ACKNOWLEDGEMENT

I would like to express my sincere thanks to all the people with whose support the Completion of this project would be incomplete. The completion of the project would be incomplete without the mention of the people who made it possible, the people whose valuable guidance and encouragement paved the way for our efforts to be successful.

I consider it a privilege to express my gratitude and respect to all those who guided me in the completion of this project.

I express my sincere gratitude to Dr. DC Pande, Scientist ‘G ’ and Mr.E.Gopukumar Technical Analyst EMI / EMC, LRDE and Captain Mr. S. Kandasamy Consultant (EMU), DRDO, Bangalore for giving me the opportunity to do my project in LRDE.

I express my hearty thanks to Mr. Jithesh V, Scientist ‘C’, EMI / EMC, LRDE, for his inspiring and enlightening guidance through all the stages of development of this project. I am extremely grateful and indebted to him for having given me all the necessary inputs, advice and Knowledge, without which this project would not have been possible.

I am indebted to my internal guide, Mr. Muralidharan M, Head Of Department, Computer Science Department, Nehru Memorial College, Puthanampatti, Bharadhidasan University, for his invaluable guidance, support, encouragement, inspiration, and cooperation.

I also convey my regards to all my teachers for providing guidance, support, knowledge and encouragement.

I also express my regards and gratitude to my family and friends who have been an inspirational source of motivation and support in tough and testing times.
ABSTRACT

Blue-tooth is an industrial specification for wireless Personal Area Network (PAN). It provides a way to connect and exchange information between devices such as Laptop, PDA, mobile phones, PC’s, printers, digital cameras and video game consoles over a secure globally unlicensed short-range radio frequency.

A Blue tooth service is an application acting as a server that provides some kind of assistance to client devices via Blue tooth communications. After registering a service record in the Database, the server application waits for a client application to initiate contact with the server to access the service. This software tool namely “A Master - Slave Communication System Using Java Bluetooth” has the following capabilities:

- Create a service record describing the service offered by the application.
- Add a service record to the server’s Database to make potential clients aware of this service.
- Accept connections from clients that request the service offered by the application.
- Update the service record in the server’s Database if characteristics of the service change.
- Remove or disable the service record in the server’s DB when the service is no longer available.

Typical responsibilities of a Blue tooth client application are to:
- Use SDP to query a remote DB for desired services.
- Initiate connections to servers offering desired services.
- Optionally, poll the DB to determine if the service has changed or has become unavailable.

The Blue tooth stack is assumed to provide the following capabilities for local Blue tooth server applications:
- A repository for service records that allows servers to add, update and remove their own service records.
- Establishing logical connections to client applications.

The Blue tooth stack is assumed to provide the following capabilities for remote service discovery clients:
- Search and retrieval of service records stored in the server’s DB (that is, acting as an SDP server).
- Establishing logical connections to server applications.
CONTENTS

1. INTRODUCTION
1.1 Organization Profile ................................................................. 1
1.2 Scopes and Objective of the Project ............................................ 4

2. Bluetooth
2.1 Bluetooth Wireless Technology: .................................................... 4
2.2 Bluetooth Specification ............................................................... 5
2.3 Bluetooth Protocols ................................................................. 5
2.4 Bluetooth Devices .................................................................. 6

3. Java Wireless Toolkit
3.1 Multiple User Environment ......................................................... 8
3.2 Toolkit Components ................................................................. 8
3.3 Toolkit Features ................................................................... 9

4. Software Requirement Specification
4.1 Specific Requirements ............................................................... 10
4.1.1 Functional Requirements ..................................................... 10
4.2. Use cases ....................................................................... 10
4.3. Use case Descriptions .......................................................... 13
5. Design

5.1 Introduction: .................................................................17
5.2 High Level Design ..........................................................17
5.2.1 High-level architectural diagram ....................................17
5.3 Low Level Design ............................................................19
5.4 Class Diagrams ...............................................................27
5.5 Sequence Diagram ..........................................................30

6. Implementation, Deployment and Testing

6.1 Implementation ...............................................................34
6.1.1 Java Coding Standards ................................................35
6.1.2 J2me Emulator ............................................................40
6.1.2.1 Emulator Controls....................................................40
6.1.3 Application Development Under Java Wireless Toolkit ......41
6.2 Deployment .................................................................45
6.3 Testing and Results .........................................................46

7. Conclusion & Future Enhancements ......................................55

8. Bibliography .................................................................57

Appendix .................................................................58
1. INTRODUCTION

1.1. Organization profile: LRDE

Electronics and radar development establishment (LRDE) is one of the R & D Establishment set up under the Defence R & D Organization to address the services needs in the field of Radar, Communication Systems and related Technologies. LRDE has its genesis in the Inspectorate of scientific stores created in 1939 at Rawalpindi which was Redesignated as technical development Establishment (Instruments and Electronics) in 1946 and located at Dehradun. In the year 1958 with bifurcation of electronics activity into inspection and R & D. The electronics research Establishment was formed at Bangalore with men and material inherited from TDE (I & E). The Establishment was renamed in 1962 as Electronics and Radar Development Establishment (LRDE) and dedicated to the design and development of radar and Communication equipment.

A number of DRDO laboratories have originated from this mother Establishment. In 1986, LRDE moved from High Grounds, Bangalore to its present location spread over 60 acres at CV Raman Nagar, Bangalore amidst calm and serene surroundings. Today, LRDE has the core Competence and Expertise to build world class Radar Systems and Software Driven Electronics Systems.

Vision of LRDE:

To create a center of excellence in Radar and Electronics Technologies by developing and delivering World class systems to meet the customer requirements

Mission of LRDE

- To design and develop Radar systems meeting the current and futuristic requirements Of services, Keeping in view the emerging threat and EW scenario

- To contribute towards the establishment of indigenous of our designs through public and private sector, work committedly towards introduction of the systems into services.
To promote research and competence building activities in the field of Radar within the laboratory and in academic institutions and continuously evolve as the center of excellence in Radar Technology.

**EMI / EMC Evaluation Facility:**

EMI / EMC laboratory is one of such department in LRDE, which mainly concentrates on the Electromagnetic Interference (EMI) and Electromagnetic Compatibility (EMC) aspects of various communication systems. A full – fledged EMI / EMC test facilities, as per MIL STD 461C, is established to evaluate the equipment and systems. Facilities exist for conducted and radiated emission / susceptibility tests, electrostatic discharge tests, shielding effectiveness and surface transfer impedance measurements for in-house and other agencies programs. The department comprises of a number of experience personnel involved in the development of softwares requires for the EMI / EMC evaluation process and has its own computer laboratory equipped with modern hardware and experienced professionals.

**Core Competencies:**

LRDE has 3 decades of Radar System Engineering and Radar System development expertise. The radar technologies have undergone tremendous advancement over the years, especially in the last decade. LRDE has been continuously improving its expertise and knowledge base of Radar system Engineering starting from INDRA class of radars to the present day state -of -the-art radars-RAJENDRA, BFSR, 3D CAR, ALH Radar and WLR. LRDE is aware that ‘change is constant’ nothing else, therefore, LRDE has adapted Collaborative approach to improve the expertise and realize advanced radar systems and Sub –systems. Technologies require adequate infrastructure for their effective utilization. LRDE has built adequate and comprehensive infrastructure to fabricate and evaluate quality radar system and sub – systems, and the infrastructure is being improved continuously. LRDE has expertise and knowledge base in infrastructure design and development, operating processes, and software application development.
The core competencies are:

Radar Systems Engineering

- Realization of radars from concept to final field evaluation and weapon system integration
- Resource base in the area of Radar systems

Radar Technologies

- Radar Antennas: L to X-Band, including Phased Arrays, Slotted Waveguide Arrays and patch arrays, Beam Steering, Array Collimation for Passive and Active Phased Arrays
- Radar Receivers, Radar Sources for high spectral purity and Radar Transmitters (High Average – TWT, CFA and Klystron based)
- Radar Signal Processors – DSPs / ASICs based
- Radar Data Processors including Multi – sensors Data Fusion
- Computer Control of multi – function array Radars
- L-Band T / R modules

Radar system & sub-systems Testing and Evaluation Capabilities

- Antenna measurement facilities
- Quality, Reliability Assurance Testing
- Radar Test facilities at kolar site
- Radar test platform
1.2. **Scope and Objective of the project:**

- Development of a Client – Server (Master – Slave) application.
- The Client – Server (Master-Slave) Communication should be through blue tooth.
- Server should have a database containing Employee-ID and EmployeeName.
- It should be possible for the Client to,
  - Only authorized users should be allowed to perform Database operations
    - Add data to the server database
    - Delete data from the server database
    - Modify data from the server database
    - Send queries to the server to get employee related information
  - A Login application should be developed for creating a user database
2. BLUETOOTH

Bluetooth is a wireless technology for communication over distances of up to 10m. Offering reasonably fast data transfer rates of around 1 Mb/s, principally between Battery-powered devices. Bluetooth’s primary intent is to support the creation of ad hoc Personal area networks (PANs) for small data transfers (or voice communication) Between devices such as Laptop, PDA, mobile phones, PC’s, printers, digital cameras and video game consoles over a secure globally unlicensed short-range radio frequency.

2.1. Bluetooth Wireless Technology:

Bluetooth wireless technology is an open specification for a low-cost, low-power, short-range radio technology for ad hoc wireless communication of voice and data anywhere in the world.

- An open specification means that the specification is publicly available and royalty free.
- Bluetooth wireless technology works anywhere in the world because it operates at 2.4 GHz in the globally available, license free, industrial, scientific, and medical (ISM) band.

2.2. Bluetooth Specification

- Bluetooth wireless technology was originally developed as a cable replacement technology for connecting devices.
- The specification defines the over-the-air behavior to ensure compatibility of Bluetooth devices from different vendors.
- It defines the complete system from the radio up to the application level, including the software stack.

2.3. Bluetooth Protocols

The Bluetooth protocol can be broken up into two main items: layers and profiles. All the layers of the Bluetooth protocol form the protocol stack. The following layers of the Bluetooth protocol "stack up":

1. Host Controller Interface (HCI)
The Host Controller Interface is a layer of software that passes all your data from your computer to your attached Bluetooth device.

2. **Logical Link Control and Adaptation Protocol (L2CAP)**

The Logical Link Control and Adaptation Protocol is the core layer of the stack through which all data must pass. L2CAP boasts some powerful features like packet segmentation and reassembling of data, as well as protocol multiplexing. If you are trying to pass a very large packet of data, L2CAP breaks up the packet and sends smaller ones.

3. **Service Discovery Protocol (SDP)**

A Bluetooth device uses Service Discovery Protocol in order to discover services.

4. **RFCOMM**

RFCOMM is commonly known as the wireless serial port, or the cable replacement protocol. The name is derived from the fact that your serial ports are called COMM1, COMM2, etc. RFCOMM simulates the functionality of a standard serial port.

5. **Telephony Control Protocol Specification (TCS, TCS Binary, TCSBIN)**

Telephony Control Protocol Specification (TCS, TCS Binary, TCS-BIN) is used to send control signals to devices that want to employ the audio capabilities within Bluetooth.


In Bluetooth, this is an adopted protocol, so the Bluetooth SIG has incorporated the existing WAP protocol into the Bluetooth protocol to fit Bluetooth's needs. WAP requires that PPP, IP, and UDP be present in the stack.

7. **Object Exchange (OBEX)**

OBEX is a communication protocol initially defined by the Infrared Data Association (IrDA). OBEX is pretty useful when you want to transfer objects like files between Bluetooth devices. OBEX does not require that TCP and IP be present in the stack, but the manufacturer is free to implement OBEX over TCP/IP.

8. **Bluetooth Network Encapsulation Protocol (BNEP)**

The Blue tooth Network Encapsulation Protocol is a layer in the Blue tooth stack that allows other networking protocols to be transmitted over Blue tooth, namely Ethernet. BNEP is a popular choice because it encapsulates TCP/IP packets in L2CAP packets before handing off the data to the L2CAP layer in the stack.
9. Human Interface Device Protocol (HID)

The Human Interface Device Protocol is another adopted protocol in the Blue tooth specification. It was originally defined in the USB specification, and it lists the rules and guidelines for transmitting information to and from human interface devices like keyboards, mice, remote controls, and video game controllers.

2.4. Blue tooth Devices

Two or more Blue tooth-enabled devices come within range and establish a connection a personal area network is formed. A personal area network can either be a Pico net or a scatter net.

Personal Area Networks: Pico nets and Scatter nets

![Diagram showing Pico net and Scatter net connections]

Figure: 1 In Pico net the slaves can only communicate to the master
Figure: 2 A scatter net is formed when a slave in one Pico net is the master in another Pico net.
3. JAVA WIRELESS TOOLKIT

Java Wireless Toolkit is a set of tools that makes it possible to create applications for mobile phones and other wireless devices. Although it is based on the Mobile Information Device Profile (MIDP) 2.1, the Java Wireless Toolkit for CLDC also supports many optional packages, making it a widely capable development toolkit.

3.1. Multiple User Environment

- Java Wireless Toolkit 2.5.2 for CLDC can be installed on a system running a supported version of either Windows or Linux.
- All users with an account on the host machine can access the toolkit, either singly or simultaneously

FIGURE - 3  Project Listing Shows Local Projects in Bold Face

To run the demonstrations, start the toolkit as follows.

- On Microsoft Windows choose Start > Programs > Sun Java Wireless Toolkit 2.5.2 for CLDC > Wireless Toolkit 2.5.2.
- On Linux, change directory to toolkit/bin. To run the toolkit, enter ./ktoolbar
- Once the application is open, press the Run button. The emulator appears running the example application
3.2 Toolkit Components

Java Wireless Toolkit for CLDC has three main components:

- The user interface automates many of the tasks involved in creating MIDP applications.
- The emulator is a simulated mobile phone. It is useful for testing MIDP applications.
- A collection of utilities provides other useful functionality, including a text messaging console and cryptographic utilities.

From the user interface you can build applications, launch the emulator, and start the utilities. Alternatively, the emulator and utilities can be run by themselves, which is useful in many situations. If you want to demonstrate MIDP applications, for example, it's useful to run the emulator by itself. The only additional tool you need is a text editor for editing source code.

3.3 Toolkit Features

Java Wireless Toolkit for CLDC supports the creation of MIDP applications with the following main features:

- **Building and packaging**: You write the source code and the toolkit takes care of the rest. With the push of a button, the toolkit compiles the source code, preverifies the class files, and packages a MIDlet suite.

- **Running and monitoring**: You can run a MIDlet suite directly in the emulator or install it using a process that resembles application installation on a real device. A memory monitor, network monitor, and method profiler are provided to analyze the operation of your MIDlets.

- **MIDlet suite signing**: The toolkit contains tools for cryptographically signing MIDlet suites. This is useful for testing the operation of MIDlets in different protection domains.
4. SOFTWARE REQUIREMENT SPECIFICATION

4.1. Specific Requirements:

1. Functional Requirements:

   It describes the functional requirements by giving various use cases.

   **Actor:**

   ![Actor Diagram]

   This actor interacts with the system to perform various tasks like establishing a Client – Server device Connection through bluetooth, login for Client, add data to database, delete data from the database etc…
4.2 Use cases:

Use case # 1: [Client Login]

This use case for connecting Client to server through blue tooth and user can login to the client side and login value validated from logindatastore.

Use case # 2: [Add data to database]
This use case for user enter employee details in the client side and data are added in server side employee database through blue tooth connection

Use case # 3: [Delete data from database]

This use case for user enter employee details in the client side and data are deleted from server side employee database through blue tooth connection

Use case # 4: [Modify data from database]
This use case for user enter employee details in the client side and data are modified from server side employee database through blue tooth connection

Use case # 5: [Search data from database]

This use case for user enter employee details in the client side and data are searched from server side employee database through blue tooth connection

Use Case # 6: [AddUser]

This use case for user adding UserId and UserName in the LoginDataStoreManger class.

Dept. of Computer Science, Nehru Memorial College, Bharathidasan University
4.3. USE CASE DESCRIPTIONS

Use case # 1: [Client Login]

Actor: User

Pre condition: Server is running

Post condition: The user has successfully logged in (or) failed.

Main flow:
- On initiation of login for client by the user, the user enter the username and password
- The user select and click the “Login” button in the menu
- The user click the “OK” button
- Client sends the login data to the server and server validates the user

Alternate flow:
- The user name / password is invalid,
  Server will not allow the user to do any employee database operation

Exception flow:
- If blue tooth device is not found at Client / Server machine, no communication can be established

Use case # 2: [Add data to database]

Actor: user

Pre condition: The user has successfully logged in

Post condition: The user added data to database

Main flow:
- User selects and clicks Employee window in the menu
- Employee manager window is displayed
- User select and click add button in the menu
- User enter the employee Id, employee Name in the Add Employee window
- User click the “ok” button
- The data sent by the Client will get entered into the server database
Alternate flow:

- If the same Employee Name available in the server database, then the following employee cannot be added to the database

Exception flow:

- If blue tooth device is not found at Client / Server machine, no communication can be established

Use case # 3: [Delete data from database]

Actor: user

Pre condition: The user has successfully logged in

Post condition: The user delete data from database

Main flow:

- User selects and clicks Employee window in the menu
- Employee manager window is displayed
- User select and click delete button in the menu
- User enter the employee Id, employee Name in the Delete Employee window
- User click the “ok” button
- The data sent by the Client will get deleted from the server database

Alternate flow:

- If the no one Employee Name available in the server database, then the following employee cannot be deleted data from the database

Exception flow:

- If blue tooth device is not found at Client / Server machine, no communication can be established

Use case # 4: [Modify data from database]

Actor: user

Pre condition: The user has successfully logged in

Post condition: The user modified data from database

Main flow:

- User selects and clicks Employee window in the menu
Use case # 5: [Search data from database]

**Actor:** user

**Pre condition:** The user has successfully logged in

**Post condition:** The user searched data from database

**Main flow:**
- User selects and clicks Employee window in the menu
- Employee manager window is displayed
- User select and click search button in the menu
- User enter the employee Id, employee Name in the search Employee window
- User click the “ok” button
- The data sent by the Client will get searched from the server database

**Alternate flow:**
- Nil.

**Exception flow:**
- If blue tooth device is not found at Client / Server machine, no communication can be established
Use Case # 6: [AddUser]

Actor: user

Pre condition: Server is running.

Post condition: The user added data to database

Main flow:

- User Clicks the Login Window.
- User enter the UserName and Password in the Login window
- User click the “Add” button
- The data will be added in to the LoginDataStoreManager

Alternate flow:

- If the same Employee Name available in the server database, then the following employee cannot be added to the database

Exception flow:

- Nil.
5. DESIGN

5.1 Introduction:

Design is the meaningful engineering representation of something that is to be built. It can be traced to a customer’s requirements and at the same time assessed for quality against a set of predefined criteria for “good” design. Design Strategy is a high level statement about the approach to develop a system. In other words it can be described as a particular approach to develop a system. It includes statements on the system’s functionality, hardware and system software platform, and method for acquisition. Design requires experimentation. It is generally divided into two steps:

- A logical step that is independent from the implementation environment.
- A physical step that focuses on the ordering of resources and details pertaining to programming languages or to the execution environment.

The development of an application may be divided into several major areas. They are chained sequentially within a waterfall lifecycle, or they are distributed among the various iterations of an iterative lifecycle. In the project used Object Oriented Design Strategy as the approach for designing the software. Object Oriented Design (OOD) creates a representation of the real-world problem domain and maps it into a solution domain that is the software. OOD results in a Design that interconnects data objects and processing operations in a way that modularizes information and processing rather than processing alone.

5.2. HIGH LEVEL DESIGN:

5.2.1 High-level architectural diagram

In the high-level architectural diagram have two Subsystems:

1. UserManager for adding user to a logindatasore.
2. Client – Server application using bluetooth.

The system architecture is an below
The application has 3 modules.

- **User Manager**
  This module creates authorized users having a user name and a password.
  The classes under this module are:
  1. RecordMIDlet
  2. Preferences

- **Client Module**
  The Client Module provides an interface to the client. It enables a user to login to the system and allows authorized users to perform database operations. The following classes constitute this module:
  1. BluetoothMIDlet
  2. ClientMIDlet

- **Server Module**
  The server Module provides an interface to the server. Employee and login database are maintained in server side. The following classes constitute this module:
  1. BluetoothMIDlet
  2. ServerMIDlet
  3. LoginDataStoreManager
  4. EmployeeDataStoreManager
5.3. LOW LEVEL DESIGN

Classes

User manager application

1. RecordMIDlet

Variables:

- Private Preferences mPreferences
  This variable represents the login preferences
- Private Form mForm
  Variable for main form creation
- Private TextField mUserField
  Variable for UserName Field creation
- Private TextField mPassWordFiled
  Variable for Password Field creation

Methods:

- Public RecordMIDlet ()
  This function represents the username and password stored in a record
- Public void startApp () throws MIDletstatechangeException
  This function represent start a background thread when the MIDlet is started
- Public void pauseApp ()
  This method call when MIDlet is moved to paused state from active state
- Public void destroyApp (boolean unconditional)
  This method call when MIDlet is enter the destroyed state
- Public void commandAction (Command c, Displayable d)
  This function represents the code that react the action
2. Preferences

Variables:
- private String mRecordStoreName
  This Variable for used to stored the record.

Methods:
- Public class Preferences (String recordStoreName)
  This function represents the preferences for storing record name.
- Public void add (String username, String password) throws RecordStoreException
  This function represents the adding login details in the record storage
- Public void showRecords () throws RecordStoreException
  This function represents the shows all the record

Client-Server Communication

ClientModule

1. BluetoothMIDlet:

Methods:
- Public class BluetoothMIDlet extends MIDlet implements Runnable, CommandListener
  This function represents the BluetoothMIDlet creation
- Public void startApp () throws MIDletstatechangeException
  This function represent start a background thread when the MIDlet is started
- Public void pauseApp ()
  This method call when MIDlet is moved to paused state from active state
- Public void destroyApp (boolean unconditional)
  This method call when MIDlet is enter the destroyed state
- Public void run ()
  This function used to running the blue tooth MIDlet action

Dept. of Computer Science, Nehru Memorial College, Bharathidasan University
• Public void commandAction (Command c, Displayable d)
  This function represents the code that react the action

2. ClientMIDlet:
   Variables:
   ▪ Private Form mForm
     This variable for creating the form
   ▪ Private TextField mUserField
     This variable for creating the user field creation
   ▪ Private TextField mPasswordField
     This variable for creating the user name creation
   ▪ Private OutputStream out
     This variable for creating the output stream creation
   ▪ Private static final command CMD_EXIT
     This variable exit command creation
   ▪ Private static final command CMD_BACK
     This variable for back command creation
   ▪ Private static final command CMD_ADD
     This variable for creating the add button creation
   ▪ Private static final command CMD_DELETE
     This variable for creating the delete button creation
   ▪ Private static final command CMD_MODIFY
     This variable for creating the modify button creation
   ▪ Private static final command CMD_SEARCH
     This variable for creating the search button creation
   ▪ Private static final command CMD_OK
     This variable for creating ok command creation
   ▪ Private static final command CMD_NEXT
     This variable for creating the next command creation
   ▪ Private Display display
     This variable for creating display the window
   ▪ Private Form mainForm
     This variable for creating the main form creation
Private Form addForm
This variable for creating the add employee form creation

Private Form deleteForm
This variable for creating the delete employee form creation

Private Form modifyForm
This variable for creating the modify employee form creation

Private Form searchForm
This variable for creating the search employee form creation

Private boolean firstTime
This variable for creating the window display in true or false

Private TextField EmployeeIdField
This variable for creating the employee id field creation

Private TextField EmployeeNameField
This variable for creating the employee name field creation

Private boolean hasToBeClosed
This variable creating the window has been closed

Private boolean islogin
This variable for creating login function is true or false

Private boolean isAdd
This function for creating add employee function is true or false

Private boolean isDelete
This function for creating delete employee function is true or false

Private boolean isModify
This function for creating modify employee function is true or false

Private boolean isSearch
This function for creating search employee function is true or false

Methods:

• Public class ClientMIDlet extends BluetoothMIDlet implements CommandListener
  This function represents the ClientMIDlet creation

• Public ClientMIDlet ()
This function represents the Client Graphical user interface forms creation

- Public void run ()
  This function used to run the appropriate client action
- Public void pauseApp ()
  This method call when MIDlet is moved to paused state from active state
- Public void destroyApp (boolean unconditional)
  This method call when MIDlet is enter the destroyed state
- Private void ConnectToServer ()
  This function represents the establishing the client and server connection
- Public void commandAction (Command c, Displayable d)
  This function represents the code that react the Client action

ServerMIDlet

1. BluetoothMIDlet:

Methods:
- Public class BluetoothMIDlet extends MIDlet implements Runnable, CommandListener
  This function represents the BluetoothMIDlet creation
- Public void startApp () throws MIDletstatechangeException
  This function represent start a background thread when the MIDlet is started
- Public void pauseApp ()
  This method call when MIDlet is moved to paused state from active state
- Public void destroyApp (boolean unconditional)
  This method call when MIDlet is enter the destroyed state
- Public void run ()
  This function used to running the blue tooth MIDlet action
- Public void commandAction (Command c, Displayable d)
  This function represents the code that react the action
2. ServerMIDlet

Variables:
- Private loginDataStoreManager loginManager
  This variable represents the login data stored on the server side
- Private EmployeeDataStoreManager employeeManager
  This variable represents the employee data stored on the server side
- Private Form mainForm
  This variable is used for creating the main form
- Private Boolean isOver
  This variable is for creating the case value is true or false
- Private InputStream in
  This variable is for creating the input stream creation
- Private ByteArrayOutputStream out
  This variable is for creating the byte array stream creation
- Private StreamConnection conn
  This variable is for creating the stream connection
- Private StreamConnectionNotifier notifier
  This variable is for creating the stream connection notifier
- Private Boolean isValid
  This variable is for validating the user input

Methods:
- Public class serverMIDlet BluetoothMIDlet implements CommandListener
  This function represents serverMIDlet creation
- Public serverMIDlet ()
  This function represents employee and login data are maintained
- Public void run ()
  This function represents the running the appropriate server action
- Public void Connect ()
This function represents the connect to server through blue tooth

- Public void ServeClient()
  This function represents the server serve to client
- Public void pauseApp()
  This method call when MIDlet is moved to paused state from active state
- Public void destroyApp()
  This method call when MIDlet is enter the destroyed state
- Public void CommandAction(Command c, displayable s)
  This function represents the code that react the Client action
- Private boolean validateUser(String data)
  This function represents validating the user input
- Private boolean addEmployee(String data)
  This function represents the add employee data in the database
- Private boolean deleteEmployee(String data)
  This function represents the delete employee data from the database
- Private boolean modifyEmployee(String data)
  This function represents the modify employee data from the database
- Private boolean searchEmployee(String data)
  This function represents the search employee data from the database

3. LoginDataStoreManager

  Methods:
  - public LoginDataStoreManager()
    This function represents the record storage information related to username and password
  - public boolean validateUser(String test) throws RecordStoreException
    This function represents the validating the user function

4. EmployeeDataStoreManager:

  Variables:
private static String dataStore = "EmployeeStore"
This variable is used to access the EmployeeStore

Methods:

- Public EmployeeDataStoreManager ()
  This function represents the employee related database
- Public boolean addEmployee (String employee)
  This function represents adding the employee data
- Public boolean deleteEmployee (String employee)
  This function represents deleting the employee data
- Public boolean modifyEmployee (String employee)
  This function represents modifying the employee data
- Public boolean searchEmployee (String employee)
  This function represents searching the employee data
- Public void showRecords () throws RecordStoreException
  This function represents shows the all employee record
5.4. CLASS DIAGRAMS

1. ClientMIDlet
2. ServerMIDlet

![Diagram of a Master-Slave Communication System using Java Bluetooth]

- **ServerMIDlet**
  - `MPIA` (Runnable CommandListener)
  - `BluetoothMIDlet`
  - `LoginDataStoreManager`
    - `addEmployee(String): boolean`
    - `deleteEmployee(String): boolean`
    - `modifyEmployee(String): boolean`
    - `sendEmployee(String): boolean`
  - `EmployeeDataStoreManager`
    - `addEmployee(String): boolean`
    - `deleteEmployee(String): boolean`
    - `modifyEmployee(String): boolean`
    - `showEmployee(String): void`
  - `ServerMIDlet`
    - `validateUser(String): boolean`
3. RecordMIDlet
5.5 SEQUENCE DIAGRAMS:
The sequence diagram shows the sequence of events that occur during the execution of the application.

1. AddUser
2. Client Login

![Sequence Diagram]

Client connects to the Server.

Sends User Name & password to Server via Bluetooth.
3. AddEmployee
4. DeleteEmployee
6. IMPLEMENTATION, DEPLOYMENT AND TESTING

6.1. Implementation

Implementation depends on the components in the system, namely, the bluetooth hardware, the Bluetooth stack, operatingsystem, the KVM, the CLDC implementation, and possibly other optional Java APIs or profiles. Because these components vary from device to device, the issues with JABWT implementation vary. Typically only selection of the bluetooth protocol stack and CLDC/KVM implementation influence the porting of JABWT to a device. For the device manufacturer today, there are wide ranges of choices for each of these components. Therefore, instead of covering the minute details of porting for a particular set of components, this chapter highlights the main issues an implementer needs to think about while implementing JABWT on a device. It is recommended that anyone attempting to port JABWT to a device understand the Bluetooth specification, the Bluetooth protocol stack, the Bluetooth radio hardware being used, and the KVM and CLDC on the device.

Adding J2ME and Bluetooth Support

The JABWT implementation can be simplified by selection of a Bluetooth protocol stack and a KVM with appropriate features. Of all the modules, the KVM-stack interface is the most crucial, hence selecting the stack for the JABWT device is an important step.

Stack Features that JABWT Requires

JABWT implementation and the Bluetooth protocol stack is simplified if the stack has the following features:

• Applications access the stack through a set of APIs.
• The stack supports asynchronous calls for all operations that may require a nontrivial amount of time for completion.
• Applications are notified of asynchronous events. The use of callback functions is the most common way of accomplishing this task, but it can also be performed through interrupts or application-level polling.
The stack should provide the following security and support features:

- Determine whether the given device is trusted
- Authorize the given connection and remote device
- Set the device PIN
- Enable or disable encryption
- Authenticate the remote device
- Determine whether a given remote device has already been authenticated
- Determine whether a given connection has already been authorized
- Determine whether a given link is encrypted
- Get the user-friendly name of the local device
- Get the user-friendly name of the remote device
- Get the class of device information for the local device
- Change the discoverable mode of the local device
- Enable or disable the connectable mode for the local device
- Get the Bluetooth address of the local device

6.1.1 Java Coding standards

The coding for the application is done in JAVA. The Source code for the application is given in Appendix

File organization:

A file consists of sections that should be separated by blank lines and an optional comment identifying each section. Files longer than 2000 lines are cumbersome and should be avoided. Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.

**Java source files have the following ordering:**

Beginning comments
Package and Import statements
Class and interface declarations
All source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice. The first non-comment line of most Java source files is a package statement. After that, import statements can follow.

**Indentation**

Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

**Wrapping Lines**

When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

**Comments**

Java programs can have two kinds of comments: implementation comments and documentation comments. Implementation comments are those found in C++, which are delimited by /*...*/, and //. Documentation comments (known as "doc comments") are Java-only, and are delimited by /**...*/. Doc comments can be extracted to HTML files using the java doc tool. Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program.

**Implementation Comment Formats**

Programs can have four styles of implementation comments: block, single-line, trailing, and end-of-line.
1. Block Comments
Block comments are used to provide descriptions of files, methods, data structures and algorithms

/*
 * Here is a block comment.
 */

2. Single-Line Comments
Short comments can appear on a single line indented to the level of the code.

if (condition) {
    /* Handle the condition. */

    ...
}

3. Trailing Comments
Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements

if (a == 2) {
    return TRUE;            /* special case */
} else {
    return isPrime(a);      /* works only for odd a */
}

4. End-Of-Line Comments
The // comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments

if (foo > 1) {
    // Do a double-flip.

    ...
}
else {
    return false;          // Explain why here.
}

//if (bar > 1) {
```java
//
//    // Do a triple-flip.
//    ...
//}
//else {
//    return false;
// }
```

5. Documentation Comments:
Doc comments describe Java classes, interfaces, constructors, methods, and fields. Each doc comment is set inside the comment delimiters `/**...*/`, with one comment per class, interface, or member. This comment should appear just before the declaration:

```java
/**
 * The Example class provides ...
 */
public class Example {...
```

Declarations

Number Per Line
One declaration per line is recommended since it encourages commenting.

Initialization
Try to initialize local variables where they're declared. The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.

Placement
Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}"). Don't wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.

Class and Interface Declarations
When coding Java classes and interfaces, the following formatting rules should be followed:

- No space between a method name and the parenthesis "(" starting its parameter list
- Open brace "{" appears at the end of the same line as the declaration statement
- Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{"-

**White Space**

**Blank Lines**
Blank lines improve readability by setting off sections of code that are logically related. Two blank lines should always be used in the following circumstances:
- Between sections of a source file
- Between class and interface definitions

**Blank Space**
Blank spaces should be used in the following circumstances:
- A blank space should appear after commas in argument lists.
- All binary operators except `.` should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands

**Programming Practices**
It consider the following formation
- Providing Access to Instance and Class Variables
- Referring to Class Variables and Methods
- Constants
- Variable Assignments
- Miscellaneous Practices
- Parentheses
- Returning Values
- Expressions before `?` in the Conditional Operator
- Special Comments
6.1.2 J2ME Emulator

6.1.2.1 Emulator Controls
Java Wireless Toolkit for CLDC emulator simulates a MIDP device on your desktop computer. It is a convenient way to see how your application performs in a MIDP environment and gives you a tight development cycle that is entirely contained on your desktop computer. The emulator does not represent a specific device, but it provides correct implementations of its supported APIs. The emulator looks and acts like a mobile phone inside a standard desktop window. This section describes how to control the emulator. The description and figures are based on the DefaultColorPhone skin, but all the skins operate in a similar way.

6.1.2.2 Working Directory Files
At installation time the installer copies a subset of files from the installation directory to the working directory of each user. The working directory contents are as follows:

- **settings** - Initially contains security.properties.
- **appdb** - The entire applications database is copied to your working directory.
- **apps** - The apps directory is created empty in your working directory. If you open a demonstration project it is automatically copied to this location. When you create a new project, it is stored here.
- **wtklib** - Contains the emulator properties file and state information for the HTTP and WMA servers and clients.
- **sessions** - This is the default save destination for monitoring tools. It is initially empty. All the source code for the demonstration applications is available in toolkit\apps, and each demonstration has its own project directory. Any applications placed in toolkit\apps are visible to all users.
you can use the mouse to click the buttons to press them. Most buttons also have keyboard shortcuts, which are generally easier to use. Keyboard numbers 0 through 9 correspond to the emulator's 0 through 9 buttons.

Another convenience is the capability to copy and paste information in text areas. You can paste text from the clipboard into a TextBox or TextField by pressing Control-v. To copy the contents of a TextBox or TextField, press Control-c. The entire contents of the text field will be placed on the clipboard.

6.1.3 Application development under java wireless toolkit

The J2ME record management system

The J2ME record management system (RMS) provides a mechanism through which MIDlets can persistently store data and retrieve it later. In a record-oriented approach, J2ME RMS comprises multiple record stores. Each record store can be visualized as a collection of records, which will remain persistent across multiple invocations of the MIDlet. The device platform is responsible for making its best effort to
maintain the integrity of the MIDlet's record stores throughout the normal use of the platform, including reboots, battery changes, etc. Each record in a record store is an array of bytes and has a unique integer identifier.

Java APIs for Bluetooth Wireless Technology

The goal of the specification was to define an open, non-proprietary standard API that can be used by all J2ME-enabled devices. Therefore, it was designed using standard J2ME APIs and CLDC/MIDP's Generic Connection Framework. Some important features:

- The specification provides basic support for Blue tooth protocols and profiles. It doesn't include specific APIs for all Blue tooth profiles simply because the number of profiles is growing.
- The specification incorporates the OBEX, L2CAP, and RFCOMM communication protocols in the JSR 82 APIs, primarily because all current Blue tooth profiles are designed to use these communication protocols.
- The JSR 82 specification addresses the Generic Access Profile, Service Discovery Application Profile, Serial Port Profile, and Generic Object Exchange Profile.
- The Service Discovery protocol is also supported. JSR 82 defines service registration in detail in order to standardize the registration process for the application programmer.

JSR 82 is Flexible

JSR 82 requires that the Blue tooth stack underlying a JSR 82 implementation be qualified for the Generic Access Profile, the Service Discovery Application Profile, and the Serial Port Profile. The stack must also provide access to its Service Discovery Protocol, and to the RFCOMM and L2CAP layers. The APIs are designed in such a way that developers can use the Java programming language to build new Bluetooth profiles on top of this API as long as the core layer specification does not change. To promote this flexibility and extensibility, the specification is not restricted to APIs that implement Blue tooth profiles. JSR 82 includes APIs for OBEX and L2CAP so that future Blue tooth profiles can be implemented in Java, and these are already being used for that purpose.
Create a Project

Click New Project. The toolkit will ask you for the name of the project and the name of the MIDlet class you will write. Fill in the names and click Create Project.

Click the Open Project button to see a list of all the available applications. Pick one of them and click the Open Project button in the dialog. If you have not opened this Project before; a copy will be created in your working directory once the application is open, press the Run button. The emulator appears running the example application.
Architecture Diagrams: CLDC and Bluetooth architecture

CLDC and MIDP and Bluetooth architecture
6.2. Deployment

The BluetoothMIDlet suite JAR contains keys that should be secret. Therefore, it is a security risk to transmit the JAR to a customer over the Internet. You might transfer it via HTTPS to a customer’s browser, and then rely on that customer to install the MIDlet suite on a mobile telephone or other small device via a serial cable. A wireless application is essential to ensure that the transactions cannot be fraudulently generated, that transactions are legally binding, and that the confidentiality of private information is adequately protected. Such assurance depends upon deployment of public-key infrastructure (PKI) technology in wireless data networks. When wireless devices are used for data communication, they have to face many environmental limitations e.g. limited computing power in wireless devices, inadequate memory space, low network bandwidth and restriction imposed by underlying communication protocols and services. This obviously creates further challenges for protocol designers, implementers and standardization body. This paper discusses the issues involved in implementing PKI in wireless data networks and possible approaches to resolve them. It also provides an overview of today’s mobile system environment and current status of wireless security implementations.

Deployment Support Networks

The Deployment Support Network (DSN) is a tool for the development, debugging and monitoring of distributed wireless embedded systems in a realistic environment. The basic idea is to use a second wireless network consisting of so-called DSN-nodes that are directly attached to the target nodes. The DSN provides a separate reliable wireless backbone network for the transport of debug and control information from and to the target-nodes. However, it is not only a replacement for the cables in wired test beds but it also implements interactive debugging services such as remote reprogramming, RPC and data/event-logging.

Performance Evaluation

The performance of our implementation on the BTnodes is mostly limited by the packet processing soft- and hardware. In fact, the micro controller is too slow for the packet processing at full Bluetooth-speed. Incoming packets are stored in a receive
buffer. If the arrival rate of packets is higher than the processing rate for a certain time, packets are dropped due to the limited capacity of the receive buffer. This affects the performance of the DSN in several ways: (a) pushed log-messages might get lost, (b) pulled log-messages might get lost, (c) commands might get lost, and (d) data dissemination packets might get lost. The probability of these cases increases with the amount of traffic on the DSN backbone network. For many scenarios the user can control what and when data is sent on the DSN. He can e.g. wait for the completion of the data dissemination before he starts pulling messages. In general, cases (b)-(d) are not critical, as they can be resolved with retransmission. However, in a scenario, where all nodes periodically generate log messages that are pushed simultaneously to the server, the log messages cannot be retransmitted. So for case (a), the user wants to know the transport capacity of the DSN, such that he can adjust the parameters of the setup.

6.3. Testing and Results

Testing:

6.3.1 Role of Testing:

The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional external, that is conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results. The increasing cost of failure and critical nature of software has brought focus on software testing in development organization. Software testing has become the critical element in software quality assurance. The approach to development is to focus on sharply so that only a minimal number of defects reach the final stage of testing. The quality has to pass through planned test cycles to ensure quality.

6.3.2. Objective of Testing:

The goal of testing is to access and improve quality of work products generated during development and modification of software. High quality product cannot be achieved through testing of source code alone. The best way to minimize the number of errors in program is to catch and remove the errors during analysis and design so that few errors in program are introduced in the source code.

Extensive unit level white box tests were carried separately for each module for ensuring that the modules will work properly. Afterwards Server-Client integration tests
were carried out for the functioning of client – server Bluetooth communication and Datastore operations. All these tests were found satisfactory.

Results

CLIENT EMULATOR
SERVER EMULATOR
Client ready for Bluetooth connection
Server ready for Bluetooth connection
Login Window
Employee Window
AddEmployee
Add data:

![Sun Java(TM) Wireless Toolkit 2.5.2 for CLDC - Employee Server](image1)

The data to add is 1#1010|lithesh|1
Data Received at server = 1, 1010|lithesh, 1

Login Records store

![Sun Java(TM) Wireless Toolkit 2.5.2 for CLDC - DataStore Manager](image2)

Record Store before update...
Record = guest@emici
Record = admin@emici
Record Store after update...
Record = 1
Record = guest@emici
Record = admin@emici
Execution completed.
3485710 bytes used executed
7. CONCLUSION & FUTURE ENHANCEMENTS

The purpose of this project is to develop a software tool namely “A Master – Slave Communication System using Java Bluetooth”. This application is the first step towards establishing a Bluetooth enabled network in a Company Campus. The application makes use of Java’s Bluetooth API to establish a Client – Server communication without the troubles of wired communication.

Although only few database operations are implemented on a very small datastore, the application can be modified in the future to include more functionality. The datastore can be enhanced to include complete information related to employees. More database operations like searching the datastore based on different criterion can be included in future. The present version doesn’t support any feedback from the server to the client. In future this can be included.

The current implementation is for PC based communication. Enhancements can be done to handle communication between devices like Laptops, PDAs, mobile phones etc.
8. BIBLIOGRAPHY

Books/Reports:

- Muller, *Bluetooth Demystified*
- Bray and Sturman, *Bluetooth 1.1: Connect Without Cables*
- Tranter and Woerner, *Wireless Personal Communication: Bluetooth Tutorial and other Technologies*
- Sing Li and Jonathan Knudsen, *Beginning J2me: From Novice to Professional*, Third Edition
- Leszek A.Maciaszek, *Requirements Analysis and system Design, Developing information systems with UML.*
- C Bala Kumar, Paul J. Kline, Timothy J. Thompson, *Bluetooth Application Programming with java Api’s*
- Martin de Jode, *Programming Java 2 Micro Edition on Symbian OS: A developer’s guide to MIDP 2.0*
- Bruce Hopkins and Ranjith Antony, *Bluetooth for Java*

Websites:

- [http://www.springer-ny.com](http://www.springer-ny.com)
- [http://www.apress.com](http://www.apress.com)
- [http://www.mkp.com](http://www.mkp.com)
APPENDIX

1. ClientMidlet Class

```java
import java.lang.*;
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.microedition.rms.RecordStoreException;
import javax.bluetooth.*;

public class ClientMIDlet extends BluetoothMIDlet implements CommandListener {

    private Form mForm;
    private TextField mUserField, mPasswordField;
    private OutputStream out;
    private Command CMD_EXIT = new Command("Exit", Command.EXIT, 1);
    private Command CMD_BACK = new Command("Back", Command.BACK, 1);
    private Command CMD_ADD = new Command("Add", Command.SCREEN, 1);
    private Command CMD_DELETE = new Command("Delete", Command.SCREEN, 1);
    private Command CMD_MODIFY = new Command("Modify", Command.SCREEN, 1);
    private Command CMD_SEARCH = new Command("Search", Command.SCREEN, 1);
```
// Ok button creation
private static final Command CMD_OK = new Command("OK", Command.SCREEN, 1);

// Next button creation
private static final Command CMD_NEXT = new Command("Employee Window", Command.SCREEN, 1);
// variable for used to display the window
private Display display;
// Creating the additional forms
private Form mainForm, addForm, deleteForm, modifyForm, searchForm;
// The window display in first time is true or false
private boolean firstTime;
// Employee name Field creation
private TextField EmployeeNameField;
// Employee id Field creation
private TextField EmployeeIdField;
// This variable creating the window has been closed
private boolean hasToBeClosed;
// This variable for creating login function is true or false
private boolean isLogin;
// This function for creating add employee function is true or false
private boolean isAdd;
// This function for creating delete employee function is true or false
private boolean isDelete;
// This function for creating modify employee function is true or false
private boolean isModify;
// This function for creating search employee function is true or false
private boolean isSearch;

/*
 * In the class User interaction form are created and the login database
 * Employee database also access from the server.
 *
 */
public ClientMIDlet() {
	// Login form creation
	mForm = new Form("Login");
	isLogin = true;
	isAdd = false;
	isDelete = false;
	isModify = false;
	isSearch = false;
	// Username field creation
mUserField = new TextField("Name", ",", 32, 0);
// Password field creation
mPasswordField = new TextField("Password", ",", 32, 0);
// Fields are added to the form
mForm.append(mUserField);
mForm.append(mPasswordField);
// Commands are added to the form
mForm.addCommand(new Command("Login", Command.OK, 0));
mForm.addCommand(new Command("Exit", Command.EXIT, 0));
mForm.addCommand(CMD_NEXT);
mForm.setCommandListener(this);
Display.getDisplay(this).setCurrent(mForm);
}
public void run () {
    while (!hasToBeClosed) {
        ConnectToServer();
    }
}
// MIDlet is moved to paused state from active state
public void pauseApp () {
    notifyDestroyed();// MIDlet uses to notify it has entered the destroyed state
}

// MIDlet is to enter the destroyed state
public void destroyApp (boolean unconditional) {}
// In the class Establishing the client and server connection
private void connectToServer () {
    try {
        // Retrieve the connection string to connect to the server
        LocalDevice local = LocalDevice.getLocalDevice();
        DiscoveryAgent agent = local.getDiscoveryAgent();
        String connString = agent.selectService(new UUID
("86b4d249fb8844d6a756ec265dd1f6a3", false), ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
        if (connString != null) {
            try {
                // Connect to the server
                StreamConnection conn = (StreamConnection)Connector.open(connString);
                out = conn.openOutputStream();
                // Close the Connection
                conn.close();
            } catch (IOException e) {} // End of the if statement
        } catch (BluetoothStateException e) {
            } catch (IOException e) {
            // End of the if statement
        } catch (BluetoothStateException e) {
public void CommandAction(Command c, Displayable s) {
    // It shows the Graphical user interface window...
    if (c == CMD_NEXT) {
        display = Display.getDisplay(this);
        firstTime = true;
        if (firstTime) {
            // Employee Manager in the main Form creation
            mainForm = new Form("Employee Manager");
            // Commands are added to the mainForm
            mainForm.addCommand(CMD_ADD);
            mainForm.addCommand(CMD_DELETE);
            mainForm.addCommand(CMD_MODIFY);
            mainForm.addCommand(CMD_SEARCH);
            mainForm.addCommand(CMD_EXIT);
            mainForm.setCommandListener(this);
            firstTime = false;
        }
        display.setCurrent(mainForm);
    }
    display.setCurrent(mainForm);
}

if (c.getCommandType() == Command.EXIT) {
    String test = "-1# | #0";
    System.out.println("Sending " + test);
    try {
        out.write(test.getBytes());
        out.close();
    } catch (IOException e) {
    }
    hasToBeClosed = true;
    destroyApp(true);
}

// LoginForm "ok" button command action
if (c.getCommandType() == Command.OK) {
    boolean isSuccess = false;
    System.out.println("Login Action...");
    if(out != null) {
        if(isLogin) {
            String test = "0#" + mUserField.getString() + "|" + mPasswordField.getString() + "+#1";
            System.out.println("Sending " + test);
            try {
            } catch (IOException e) {
            }
        }
        hasToBeClosed = true;
        destroyApp(true);
    }
}
out.write(test.getBytes());
out.close();
} catch (IOException e) { }

//Employee Form "ok" button command action
if(c == CMD_OK) {
boolean isSuccess = false;
if(out != null) {
if(isAdd) {
String dataToAdd = "1#" +
EmployeeIdField.getString() + "|"
+EmployeeNameField.getString() + "#1";//new statement
System.out.println("The data to add is " +
dataToAdd);
try {
out.write(dataToAdd.getBytes());
out.close();
} catch (IOException e) { }
} else if(isDelete) {
String dataToDelete = "2#" +
EmployeeIdField.getString() + "|" +
EmployeeNameField.getString() +"#1";//new statement
System.out.println("The data to delete is " +
dataToDelete);
try {
out.write(dataToDelete.getBytes());
out.close();
} catch (IOException e) { }
} else {
System.out.println("Unknown action...");
}
}

//Add Button Command action
if (c == CMD_ADD) {
isLogin = false;
isAdd = true;
isDelete = false;
// AddForm creation
addForm = new Form ("Add Employee");
EmployeeIdField = new TextField("Employee ID:", null, 12,
TextField.ANY);
addForm.append(EmployeeIdField);
EmployeeNameField = new TextField("Employee Name:", null, 12,
TextField.ANY);
addForm.append(EmployeeNameField);
// Commands are added in addForm
addForm.addCommand(CMD_BACK);
addForm.addCommand(CMD_OK);
addForm.setCommandListener(this);

display.setCurrent(addForm);

}// end of the addCommand action

// Delete Button Command action

if(c == CMD_DELETE) {
isLogin = false;
isAdd = false;
isDelete = true;
// DeleteForm creation
deleteForm = new Form("Delete Employee");
EmployeeIdField = new TextField("Employee ID:", null, 12,
TextField.ANY);
deleteForm.append(EmployeeIdField);
EmployeeNameField = new TextField("Employee Name:", null, 12,
TextField.ANY);
deleteForm.append(EmployeeNameField);
// Commands are added to deleteForm
deleteForm.addCommand(CMD_BACK);
deleteForm.addCommand(CMD_OK);
deleteForm.setCommandListener(this);
display.setCurrent(deleteForm);
}// end of the deleteCommand action

if(c == CMD_MODIFY) {
isLogin = false;
isAdd = false;
isDelete = false;
isModify = true;
isSearch = false;
// ModifyForm creation
modifyForm = new Form("Modify Employee");
EmployeeIdField = new TextField("Employee ID:", null, 12,
TextField.ANY);
}
modifyForm.append(EmployeeIdField);
EmployeeNameField= new TextField("Employee Name:",null, 12,
TextField.ANY);
modifyForm.append(EmployeeNameField);
//Commands are added to modifyForm
modifyForm.addCommand(CMD_BACK);
modifyForm.addCommand(CMD_OK);
modifyForm.setCommandListener(this);
display.setCurrent(modifyForm);
}
// end of the modifyCommand action
if(c == CMD_SEARCH) {
    isLogin  = false;
    isAdd    = false;
    isDelete = false;
    isModify = false;
    isSearch = true;
    //SearchForm creation
    searchForm = new Form("Search Employee");
    EmployeeIdField= new TextField("Employee ID:", null, 12,
    TextField.ANY);
    searchForm.append(EmployeeIdField);
    EmployeeNameField= new TextField("Employee Name:", null, 12,
    TextField.ANY);
    searchForm.append(EmployeeNameField);
    //Commands are added to searchForm
    searchForm.addCommand(CMD_BACK);
    searchForm.addCommand(CMD_OK);
    searchForm.setCommandListener(this);
    display.setCurrent(searchForm);
}
// end of the searchCommand action

if (c == CMD_BACK) {
    display.setCurrent(mainForm);
}
// End of the command action
}//end of the ClientMIDlet class

2. ServerMIDlet Class

import java.lang.*;
import java.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.microedition.midlet.*;
import javax.bluetooth.*;
import javax.microedition.rms.*/;
public class ServerMIDlet extends BluetoothMIDlet implements CommandListener {

    //**********************************************************
    // Variable Declaration
    //**********************************************************
    //Variable creation for logiManager
    private LoginDataStoreManager loginManager;
    //Variable creation for employeeManager
    private EmployeeDataStoreManager employeeManager;
    //Variable for mainForm creation
    private Form mainForm;
    private boolean isOver;
    //Variable for inputStream creation
    private InputStream in;
    //Variable for ByteArray creation
    private ByteArrayOutputStream out;
    //Variable for Streamconnection creation
    private StreamConnection conn;
    //Variable for streamconnection notifier
    private StreamConnectionNotifier notifier;
    private boolean isValid;

    /*
     *This class represents the server form creation and login Database and
     *Employee Database are maintained
     */

    public ServerMIDlet() {
        loginManager = new LoginDataStoreManager();
        isOver = false;
        isValid = false
        employeeManager = new EmployeeDataStoreManager();
        // Create a Form and add the Exit command to the Form
        mainForm = new Form("Server");
        mainForm.addCommand(new Command("Exit", Command.EXIT, 0));
        mainForm.add Command(new Command("Ok", Command.OK,1));
        mainForm.setCommandListener(this);
        Display.getDisplay(this).setCurrent(mainForm);
    }
    public void run() {
        while(!isOver) {//new statement
            connect();
        }
    }
}
private void connect() {
    try {
        // Make the local device discoverable for the
        // client to locate
        LocalDevice local = LocalDevice.getLocalDevice();
        if (!local.setDiscoverable(DiscoveryAgent.GIAC)) {
            mainForm.append("Failed to change to the " + "discoverable
            mode");
            return;
        }
        // Create a server connection object to accept
        // a connection from a client
        notifier = (StreamConnectionNotifier) Connector.open("btspp://localhost:
        +"86b4d249fb8844d6a756ec265dd1f6a3");
        // Accept a connection from the client
        conn = notifier.acceptAndOpen();
        serveClient();
    } catch (BluetoothStateException e) {
        mainForm.append("BluetoothStateException: ");
        mainForm.append(e.getMessage());
    } catch (IOException e) {
        mainForm.append("IOException: ");
        mainForm.append(e.getMessage());
    }
}
private void serveClient() {
    try {
        // Open the input to read data from
        in = conn.openInputStream();
        out = new ByteArrayOutputStream();
        // Read the data sent from the client until
        // the end of stream
        int data;
        while ((data = in.read()) != -1) {
            out.write(data);
        }
        String dataReceived = out.toString();
        // Add the text sent from the client to the Form
        String caseValue = dataReceived.substring(0,
        dataReceived.indexOf("#"));
    } catch (IOException e) {
        mainForm.append("IOException: ");
        mainForm.append(e.getMessage());
    }
}
dataReceived = dataReceived.substring(dataReceived.indexOf('#') + 1);
String actualData = dataReceived.substring(0, 
dataReceived.indexOf('#'));
dataReceived = dataReceived.substring(dataReceived.indexOf('#') + 1);
String booleanData = dataReceived;
System.out.println("Data Received at server = " + caseValue + ", "
+ actualData + ", " + booleanData);
boolean isAdded = false;
boolean isDeleted = false;
boolean isModified = false;
boolean isSearched = false;
if(booleanData.equals("0")) {
    isOver = true;
} else {
    isOver = false;
}
if(caseValue.equals("0")) {
    isValid = validateUser(actualData);
    mainForm.append("Welcome " + actualData.substring(0,
        actualData.indexOf('|')));
    System.out.println("is Valid = " + isValid);
} else if(caseValue.equals("1") && isValid){
    //add employee
    isAdded = addEmployee(actualData);
    try {
        employeeManager.showRecords();
    } catch(RecordStoreException rse) {
    }
} else if(caseValue.equals("2") && isValid){
    //delete employee
    isDeleted = deleteEmployee(actualData);
    try {employeeManager.showRecords();
        } catch(RecordStoreException rse) {
    }
} else if(caseValue.equals("3") && isValid){
    //modify employee
    isModified = modifyEmployee(actualData);
    try {
        employeeManager.showRecords();
    } catch(RecordStoreException rse) {
    }
} else if(caseValue.equals("4") && isValid){
    //search employee
    isSearched = searchEmployee(actualData);
try {
    employeeManager.showRecords();
} catch (RecordStoreException rse) {

}

// Close all open resources
in.close();
out.close();
conn.close();
notifier.close();

catch (BluetoothStateException e) {
    mainForm.append("BluetoothStateException: ");
    mainForm.append(e.getMessage());
}
catch (IOException e) {
    mainForm.append("IOException: ");
    mainForm.append(e.getMessage());
}

public void pauseApp() {
    notifyDestroyed();
}

public void destroyApp(boolean unconditional) {
    // code that react the action
    public void commandAction(Command c, Displayable s) {
        if (c.getCommandType() == Command.EXIT) {
            destroyApp(true);
        }
    }
    // end of the Command action function
    private boolean validateUser(String data) {
        boolean isSuccess = false;
        try {
            isSuccess = loginManager.validateUser(data);
        } catch (RecordStoreException rse) {

        }
        return isSuccess;
    }
    // end of the validateUser function
    private boolean addEmployee(String data) {
        boolean isExisting = false;
        try {
            isExisting = employeeManager.addEmployee(data);
        } catch (RecordStoreException rse) {
            System.out.println("Cannot Add " + data);
        }
        return isExisting;
private boolean deleteEmployee(String data) {
    boolean isSuccess = false;
    try {
        isSuccess = employeeManager.deleteEmployee(data);
    } catch (RecordStoreException rse) {
        System.out.println("Cannot Delete " + data);
    }
    return isSuccess;
}

private boolean modifyEmployee(String data) {
    boolean isSuccess = false;
    try {
        isSuccess = employeeManager.modifyEmployee(data);
    } catch (RecordStoreException rse) {
        System.out.println("Cannot Modify " + data);
    }
    return isSuccess;
}

private boolean searchEmployee(String data) {
    boolean isSuccess = false;
    try {
        isSuccess = employeeManager.searchEmployee(data);
    } catch (RecordStoreException rse) {
        System.out.println("Cannot Search " + data);
    }
    return isSuccess;
}

import javax.microedition.rms.*;
public class LoginDataStoreManager {
    public LoginDataStoreManager() {
    }

    public boolean validateUser(String test) throws RecordStoreException {
        RecordStore rs = null;
        RecordEnumeration re = null;
        boolean isSuccess = false;
        System.out.println("Data received for login validation = " + test);
        try {
            rs = RecordStore.openRecordStore("LoginStore", true);
            re = rs.enumerateRecords(null, null, false);
            while (re.hasNextElement()) {
            }
        }
    }
}
byte[] raw = re.nextRecord();
String data = new String(raw);
System.out.println("Data in data store = " + data);
if(test.equals(data)) {
    isSuccess = true;
    break;
}
}
finally {
    if (re != null) {
        re.destroy();
    }
    if (rs != null) {
        rs.closeRecordStore();
    }
}
return isSuccess;
} // end of the validateUser class
} // end of the LoginDataStoreManager class

4. EmployeeDataStoreManager Class

import javax.microedition.rms.*;
public class EmployeeDataStoreManager {
    private static String dataStore = "EmployeeStore";
    public EmployeeDataStoreManager() {
    }
    public boolean addEmployee(String employee) throws RecordStoreException {
        RecordStore rs = null;
        RecordEnumeration re = null;
        boolean isExisting = false;
        String toAdd = employee;
        try {
            rs = RecordStore.openRecordStore(dataStore, true);
            re = rs.enumerateRecords(null, null, false);
            while (re.hasNextElement()) {
                byte[] raw = re.nextRecord();
                String data = new String(raw);
                if(toAdd.equals(data)) {
                    isExisting = true;
                    break;
                }
            }
            if(!isExisting) {
                byte[] dataToAdd = toAdd.getBytes();
            }
        } finally {
            if (re != null) {
                re.destroy();
            }
            if (rs != null) {
                rs.closeRecordStore();
            }
        }
        return isSuccess;
    }
} // end of the EmployeeDataStoreManager class
rs.addRecord(dataToAdd, 0, dataToAdd.length);
}
}
finally {
  if (re != null) {
    re.destroy();
  }
  if (rs != null) {
    rs.closeRecordStore();
  }
}
return isExisting;
} // end of the addEmployee class

public boolean deleteEmployee(String employee) throws RecordStoreException {
  RecordStore rs = null;
  RecordEnumeration re = null;
  RecordEnumeration re1 = null;
  boolean isSuccess = false;
  int id = 0;
  System.out.println("In Employee manager: Rcvd data = " + employee);
  try {
    rs = RecordStore.openRecordStore(dataStore, true);
    re = rs.enumerateRecords(null, null, false);
    re1 = rs.enumerateRecords(null, null, false);
    while (re.hasNextElement()) {
      byte[] raw = re.nextRecord();
      String data = new String(raw);
      id = re1.nextRecordId();
      System.out.println("Data in DB = " + data + ", id = " + id);
      if(employee.equals(data)) {
        rs.deleteRecord(id);
        isSuccess = true;
        break;
      }
    }
  }
  finally {
    if (re != null) {
      re.destroy();
    }
    if (re1 != null) {
      re1.destroy();
    }
    if (rs != null) {
      rs.closeRecordStore();
    }
  }
} // end of the deleteEmployee class
public boolean modifyEmployee(String employee) throws RecordStoreException {
    RecordStore rs = null;
    RecordEnumeration re = null;
    RecordEnumeration re1 = null;
    boolean isSuccess = false;
    int id = 0;
    System.out.println("In Employee manager: Rcvd data = " + employee);
    try {
        rs = RecordStore.openRecordStore(dataStore, true);
        re = rs.enumerateRecords(null, null, false);
        re1 = rs.enumerateRecords(null, null, false);
        while (re.hasNextElement()) {
            byte[] raw = re.nextRecord();
            String data = new String(raw);
            id = re1.nextRecordId();
            System.out.println("Data in DB = " + data + ", id = " + id);
            if(employee.equals(data)) {
                rs.deleteRecord(id);
                isSuccess = true;
                break;
            }
        }
    } finally {
        if (re != null) {
            re.destroy();
        }
        if (re1 != null) {
            re1.destroy();
        }
        if (rs != null) {
            rs.closeRecordStore();
        }
    }
    return isSuccess;
} // end of the modifyEmployee class

public boolean searchEmployee(String employee) throws RecordStoreException {
    RecordStore rs = null;
    RecordEnumeration re = null;
    RecordEnumeration re1 = null;
    boolean isSuccess = false;
    int id = 0;
    System.out.println("In Employee manager: Rcvd data = " + employee);
    try {
        rs = RecordStore.openRecordStore(dataStore, true);
        re = rs.enumerateRecords(null, null, false);
        re1 = rs.enumerateRecords(null, null, false);
        while (re.hasNextElement()) {
            byte[] raw = re.nextRecord();
            String data = new String(raw);
            id = re1.nextRecordId();
            System.out.println("Data in DB = " + data + ", id = " + id);
            if(employee.equals(data)) {
                rs.deleteRecord(id);
                isSuccess = true;
                break;
            }
        }
    } finally {
        if (re != null) {
            re.destroy();
        }
        if (re1 != null) {
            re1.destroy();
        }
        if (rs != null) {
            rs.closeRecordStore();
        }
    }
    return isSuccess;
} // end of the searchEmployee class

public boolean deleteEmployee() throws RecordStoreException {
    RecordStore rs = null;
    RecordEnumeration re = null;
    RecordEnumeration re1 = null;
    boolean isSuccess = false;
    int id = 0;
    System.out.println("In Employee manager: Rcvd data = " + employee);
    try {
        rs = RecordStore.openRecordStore(dataStore, true);
        re = rs.enumerateRecords(null, null, false);
        re1 = rs.enumerateRecords(null, null, false);
        while (re.hasNextElement()) {
            byte[] raw = re.nextRecord();
            String data = new String(raw);
            id = re1.nextRecordId();
            System.out.println("Data in DB = " + data + ", id = " + id);
            if(employee.equals(data)) {
                rs.deleteRecord(id);
                isSuccess = true;
                break;
            }
        }
    } finally {
        if (re != null) {
            re.destroy();
        }
        if (re1 != null) {
            re1.destroy();
        }
        if (rs != null) {
            rs.closeRecordStore();
        }
    }
    return isSuccess;
} // end of the deleteEmployee class

// end of the deleteEmployee class

public boolean searchEmployee(String employee) throws RecordStoreException {
    RecordStore rs = null;
    RecordEnumeration re = null;
    RecordEnumeration re1 = null;
    boolean isSuccess = false;
    int id = 0;
    System.out.println("In Employee manager: Rcvd data = " + employee);
    try {
        rs = RecordStore.openRecordStore(dataStore, true);
        re = rs.enumerateRecords(null, null, false);
        re1 = rs.enumerateRecords(null, null, false);
        while (re.hasNextElement()) {
            byte[] raw = re.nextRecord();
            String data = new String(raw);
            id = re1.nextRecordId();
            System.out.println("Data in DB = " + data + ", id = " + id);
            if(employee.equals(data)) {
                rs.deleteRecord(id);
                isSuccess = true;
                break;
            }
        }
    } finally {
        if (re != null) {
            re.destroy();
        }
        if (re1 != null) {
            re1.destroy();
        }
        if (rs != null) {
            rs.closeRecordStore();
        }
    }
    return isSuccess;
} // end of the searchEmployee class

// end of the searchEmployee class
int id = 0;
System.out.println("In Employee manager: Rcvd data = " + employee);
try {
    rs = RecordStore.openRecordStore(dataStore, true);
    re = rs.enumerateRecords(null, null, false);
    re1 = rs.enumerateRecords(null, null, false);
    while (re.hasNextElement()) {
        byte[] raw = re.nextRecord();
        String data = new String(raw);
        id = re1.nextRecordId();
        System.out.println("Data in DB = " + data + ", id = " + id);
        if(employee.equals(data)) {
            rs.deleteRecord(id);
            isSuccess = true;
            break;
        }
    }
} finally {
    if (re != null) {
        re.destroy();
    }
    if (re1 != null) {
        re1.destroy();
    }
    if (rs != null) {
        rs.closeRecordStore();
    }
}
return isSuccess;
} // end of the searchEmployee class
public void showRecords() throws RecordStoreException {
    RecordStore rs = null;
    RecordEnumeration re = null;
    RecordEnumeration re1 = null;
    try {
        rs = RecordStore.openRecordStore(dataStore, true);
        re = rs.enumerateRecords(null, null, false);
        re1 = rs.enumerateRecords(null, null, false);
        int num = re.numRecords();
        int id = -1;
        System.out.println("Number of records = " + num);
        while (re.hasNextElement()) {
            byte[] raw = re.nextRecord();
        }
    } finally {
        if (re != null) {
            re.destroy();
        }
        if (re1 != null) {
            re1.destroy();
        }
        if (rs != null) {
            rs.closeRecordStore();
        }
    }
}
A Master – Slave Communication System Using Java Bluetooth

id = re1.nextRecordId();
String data = new String(raw);
System.out.println("Record = "+id+": "+data);
}
} finally {
if (re != null) {
    re.destroy();
}
if (re1 != null) {
    re1.destroy();
}
if (rs != null) {
    rs.closeRecordStore();
}
} // end of the showRecord class
} // end of the EmployeeDataStoreManager class

5. BlueToothMIDlet class
import java.lang.*;
import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.bluetooth.*;
public class BluetoothMIDlet extends MIDlet implements Runnable, CommandListener {
    public BluetoothMIDlet() {
    }
    /**
     * Starts a background thread when the MIDlet is
     * started.
     */
    public void startApp() throws MIDletStateException {
        new Thread(this).start();
    }
    /**
     * This method called when MIDlet is moved to paused state
     * from active state
     */
    public void pauseApp() {
    }
    /**
     * This method called when MIDlet is enter the
     * destroyed state
     */

Dept. of Computer Science, Nehru Memorial College, Bharathidasan University
/**
 * Destroys the MIDlet when a Command occurs.
 */
public void commandAction(Command c, Displayable d) {
    notifyDestroyed();
}
} // end of the BlueToothMIDlet class

6. RecordMIDlet Class

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.RecordStoreException;
public class RecordMIDlet extends MIDlet implements CommandListener {
    private Preferences mPreferences;
    private Form mForm;
    private TextField mUserField, mPasswordField;
    public RecordMIDlet() {
        try {
            mPreferences = new Preferences("LoginDataStore");
        } catch (RecordStoreException rse) {
            mForm = new Form("Exception");
            mForm.append(new StringItem(null, rse.toString()));
            mForm.addCommand(new Command("Exit", Command.EXIT, 0));
            mForm.setCommandListener(this);
            return;
        }
        mForm = new Form("Login");
        mUserField = new TextField("Name", "", 32, 0);
        mPasswordField = new TextField("Password", "", 32, 0);
        mForm.append(mUserField);
        mForm.append(mPasswordField);
        mForm.addCommand(new Command("Add", Command.OK, 0));
        mForm.addCommand(new Command("Exit", Command.EXIT, 1));
        mForm.setCommandListener(this);
    }
    public void startApp() {
        Display.getDisplay(this).setCurrent(mForm);
    }
    public void pauseApp() {
        notifyDestroyed();
    }
    public void destroyApp(boolean unconditional) {
    }
    public void run() {
    }
}
public void destroyApp(boolean unconditional) {
}
public void commandAction(Command c, Displayable s) {
    if (c.getCommandType() == Command.EXIT) {
        destroyApp(true);
    } else if(c.getCommandType() == Command.OK) {
        try {
            System.out.println("Record Store before update...");
            mPreferences.showRecords();
            mPreferences.add(mUserField.getString(), mPassword.Field.getString());
            System.out.println("Record Store after update...");
            mPreferences.showRecords();
        } catch (RecordStoreException rse) {}
    }
} //end of the command action
} //end of the RecordMIDlet class

7. Preferences Class

import java.util.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.*;
public class Preferences {
    private String mRecordStoreName;
    public Preferences(String recordStoreName) throws RecordStoreException {
        mRecordStoreName = recordStoreName;
    }
    public void add(String userName, String password) throws RecordStoreException {
        RecordStore rs = null;
        RecordEnumeration re = null;
        boolean isExisting = false;
        String toAdd = userName + "|" + password;
        try {
            rs = RecordStore.openRecordStore(mRecordStoreName, true);
            re = rs.enumerateRecords(null, null, false);
            while (re.hasNextElement()) {
                byte[] raw = re.nextRecord();
                String data = new String(raw);
                if(toAdd.equals(data)) {
                    isExisting = true;
                    break;
                }
            }
        } catch (RecordStoreException rse) {}
    }
}
public void showRecords() throws RecordStoreException {
    RecordStore rs = null;
    RecordEnumeration re = null;
    try {
        rs = RecordStore.openRecordStore(mRecordStoreName, true);
        re = rs.enumerateRecords(null, null, false);
        while (re.hasNextElement()) {
            byte[] raw = re.nextRecord();
            String data = new String(raw);
            System.out.println("Record = " + data);
        }
    } finally {
        if (re != null) re.destroy();
        if (rs != null) rs.closeRecordStore();
    }
} // End of the show Record function

} // End of the Preferences class