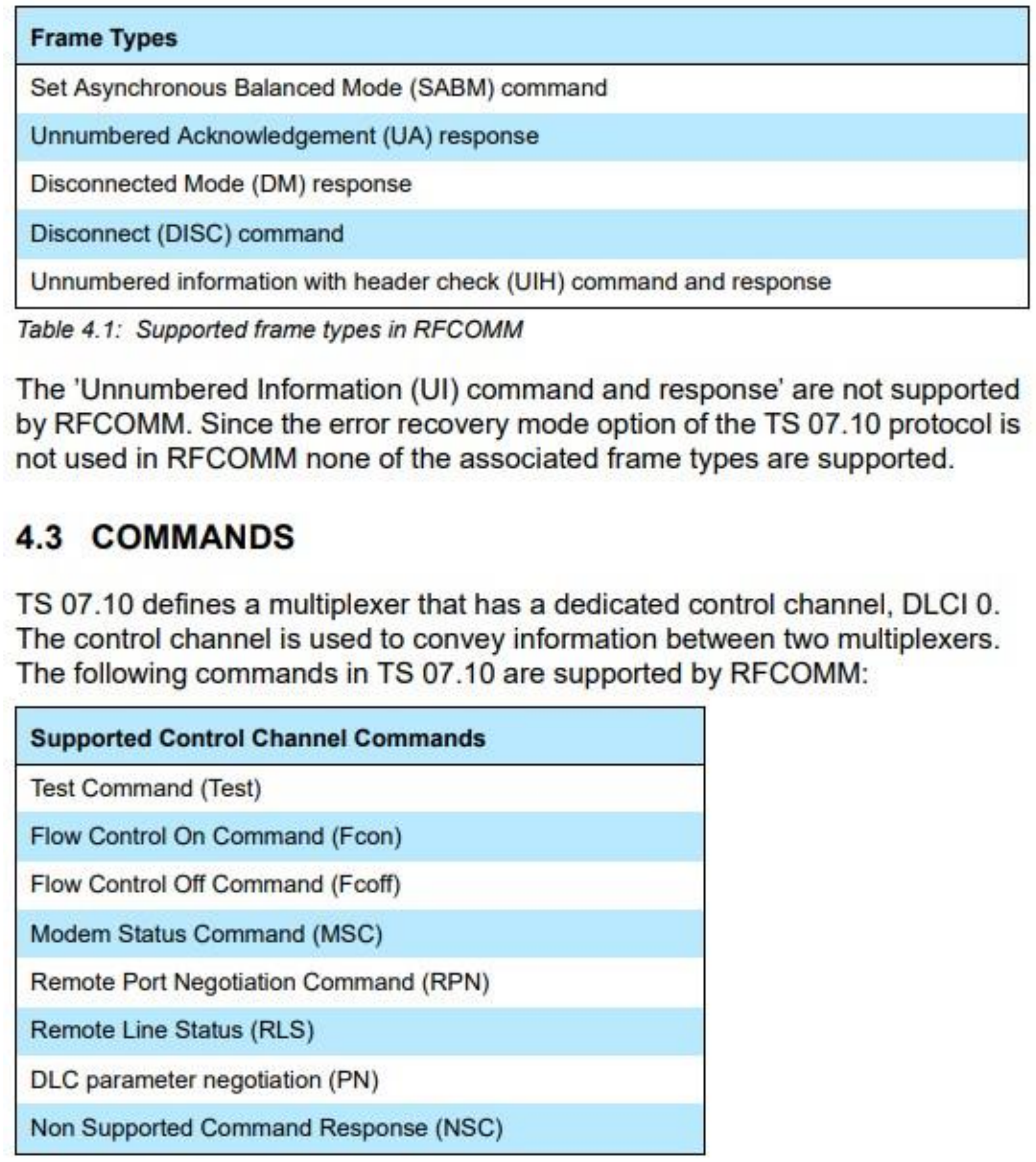


FIGURE 8 : RFCOMM reference model

4.2 FRAME TYPES Table 4.1 shows the TS 7.10 frame types that are supported in RFCOMM.



Frame Types
Set Asynchronous Balanced Mode (SABM) command
Unnumbered Acknowledgement (UA) response
Disconnected Mode (DM) response
Disconnect (DISC) command
Unnumbered information with header check (UIH) command and response

Table 4.1: Supported frame types in RFCOMM

The 'Unnumbered Information (UI) command and response' are not supported by RFCOMM. Since the error recovery mode option of the TS 07.10 protocol is not used in RFCOMM none of the associated frame types are supported.

4.3 COMMANDS

TS 07.10 defines a multiplexer that has a dedicated control channel, DLCI 0. The control channel is used to convey information between two multiplexers. The following commands in TS 07.10 are supported by RFCOMM:

Supported Control Channel Commands
Test Command (Test)
Flow Control On Command (Fcon)
Flow Control Off Command (Fcoff)
Modem Status Command (MSC)
Remote Port Negotiation Command (RPN)
Remote Line Status (RLS)
DLC parameter negotiation (PN)
Non Supported Command Response (NSC)

4.4 CONVERGENCE LAYERS

RFCOMM only supports the type 1 convergence layer in TS 07.10. The Modem Status Command (MSC) shall be used to convey the RS-232 control signals and the break signal for all emulated serial ports.

5 TS 07.10 ADAPTATIONS FOR RFCOMM

5.1 MEDIA ADAPTATION

The opening flag and the closing flags in the 07.10 basic option frame are not used in RFCOMM,instead it is only the fields contained between the flags that are exchanged between the L2CAP layer and RFCOMM layer

5.1.1 FCS calculation

In 07.10, the frame check sequence(FCS) is calculated on different sets of fields for different frame types.These are the fields that the FCS are calculated on:

For SABM, DISC, UA, DM frames: on Address,Control and length field.
For UIH frames : on Address and Control field

5.2 TS 07.10 MULTIPLEXER START-UP AND CLOSEDOWN PROCEDURE

The start-up and closedown procedures as specified in section 5.7 in TS 07.10 are not supported. This means that the AT-command AT+CMUX is not supported by RFCOMM, neither is the multiplexer close down (CLD) command. At any time, there must be at most one RFCOMM session between any pair of devices. When establishing a new DLC, the initiating entity must check if there already exists an RFCOMM session with the remote device, and if so, establish the new DLC on that. A session is identified by the Bluetooth BD_ADDR of the two endpoints1.

5.2.1 Start-up procedure

The device opening up the first emulated serial port connection between two devices is responsible for first establishing the multiplexer control channel. This involves the following steps:

• Establish an L2CAP channel to the peer RFCOMM entity, using L2CAP service primitives, see L2CAP "Service Primitives" on p • Start the RFCOMM multiplexer by sending SABM command on DLCI 0, and await UA response from peer entity. (Further optional negotiation steps are possible.) After these steps, DLCs for user data traffic can be established.

5.2.2 Close-down procedure

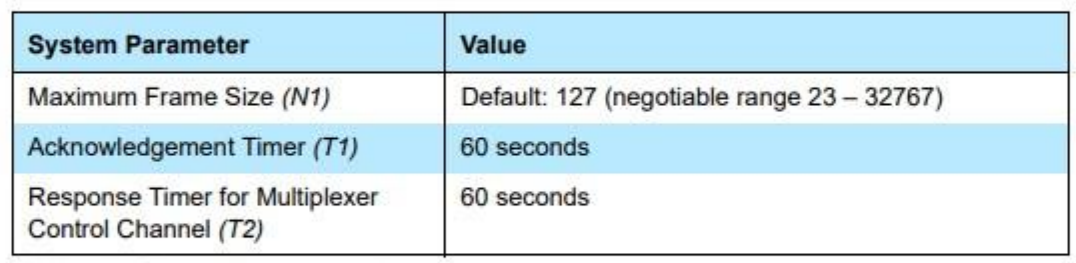
The device closing the last connection (DLC) on a particular session is responsible for closing the multiplexer by closing the corresponding L2CAP channel. Closing the multiplexer by first sending a DISC command frame on DLCI 0 is optional, but it is mandatory to respond correctly to a DISC (with UA response).

5.2.3 Link loss handling

If an L2CAP link loss notification is received, the local RFCOMM entity is responsible for sending a connection loss notification to the port emulation/ proxy entity for each active DLC. Then all resources associated with the RFCOMM session should be freed. The appropriate action to take in the port emulation/proxy entity depends on the API on top. For example, for an emulated serial port (vCOMM), it would be suitable to drop CD, DSR and CTS signals (assuming device is a DTE).

5.3 SYSTEM PARAMETERS

Table 5.1 contains all the applicable system parameters for the RFCOMM implementation of the TS 07.10 multiplexer.



System Parameter	Value
Maximum Frame Size *(N1)*	Default: 127 (negotiable range 23 – 32767)
Acknowledgement Timer *(T1)*	60 seconds
Response Timer for Multiplexer Control Channel *(T2)*	60 seconds

Table 5.1: System parameter values

Note: The timer T1 is the timeout for frames sent with the P/F-bit set to one (this applies only to

SABM and DISC frames in RFCOMM). T2 is the timeout for commands sent in UIH frames on DLCI 0. Since RFCOMM relies on lower layers to provide reliable transmission, the default action performed on timeouts is to close down the multiplexer session. The only exception to this is when trying to set up a new DLC on an existing session; i.e. waiting for the UA response for a SABM

command. In this case, the initiating side may defer the timeout by an unspecified amount of time if it has knowledge that the delay is due to user interaction (e.g. authentication procedure in progress). When/if the connection attempt is eventually considered to have timed out, the initiating side must send a DISC command frame on the same DLCI as the original SABM command frame, in order to notify the other party that the connection attempt is aborted.
(After that the initiating side will, as usual, expect a UA response for the DISC command.)

5.4 MULTIPLEXER CONTROL COMMANDS

Note that in TS 07.10, some Multiplexer Control commands pertaining to specific DLCIs may be exchanged on the control channel (DLCI 0) before the corresponding DLC has been established. (This refers the PN and RPN commands.) All such states associated with an individual DLC must be reset to their default values upon receiving a DISC command frame, or when closing the DLC from the local side. This is to ensure that all DLC (re-)establishments on the same session will have predictable results, irrespective of the session history. 5.5.1 Remote Port Negotiation Command (RPN) The RPN command can be used before a new DLC is opened and should be used whenever the port settings change. The RPN command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. (Although the handling of individual settings are implementation-dependent.)

5.4.1 Remote Line Status Command (RLS)

This command is used for indication of remote port line status. The RLS command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. (Although the handling of individual settings are implementation-dependent.) 5.5.3 DLC parameter negotiation (PN) The PN command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. This command can be used before a new DLC is opened.

There are some parameters in the PN command which convey information not applicable to RFCOMM. These fields must therefore be set to predetermined values by the sender, and they must be ignored by the receiver. This concern the following fields (see table 3 in ref. [1]): • I1-I4 must be set to 0. (Meaning: use UIH frames.) • CL1-CL4 must be set to 0. (Meaning: use convergence layer type 1.) • T1-T8 must be set to 0. (Meaning: acknowledgment timer T1, which is not negotiable in RFCOMM.) • NA1-NA8 must be set to 0. (Meaning: number of retransmissions N2; always 0 for RFCOMM) • K1-K3 must be set to 0. (Meaning: defines the window size for error recovery mode, which is not used for RFCOMM.)

If a command is received with invalid (or for some reason unacceptable) values in any field, a DLC parameter negotiation response must be issued with values that are acceptable to the responding device.

6 FLOW CONTROL

Wired ports commonly use flow control such as RTS/CTS to control communications. On the other hand, the flow control between RFCOMM and the lower layer L2CAP depends on the service interface supported by the implementation. In addition RFCOMM has its own flow control mechanisms. This section describes the different flow control mechanisms.

6.1 L2CAP FLOW CONTROL IN OVERVIEW

L2CAP relies on the flow control mechanism provided by the Link Manager layer in the baseband. The flow control mechanism between the L2CAP and RFCOMM layers is implementation-specific.

6.2 WIRED SERIAL PORT FLOW CONTROL

Wired Serial ports falls into two camps – software flow control using characters such as XON/XOFF, and flow control using RTS/CTS or DTR/DSR circuits. These methods may be used by both sides of a wired link, or may be used only in one direction.

6.3 RFCOMM FLOW CONTROL

The RFCOMM protocol provides two flow control mechanisms: 1. The RFCOMM protocol contains flow control commands that operate on the aggregate data flow between two RFCOMM entities; i.e. all DLCIs are affected. The control channel commands, FCon and FCoff, are defined in section 5.4.6.3 in ref [1]. 2. The Modem Status command as defined in section 5.4.6.3 in ref [1] is the flow control mechanism that operates on individual DLCI.

6.4 PORT EMULATION ENTITY SERIAL FLOW CONTROL

On Type 1 devices some port drivers (Port Emulation Entities plus RFCOMM) will need to provide flow control services as specified by the API they are emulating. An application may

request a particular flow control mechanism like XON/XOFF or RTS/CTS and expect the port driver to handle the flow control. On type 2 devices the port driver may need to perform flow control on the nonRFCOMM part of the communication path; i.e. the physical RS-232 port. This flow control is specified via the control parameters sent by the peer RFCOMM entity (usually a type 1 device). The description of flow control in this section is for port drivers on type 1 devices. Since RFCOMM already has its own flow control mechanism, the port driver does not need to perform flow control using the methods requested by the application. In the ideal case, the application sets a flow control mechanism and assumes that the COMM system will handle the details. The port driver could then simply ignore the request and rely on RFCOMM's flow control. The application is able to send and receive data, and does not know or care that the port driver did not perform flow control using the mechanism requested. However, in the real world some problems arise. • The RFCOMM-based port driver is running on top of a packet-based protocol where data may be buffered somewhere in the communication path. Thus, the port driver cannot perform flow control with the same precision as in the wired case. • The application may decide to apply the flow control mechanism itself in addition to requesting flow control from the port driver. These problems suggest that the port driver must do some additional work to perform flow control emulation properly. Here are the basic rules for flow control emulation. • The port driver will not solely rely on the mechanism requested by the application but use a combination of flow control mechanisms. • The port driver must be aware of the flow control mechanisms requested by the application and behave like the wired case when it sees changes on the non-data circuits (hardware flow control) or flow control characters in the incoming data (software flow control).

For example, if XOFF and XON characters would have been stripped in the wired case they must be stripped by the RFCOMM based port driver. • If the application sets a flow control mechanism via the port driver interface and then proceeds to invoke the mechanism on its own, the port driver must behave in a manner similar to that of the wired case (e.g. If XOFF and XON characters would have been passed through to the wire in the wired case the port driver must also pass these characters). These basic rules are applied to emulate each of the wired flow control schemes. Note that multiple types of flow control can be set at the same time. Section 5.4.8 in ref [1] defines each flow control mechanism.

7 INTERACTION WITH OTHER ENTITIES

7.1 PORT EMULATION AND PORT PROXY ENTITIES

This section defines how the RFCOMM protocol should be used to emulate serial ports. Figure 7.1 shows the two device types that the RFCOMM protocol supports

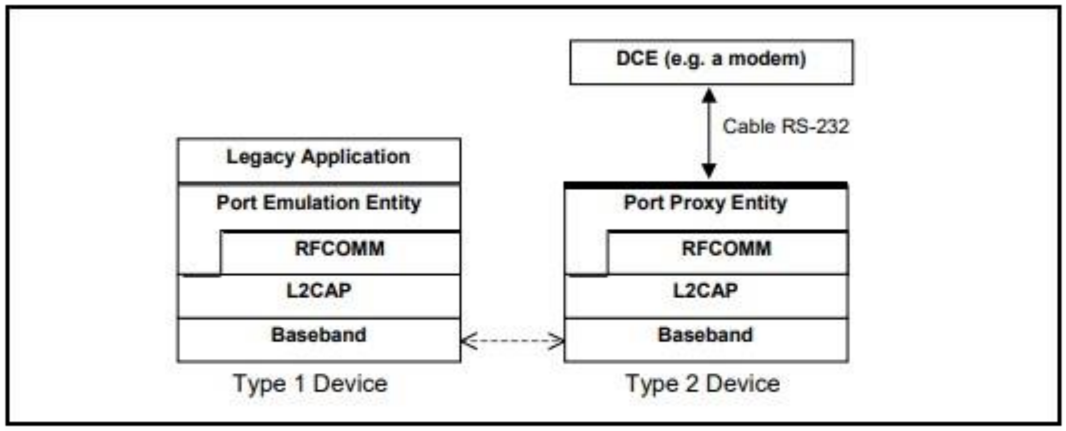


Figure 7.1: The RFCOMM communication model

Type 1 devices are communication endpoints such as computers and printers.
Type 2 devices are part of a communication segment; e.g. modems.

7.1.1 Port Emulation Entity

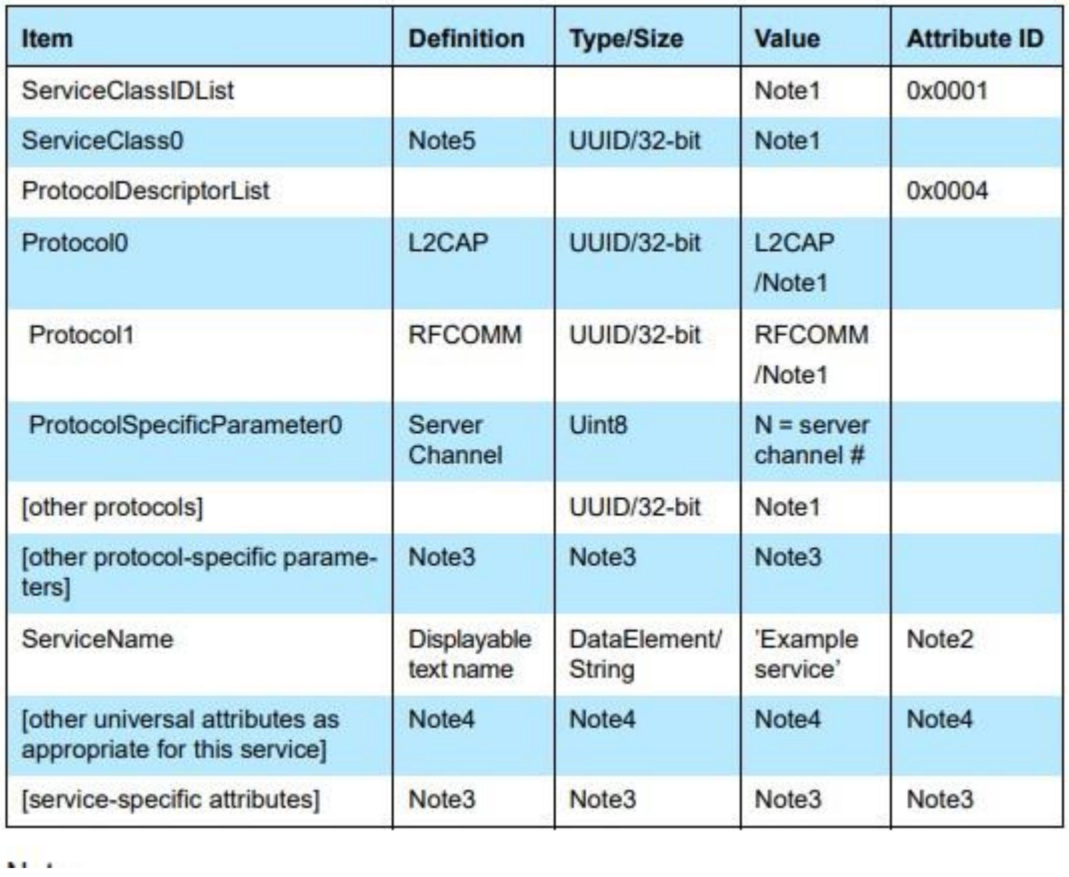
The port emulation entity maps a system specific communication interface (API) to the RFCOMM services.

7.1.2 Port Proxy Entity

The port proxy entity relays data from RFCOMM to an external RS-232 interface linked to a DCE. The communications parameters of the RS-232 interface are set according to received RPN commands, see Section 5.5.1.

7.2 SERVICE REGISTRATION AND DISCOVERY

Registration of individual applications or services, along with the information needed to reach those (i.e. the RFCOMM Server Channel) is the responsibility of each application respectively (or possibly a Bluetooth configuration application acting on behalf of legacy applications not directly aware of Bluetooth). Below is a template/example for developing service records for a given service or profile using RFCOMM. It illustrates the inclusion of the ServiceClassList with a single service class, a ProtocolDescriptor List with two protocols(although there may be more protocols on top of RFCOMM). The example shows the use of one other universal attribute (ServiceName). For each service running on top of RFCOMM, appropriate SDP-defined universal attributes and/or service-specific attributes will apply. For additional information on Service Records, see the SDP Specification,. The attributes that a client application needs (at a minimum) to connect to a service on top of RFCOMM are the ServiceClassIDList and the ProtocolDescriptorList (corresponding to the shaded rows in the table below).



Item	Definition	Type/Size	Value	Attribute ID
ServiceClassIDList			Note1	0x0001
ServiceClass0	Note5	UUID/32-bit	Note1	
ProtocolDescriptorList				0x0004
Protocol0	L2CAP	UUID/32-bit	L2CAP /Note1	
Protocol1	RFCOMM	UUID/32-bit	RFCOMM /Note1	
ProtocolSpecificParameter0	Server Channel	Uint8	N = server channel #	
[other protocols]		UUID/32-bit	Note1	
[other protocol-specific parame-ters]	Note3	Note3	Note3	
ServiceName	Displayable text name	DataElement/ String	'Example service'	Note2
[other universal attributes as appropriate for this service]	Note4	Note4	Note4	Note4
[service-specific attributes]	Note3	Note3	Note3	Note3

7.3 LOWER LAYER DEPENDENCIES

7.3.1 Reliability

RFCOMM uses the services of L2CAP to establish L2CAP channels to RFCOMM entities on other devices. An L2CAP channel is used for the RFCOMM/TS 07.10 multiplexer session. On such a channel, the TS 07.10 frames listed in Section 4.2 are sent, with the adaptation defined in Section 5.1.

Some frame types (SABM and DISC) as well as UIH frames with multiplexer control commands sent on DLCI 0 always require a response from the remote entity, so they are acknowledged on the RFCOMM level (but not retransmitted in the absence of acknowledgment, see Section 5.3). Data frames do not require any response in the RFCOMM protocol, and are thus unacknowledged. Therefore, RFCOMM must require L2CAP to provide channels with maximum reliability, to ensure that all frames are delivered in order, and without duplicates.

Should an L2CAP channel fail to provide this, RFCOMM expects a link loss notification, which should be handled by RFCOMM as described in Section 5.2.3.

7.3.2 Low power modes

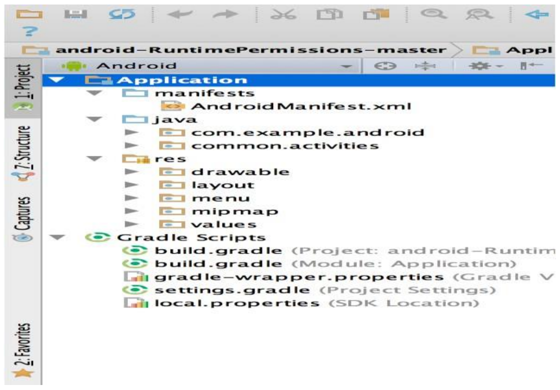
If all L2CAP channels towards a certain device are idle for a certain amount of time, a decision may be made to put that device in a low power mode (i.e. use hold, sniff or park, see ’Baseband Specification. This will be done without any interference from RFCOMM. RFCOMM can state its latency requirements to L2CAP.This information may be used by lower layers to decide which low power mode(s) to use. The RFCOMM protocol as such does not suffer from latency delays incurred by low power modes, and consequentially, this specification does not state any maximum latency requirement on RFCOMM’s behalf. Latency sensitivity inherently depends on application requirements, which suggests that an RFCOMM service interface implementation could include a way for applications to state latency requirements, to be aggregated and conveyed to L2CAP by the RFCOMM implementation. (That is if such procedures make sense for a particular platform.) .

8 ANDRIOD STUDIO

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

8.1 Project Structure



```
android-RuntimePermissions-master > Appl
Android
▼ Application
    ▼ manifests
        AndroidManifest.xml
    ▼ java
        ► com.example.android
        ► common.activities
    ▼ res
        ► drawable
        ► layout
        ► menu
        ► mipmap
        ► values
▼ Gradle Scripts
    build.gradle (Project: android-Runtim
    build.gradle (Module: Application)
    gradle-wrapper.properties (Gradle V
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

FIGURE 9. THE PROJECT FILES IN ANDROID VIEW.

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the AndroidManifest.xml file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in figure 1, it's showing as**Android**).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

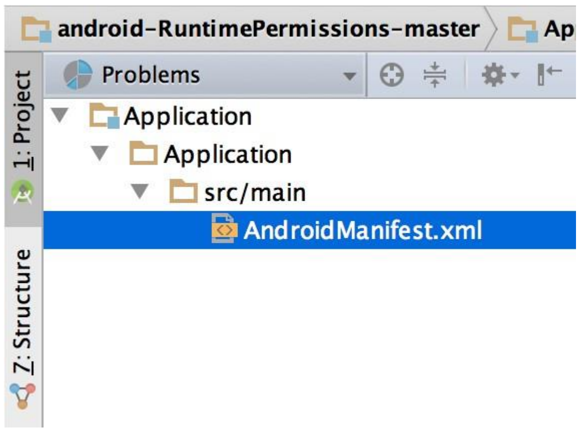
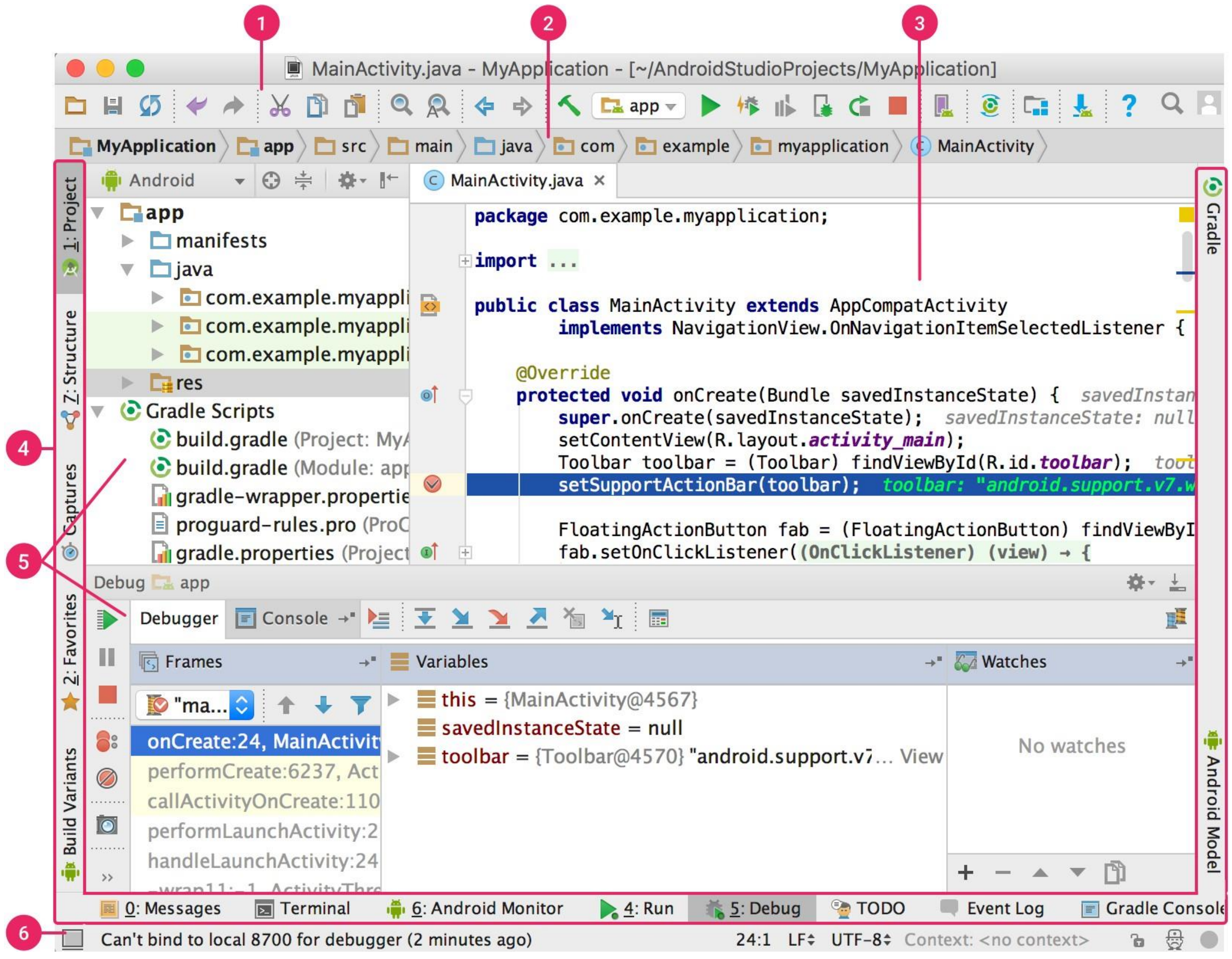


FIGURE 10 The project files in Problems view, showing a layout file with a problem.

8.1.1 User Interface

The Android Studio main window is made up of several logical areas identified in figure 3.



```
package com.example.myapplication;

import ...

public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {   savedInstan
        super.onCreate(savedInstanceState);   savedInstanceState: null
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);   tool
        setSupportActionBar(toolbar);   toolbar: "android.support.v7.w

        FloatingActionButton fab = (FloatingActionButton) findViewByI
        fab.setOnClickListener((OnClickListener) (view) → {
```

```
onCreate:24, MainActivit
performCreate:6237, Act
callActivityOnCreate:110
performLaunchActivity:2
handleLaunchActivity:24
```

```
this = {MainActivity@4567}
savedInstanceState = null
toolbar = {Toolbar@4570} "android.support.v7... View
```

```
Can't bind to local 8700 for debugger (2 minutes ago)        24:1  LF÷  UTF-8÷  Context: <no context>
```

1. he **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.

2. The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.

3. The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.

4. The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.

5. The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.

6. The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

You can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. You can also use keyboard shortcuts to access most IDE features.

At any time, you can search across your source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This can be very useful if, for example, you are trying to locate a particular IDE action that you have forgotten how to trigger.

8.2 Tool Windows

Instead of using preset perspectives, Android Studio follows your context and automatically brings up relevant tool windows as you work. By default, the most commonly used tool windows are pinned to the tool window bar at the edges of the application window.

- To expand or collapse a tool window, click the tool's name in the tool window bar. You can also drag, pin, unpin, attach, and detach tool windows.

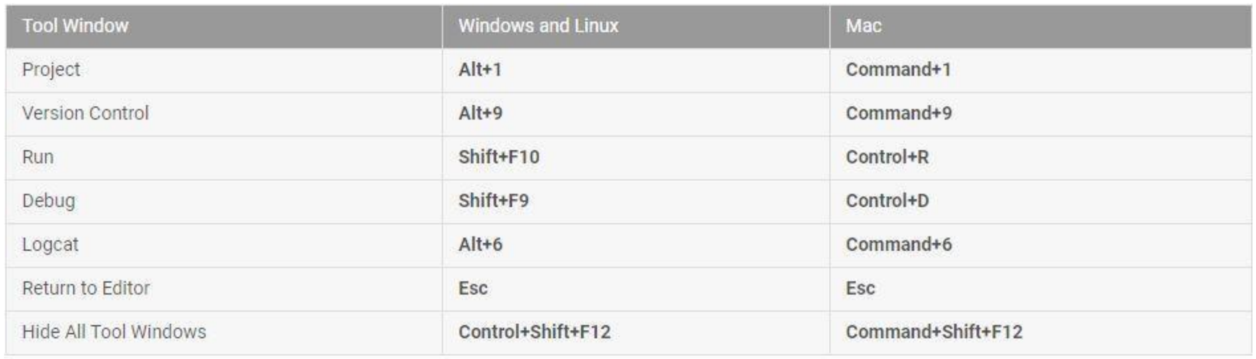
- To return to the current default tool window layout, click **Window > Restore Default Layout** or customize your default layout by clicking **Window > Store Current Layout as Default**.

- To show or hide the entire tool window bar, click the window icon ▢ in the bottom left-hand corner of the Android Studio window.

- To locate a specific tool window, hover over the window icon and select the tool window from the menu.

You can also use keyboard shortcuts to open tool windows. Table 1 lists the shortcuts for the most common windows.

Table 1. Keyboard shortcuts for some useful tool windows.



Tool Window	Windows and Linux	Mac
Project	**Alt+1**	**Command+1**
Version Control	**Alt+9**	**Command+9**
Run	**Shift+F10**	**Control+R**
Debug	**Shift+F9**	**Control+D**
Logcat	**Alt+6**	**Command+6**
Return to Editor	**Esc**	**Esc**
Hide All Tool Windows	**Control+Shift+F12**	**Command+Shift+F12**

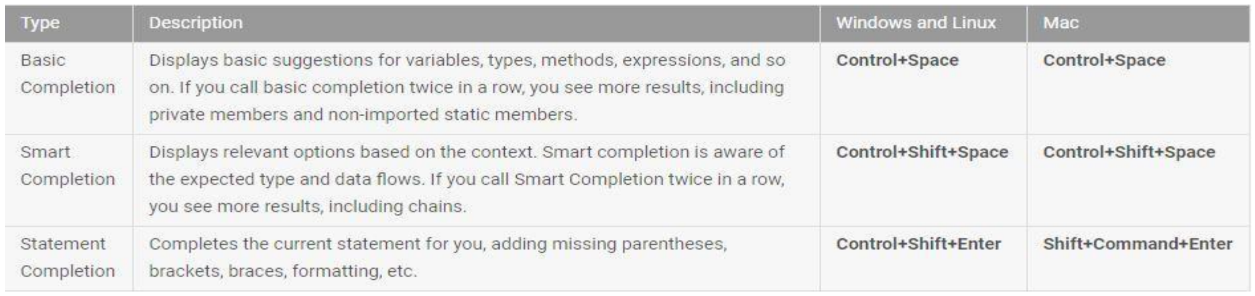
If you want to hide all toolbars, tool windows, and editor tabs, click **View > Enter Distraction Free Mode**. This enables *Distraction Free Mode*. To exit Distraction Free Mode, click **View > Exit Distraction Free Mode**.

You can use Speed Search to search and filter within most tool windows in Android Studio. To use Speed Search, select the tool window and then type your search query.

8.3 Code Completion

Android Studio has three types of code completion, which you can access using keyboard shortcuts.

Table 2. Keyboard shortcuts for code completion.



Type	Description	Windows and Linux	Mac
Basic Completion	Displays basic suggestions for variables, types, methods, expressions, and so on. If you call basic completion twice in a row, you see more results, including private members and non-imported static members.	**Control+Space**	**Control+Space**
Smart Completion	Displays relevant options based on the context. Smart completion is aware of the expected type and data flows. If you call Smart Completion twice in a row, you see more results, including chains.	**Control+Shift+Space**	**Control+Shift+Space**
Statement Completion	Completes the current statement for you, adding missing parentheses, brackets, braces, formatting, etc.	**Control+Shift+Enter**	**Shift+Command+Enter**

You can also perform quick fixes and show intention actions by pressing **Alt+Enter**.

8.4 Find sample code

The Code Sample Browser in Android Studio helps you find high-quality, Google-provided Android code samples based on the currently highlighted symbol in your project. For more information

8.5 Navigation

Here are some tips to help you move around Android Studio.

- Switch between your recently accessed files using the *Recent Files* action.
  Press **Control+E** (**Command+E** on a Mac) to bring up the Recent Files action. By default, the last accessed file is selected. You can also access any tool window through the left column in this action.

- View the structure of the current file using the *File Structure* action. Bring up the File Structure action by pressing **Control+F12** (**Command+F12** on a Mac). Using this action, you can quickly navigate to any part of your current file.

- Search for and navigate to a specific class in your project using the *Navigate to Class* action. Bring up the action by pressing **Control+N**(**Command+O** on a Mac). Navigate to Class supports sophisticated expressions, including camel humps, paths, line navigate to, middle name matching, and many more. If you call it twice in a row, it shows you the results out of the project classes.

- Navigate to a file or folder using the *Navigate to File* action. Bring up the Navigate to File action by pressing **Control+Shift+N** (**Command+Shift+O** on a Mac). To search for folders rather than files, add a / at the end of your expression.

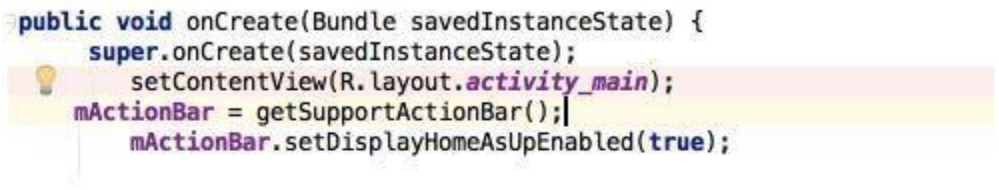
- Navigate to a method or field by name using the *Navigate to Symbol* action. Bring up the Navigate to Symbol action by pressing **Control+Shift+Alt+N**(**Command+Option+O** on a Mac).

- Find all the pieces of code referencing the class, method, field, parameter, or statement at the current cursor position by pressing **Alt+F7** (**Option+F7**on a Mac).

8.6 Style and Formatting

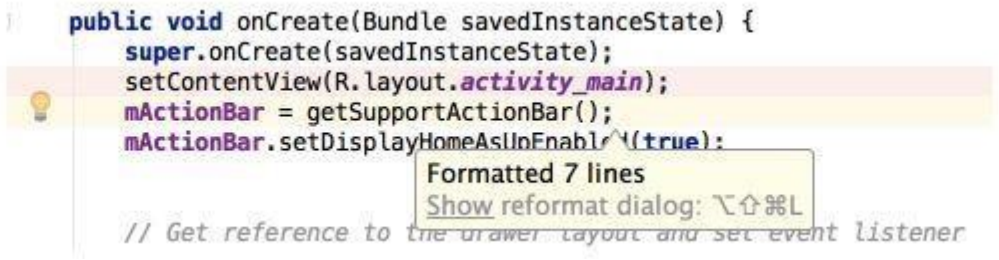
As you edit, Android Studio automatically applies formatting and styles as specified in your code style settings. You can customize the code style settings by programming language, including specifying conventions for tabs and indents, spaces, wrapping and braces, and blank lines. To customize your code style settings, click **File > Settings > Editor > Code Style** (**Android Studio > Preferences > Editor > Code Style** on a Mac.)

Although the IDE automatically applies formatting as you work, you can also explicitly call the *Reformat Code* action by pressing **Control+Alt+L**(**Opt+Command+L** on a Mac), or auto-indent all lines by pressing **Control+Alt+I** (**Control+Option+I** on a Mac).



```
public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
   mActionBar = getSupportActionBar();
         mActionBar.setDisplayHomeAsUpEnabled(true);
```

Code before formatting.



```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mActionBar = getSupportActionBar();
    mActionBar.setDisplayHomeAsUpEnabl(true);

    // Get reference to the drawer layout and set event listener
```

Code after formatting.

8.7 Version Control Basics

Android Studio supports a variety of version control systems (VCS's), including Git, GitHub, CVS, Mercurial, Subversion, and Google Cloud Source Repositories.

After importing your app into Android Studio, use the Android Studio VCS menu options to enable VCS support for the desired version control system, create a repository, import the new files into version control, and perform other version control operations:

1. From the Android Studio **VCS** menu, click **Enable Version Control Integration**.
2. From the drop-down menu, select a version control system to associate with the project root, and then click **OK**.

The VCS menu now displays a number of version control options based on the system you selected.

8.8 Gradle Build System

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android plugin for Gradle. This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process.
- Create multiple APKs for your app, with different features using the same project and modules.
- Reuse code and resources across sourcesets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files. Android Studio build files are namedbuild.gradle. They are plain text files that use Groovy syntax to configure the build with elements provided by the Android plugin for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

To learn more about the build system and how to configure, see Configure Your Build.

8.8.1 Build Variants

The build system can help you create different versions of the same application from a single project. This is useful when you have both a free version and a paid version of your app, or if you want to distribute multiple APKs for different device configurations on Google Play.

For more information about configuring build variants, see Configure Build Variants.

8.8.2 Multiple APK Support

Multiple APK support allows you to efficiently create multiple APKs based on screen density or ABI. For example, you can create separate APKs of an app for the hdpi and mdpi screen densities, while still considering them a single variant and allowing them to share test APK, javac, dx, and ProGuard settings.

For more information about multiple APK support, read Build Multiple APKs.

8.8.3 Resource Shrinking

Resource shrinking in Android Studio automatically removes unused resources from your packaged app and library dependencies. For example, if your application is using Google Play services to access Google Drive functionality, and you are not currently using Google Sign-In, then resource shrinking can remove the various drawable assets for theSignInButton buttons.

8.8.4 Managing Dependencies

Dependencies for your project are specified by name in the build.gradle file. Gradle takes care of finding your dependencies and making them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your build.gradle file. Android Studio configures projects to use the Maven Central Repository by default. (This configuration is included in the top-level build file for the project.) For more information about configuring dependencies, read Add Build Dependencies.

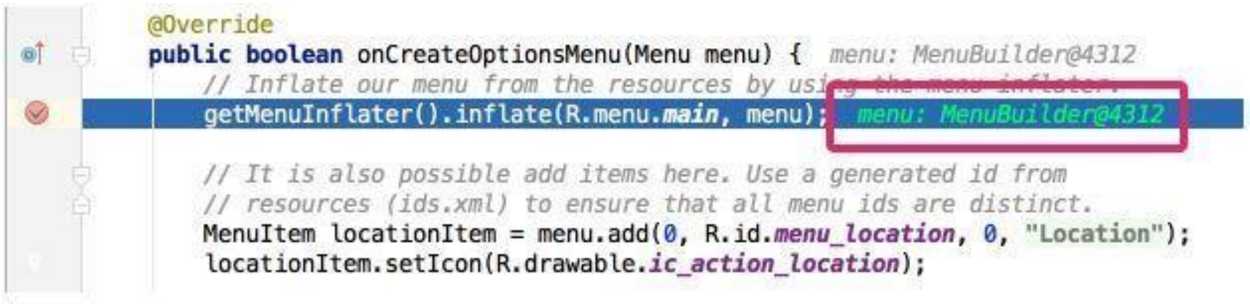
8.9 Debug and Profile Tools

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools.

8.9.1 Inline debugging

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

- Inline variable values
- Referring objects that reference a selected object
- Method return values
- Lambda and operator expressions
- Tooltip values



```
@Override
public boolean onCreateOptionsMenu(Menu menu) {  menu: MenuBuilder@4312
    // Inflate our menu from the resources by using the menu inflater.
    getMenuInflater().inflate(R.menu.main, menu);  menu: MenuBuilder@4312

    // It is also possible add items here. Use a generated id from
    // resources (ids.xml) to ensure that all menu ids are distinct.
    MenuItem locationItem = menu.add(0, R.id.menu_location, 0, "Location");
    locationItem.setIcon(R.drawable.ic_action_location);
```

An inline variable value.

To enable inline debugging, in the **Debug** window, click **Settings** and select the checkbox for **Show Values Inline**.

8.9.2 Performance profilers

Android Studio provides performance profilers so you can more easily track your app’s memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and analyze network requests. With your app running on a device or emulator, open the **Android Profiler**tab.

For more information about performance profilers, see Performance Profiling Tools.

8.9.3 Heap dump

When you’re profiling memory usage in Android Studio, you can simultaneously initiate garbage collection and dump the Java heap to a heap snapshot in an Android-specific HPROF binary format file. The HPROF viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks.

For more informatin about working with heap dumps, see Inspect the Heap and Allocations.

8.9.4 Memory Profiler

You can use Memory Profiler to track memory allocation and watch where objects are being allocated when you perform certain actions. Knowing these allocations enables you to optimize your app’s performance and memory use by adjusting the method calls related to those actions.

For information about tracking and analyzing allocations, see Inspect the Heap and Allocations.

8.9.5 Data file access

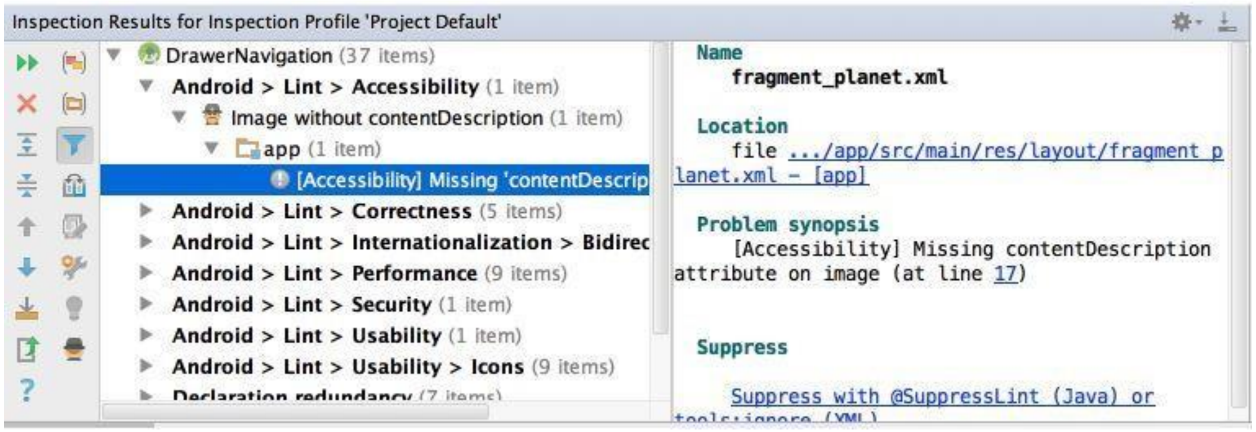
The Android SDK tools, such as Systrace, and logcat, generate performance and debugging data for detailed app analysis.

To view the available generated data files, open the Captures tool window. In the list of the generated files, double-click a file to view the data. Right-click any .hprof files to convert them to the standard Investigate Your RAM Usage file format.

8.9.6 Code inspections

Whenever you compile your program, Android Studio automatically runs configured Lint and other IDE inspections to help you easily identify and correct problems with the structural quality of your code.

The Lint tool checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.



Inspection Results for Inspection Profile 'Project Default'

```
▼ DrawerNavigation (37 items)
  ▼ Android > Lint > Accessibility (1 item)
    ▼ Image without contentDescription (1 item)
      ▼ app (1 item)
          [Accessibility] Missing 'contentDescrip
  ▶ Android > Lint > Correctness (5 items)
  ▶ Android > Lint > Internationalization > Bidirec
  ▶ Android > Lint > Performance (9 items)
  ▶ Android > Lint > Security (1 item)
  ▶ Android > Lint > Usability (1 item)
  ▶ Android > Lint > Usability > Icons (9 items)
  ▶ Declaration redundancy (7 items)
```

```
Name
    fragment_planet.xml

Location
    file .../app/src/main/res/layout/fragment_p
lanet.xml - [app]

Problem synopsis
    [Accessibility] Missing contentDescription
attribute on image (at line 17)

Suppress

    Suppress with @SuppressLint (Java) or
tools:ignore (XML)
```

The results of a Lint inspection in Android Studio.

In addition to Lint checks, Android Studio also performs IntelliJ code inspections and validates annotations to streamline your coding workflow.

For more information, see Improve Your Code with Lint.

8.9.7 Annotations in Android Studio

Android Studio supports annotations for variables, parameters, and return values to help you catch bugs, such as null pointer exceptions and resource type conflicts. The Android SDK Manager packages the SupportAnnotations library in the Android Support Repository for use with Android Studio. Android Studio validates the configured annotations during code inspection.

For more details about Android annotations, see Improve Code Inspection with Annotations.

9 IMPLEMENTATION

APPENDIX:

```
 //Constraints:
 package com.example.android.bluetoothchat;
 public interface Constants {
    public static final int MESSAGE_STATE_CHANGE = 1;
 public static final int MESSAGE_READ = 2;     public static
 final int MESSAGE_WRITE = 3;     public static final int
 MESSAGE_DEVICE_NAME = 4;     public static final int
 MESSAGE_TOAST = 5;     public static final String
 DEVICE_NAME = "device_name";     public static final
 String TOAST = "toast";


 }


//Main activity:
 package com.example.android.bluetoothchat;


 import android.os.Bundle; import
 android.support.v4.app.FragmentTransaction;
 import android.view.Menu; import
 android.view.MenuItem; import
 android.widget.ViewAnimator;
```

```
import
com.example.android.common.activities.SampleActivityBase;
import com.example.android.common.logger.Log; import
com.example.android.common.logger.LogFragment; import
com.example.android.common.logger.LogWrapper; import
com.example.android.common.logger.MessageOnlyLogFilter; public
class MainActivity extends SampleActivityBase {

    public static final String TAG = "MainActivity";
private boolean mLogShown;

    @Override     protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

        if (savedInstanceState == null) {
            FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();             BluetoothChatFragment fragment
= new BluetoothChatFragment();             transaction.replace(R.id.sample_content_fragment,
fragment);             transaction.commit();
        }
    }
```

```
   @Override     public boolean
onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }


   @Override     public boolean
onPrepareOptionsMenu(Menu menu) {
        MenuItem logToggle = menu.findItem(R.id.menu_toggle_log);
logToggle.setVisible(findViewById(R.id.sample_output) instanceof ViewAnimator);
logToggle.setTitle(mLogShown ? R.string.sample_hide_log : R.string.sample_show_log);


        return super.onPrepareOptionsMenu(menu);
    }

   @Override     public boolean
onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()) {
case R.id.menu_toggle_log:
mLogShown = !mLogShown;
                ViewAnimator output = (ViewAnimator) findViewById(R.id.sample_output);
                if (mLogShown) {
output.setDisplayedChild(1);
                } else {
                    output.setDisplayedChild(0);
                }
```

```
            supportInvalidateOptionsMenu();

            return true;

        }

        return super.onOptionsItemSelected(item);

    }

    @Override     public void

initializeLogging() {

        LogWrapper logWrapper = new LogWrapper();

        Log.setLogNode(logWrapper);

        MessageOnlyLogFilter msgFilter = new MessageOnlyLogFilter();

logWrapper.setNext(msgFilter);

        LogFragment logFragment = (LogFragment) getSupportFragmentManager()

                .findFragme

"Ready");

    } ntById(R.id.log_fragment);

        msgFilter.setNext(logFragment.getLogView());


        Log.i(TAG,

}


    }

}
```

10. TESTING

10.1 INTRODUCTION TO TESTING

Software testing is an activity to check whether the actual results match the expected results and to ensure that the software system is Defect free. It involves execution of a software component or system component to evaluate one or more properties of interest.

Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools. Some prefer saying Software testing as a white box and Black Box Testing.

BLACK BOX TESTING:

Crudely put, when the tester has no idea of the internal working of the system which he is testing, that approach is called black box testing.

In this case, the system under test is viewed as a “black box”.

Requirements Document or Functional Specification Document forms the basis of this testing, which requires the user to understand the processes within the software.

WHITE BOX TESTING:

In white box testing methodology, the tester has the knowledge of the internals of a system and knows how the system is implemented. The tester uses this knowledge to develop test cases that will examine the control flow, information flow, data flow, exception and error handling as well as coding practices of the system.

10.2 TYPES OF TESTING

We do three distinct kinds of testing on a typical software system: unit/ component testing, integration testing, and system testing. The objectives of each class is different and therefore, we can expect the mix of test methods used to differ. They are:

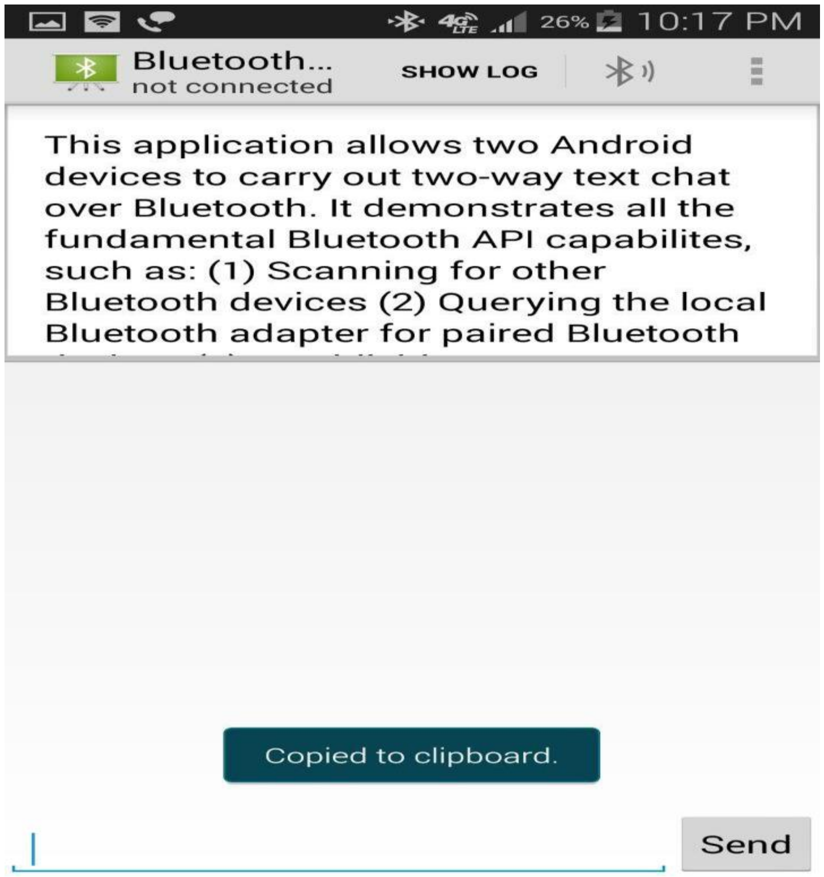
1.Unit Testing A unit is the smallest testable piece of software, by which I mean that it can be compiled or assembled, linked, loaded, and put under the control of a test harness or driver. A unit is usually the work of one programmer and it consists of several hundred or fewer, lines of source code. Unit testing is the testing we do to show that the unit does not satisfy its functional specification and/or that its implemented structure does not match the intended design structure. When our tests reveal such faults, we say that there is a unit bug.

2.Component Testing A component is an integrated aggregate of one or more units. A unit is a component; a component with subroutines it calls is a component, etc . By this (recursive)definition, a component can be anything from a unit to an entire system. Component testing is the testing we do to show that the component does not satisfy its functional specification and/or that its implemented structure does not match the intended design structure. When our tests reveal such problems, we say that there is a component bug.

3.Integration Testing Integration is a *process* by which components are aggregated to create larger components. Integration testing is testing done to show that even though the components were individually satisfactory, as demonstrated by successful passage of component tests, the combination of components are incorrect or inconsistent. For example, components A and B have both passed their component tests. Integration testing is aimed as showing inconsistencies between A and B. Examples of such inconsistencies are improper call or return sequences, inconsistent data validation criteria, and inconsistent handling of data objects. Integration testing should not be confused with testing integrated objects, which is just higher level component testing. Integration testing is specifically aimed at exposing the problems that arise from the combination of components. The sequence, then, consists of component testing for components A and B, integration testing for the combination of A and B, and finally, component testing for the "new" component (A,B).*

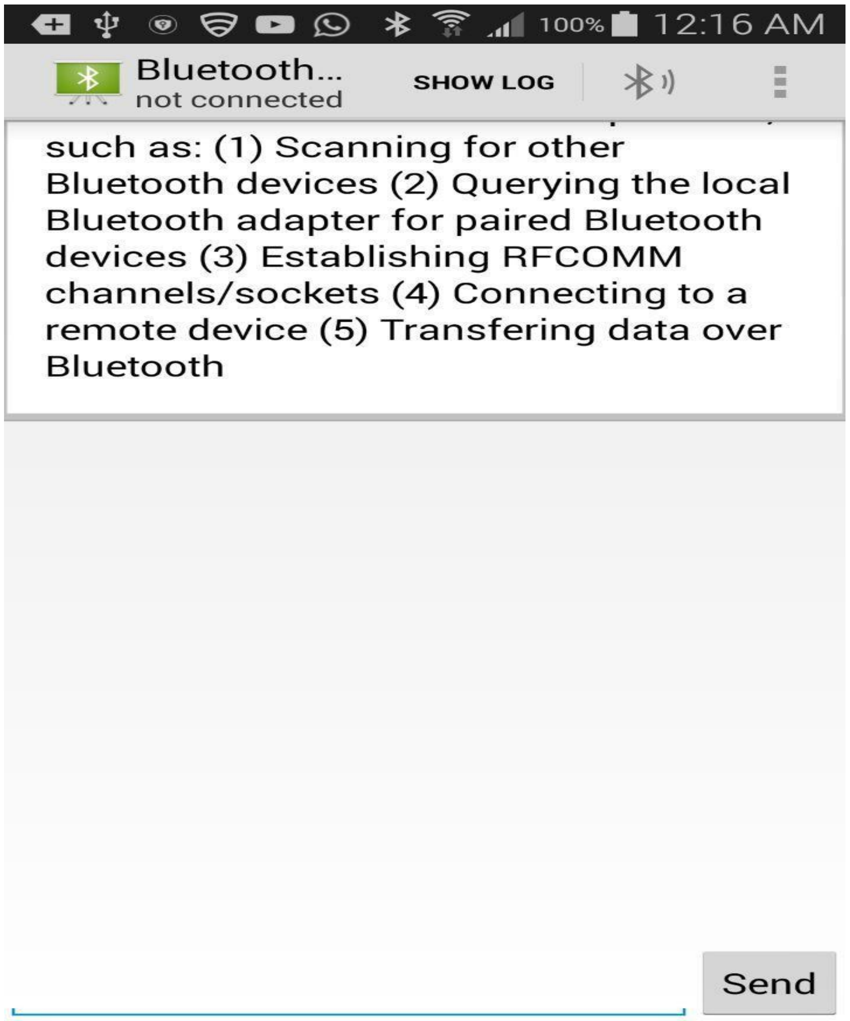
4.System Testing A system is a big component. System testing is aimed at revealing bugs that cannot be attributed to components as such, to the inconsistencies between components, or to the planned interactions of components and other objects. System testing concerns issues and behaviors that can only be exposed by testing the entire integrated system or a major part of it. System testing includes testing for performance, security, accountability, configuration sensitivity, start-up, and recovery.

10.3 TEST CASES



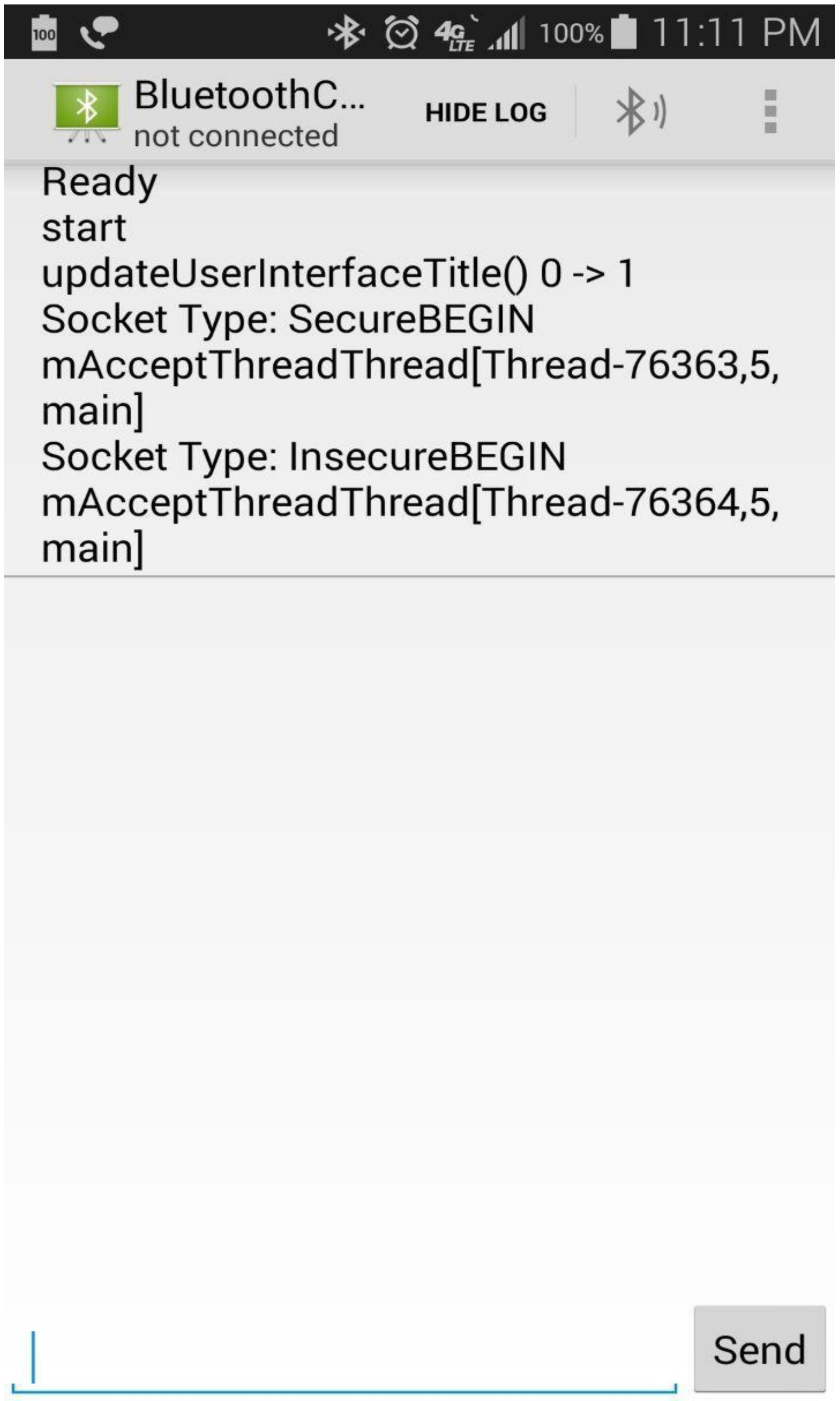
This application allows two Android devices to carry out two-way text chat over Bluetooth. It demonstrates all the fundamental Bluetooth API capabilites, such as: (1) Scanning for other Bluetooth devices (2) Querying the local Bluetooth adapter for paired Bluetooth

This interface will allow the connected users to communicate with each other. Here in the above image you can notice a text field provided where the user can send his/her text to the other person on the other side.



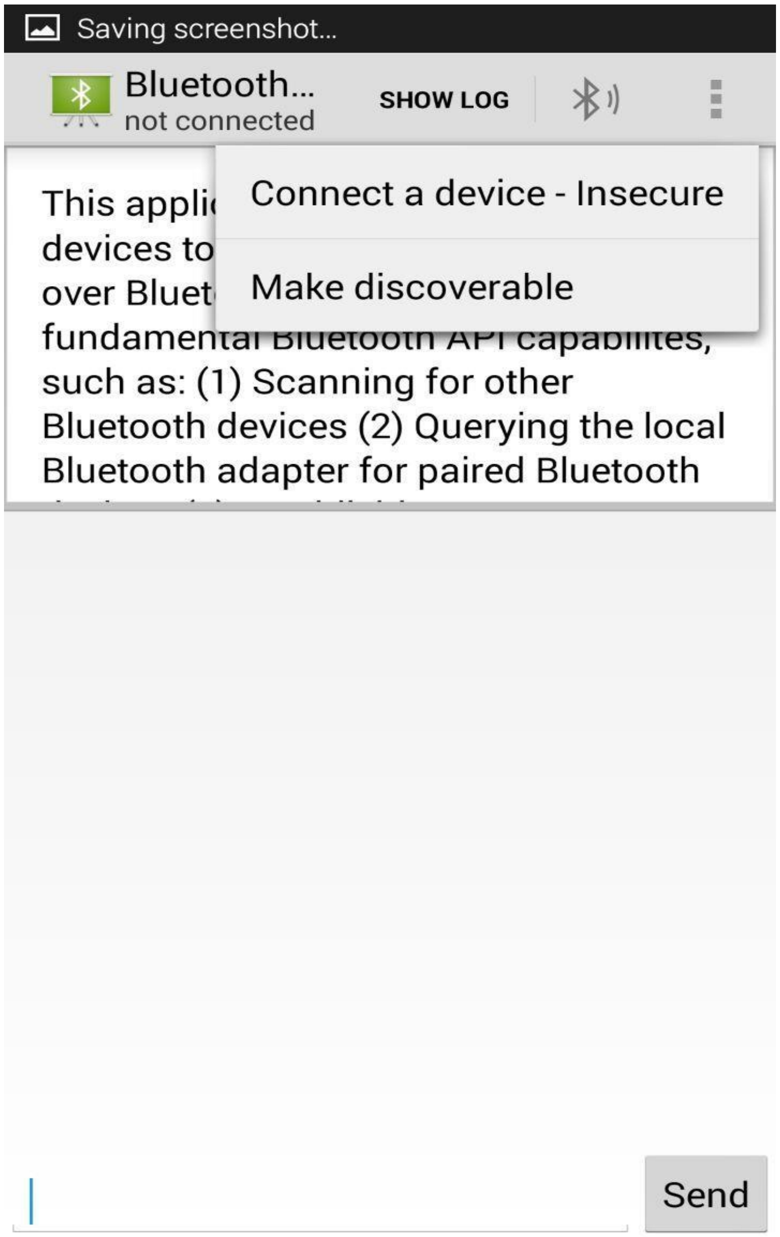
such as: (1) Scanning for other Bluetooth devices (2) Querying the local Bluetooth adapter for paired Bluetooth devices (3) Establishing RFCOMM channels/sockets (4) Connecting to a remote device (5) Transfering data over Bluetooth

The text which the person wants to send is entered in the text field and on clicking the send button which is visible on the right bottom of the screen will deliver the message to the other connected person.



```
Ready
start
updateUserInterfaceTitle() 0 -> 1
Socket Type: SecureBEGIN
mAcceptThreadThread[Thread-76363,5,
main]
Socket Type: InsecureBEGIN
mAcceptThreadThread[Thread-76364,5,
main]
```

We can see a option “show log” at the top on clicking that we get few statements as shown in the image this tells us that the application is ready and we can connect to anyone and communicate with them. This also shows the socket type and the thread.



This appli… devices to … over Bluet… fundamental Bluetooth API capabilites, such as: (1) Scanning for other Bluetooth devices (2) Querying the local Bluetooth adapter for paired Bluetooth

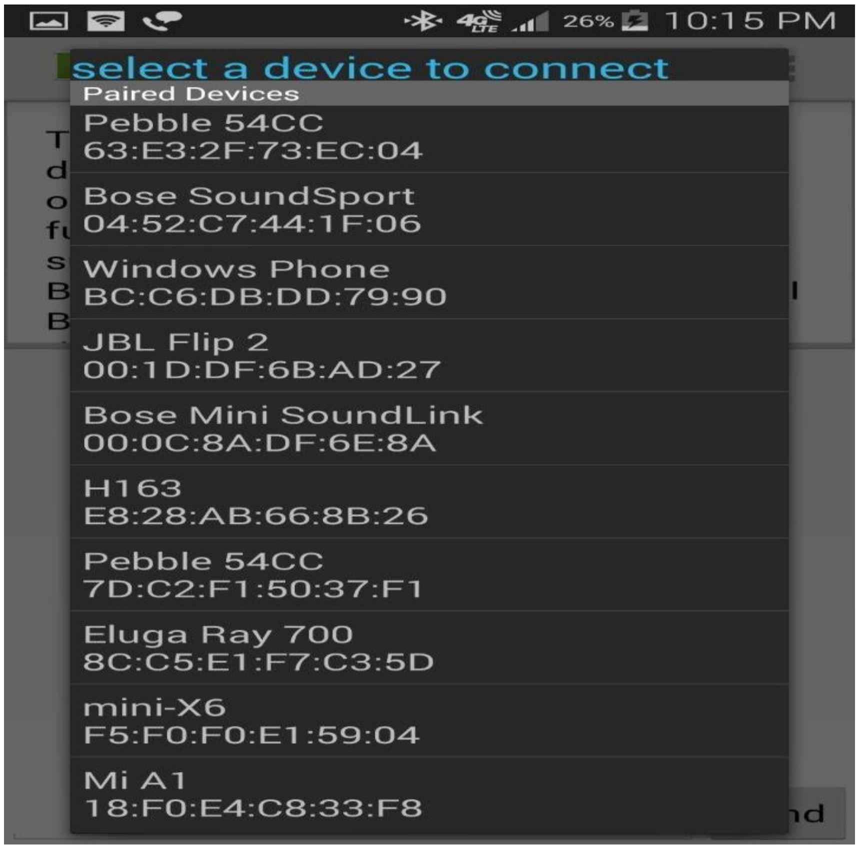
On clicking the option that is provided at the right top corner of the screen we get two options.

- Connect a device – insecure.
- Make discoverable.

"Connect a device-insecure": here the device will be connected to the device that is already connected to our device before.

Here the option "Make discoverable" tells us that our device is visible to all others who are in the Bluetooth range and can connect to them. On clicking that option we can make our device to be hidden to the people around us.

Inorder to communicate with the other person we must first pair our device with the persons device using Bluetooth. So to connect with the other person we must click on the Bluetooth symbol that is provided on the top. On clicking that Bluetooth symbol we get the information of all the devices that are been connected before. Choose the one you want to communicate with.
As shown in the below image .



10:15 PM

select a device to connect

Paired Devices

Pebble 54CC
63:E3:2F:73:EC:04

Bose SoundSport
04:52:C7:44:1F:06

Windows Phone
BC:C6:DB:DD:79:90

JBL Flip 2
00:1D:DF:6B:AD:27

Bose Mini SoundLink
00:0C:8A:DF:6E:8A

H163
E8:28:AB:66:8B:26

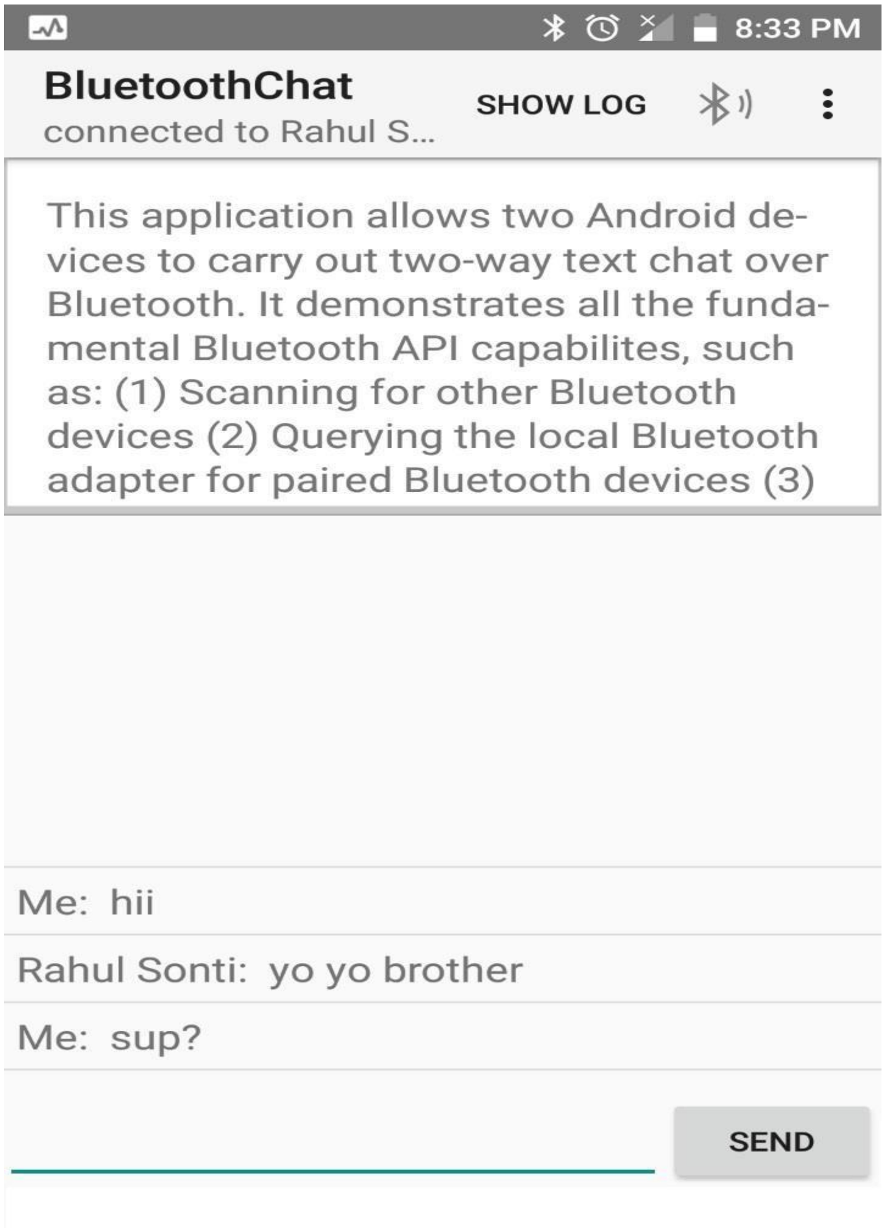
Pebble 54CC
7D:C2:F1:50:37:F1

Eluga Ray 700
8C:C5:E1:F7:C3:5D

mini-X6
F5:F0:F0:E1:59:04

Mi A1
18:F0:E4:C8:33:F8

11. OUTPUT



This application allows two Android devices to carry out two-way text chat over Bluetooth. It demonstrates all the fundamental Bluetooth API capabilites, such as: (1) Scanning for other Bluetooth devices (2) Querying the local Bluetooth adapter for paired Bluetooth devices (3)

Me: hii

Rahul Sonti: yo yo brother

Me: sup?

Here your chat box ready, happy texting.

12. CONCLUSION

Finally, Bluetooth chat application is an android application that allows its users to communicate without the need of any internet services. The users must only connect to the other with the Bluetooth service and can communicate with him/her. So, this application can be enhanced and implemented in various platforms like the IOS, Windows, Ubuntu, etc...

This would allow the user to communicate on a large scale in the future. This is still the prototype where you enable a connection with the paired devices and start conversating with them. This would bring us to a conclusion that it'd be implemented on various new networks other than the Bluetooth in the future. This application deals with the peer to peer connection via the Bluetooth network and the connection is enabled within the range of Bluetooth.

13. BIBLIOGRAPHY

1. *https://www.google.co.in/url?sa=t&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiR3JR8qzaAhXGN48KHVb6DxcQFghXMAA&url=https%3A%2F%2Fwww.bluetooth.com%2F&usg=AOvVaw1rDjuGa2dzD2LO8NPxa1eT – Bluetooth*
2. *https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=0ahUKEwiR3J-R8qzaAhXGN48KHVb6DxcQFghlMAE&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FBluetooth&usg=AOvVaw0fX3pZJ27Znt4X8Bc4MGYj*
3. *https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&cad=rja&uact=8&ved=0ahUKEwiR3J-R8qzaAhXGN48KHVb6DxcQFgh-MAM&url=https%3A%2F%2Fsearchmobilecomputing.techtarget.com%2Fdefinition%2FBluetooth&usg=AOvVaw3xlXp_cah0sNARTpspx8OU – The Bluetooth programming cook book*
4. *https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=12&cad=rja&uact=8&ved=0ahUKEwiQ__fy8qzaAhWHwI8KHQqgDhYQFghXMAs&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FList_of_Bluetooth_protocols&usg=AOvVaw24sV54FjUfpXf7hKVyAv9z – The protocols of RFCOMM*
5. *https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=13&cad=rja&uact=8&ved=0ahUKEwiQ__fy8qzaAhWHwI8KHQqgDhYQFghsMAw&url=https%3A%2F%2Fwww.amd.e-technik.uni-rostock.de%2Fma%2Fgol%2Flectures%2Fwirlec%2Fbluetooth_info%2Frfcomm.html&usg=AOvVaw046ai44juTWBrAGPuIHp09 – The layers of RFCOMM*