

Now compatible with  
**FLOWCODE7**

## Bluetooth Communications



EB639-80-11

**MATRIX**  
[www.matrixtsl.com](http://www.matrixtsl.com)

Copyright © 2014 Matrix Technology Solutions Limited



**EB639**

# **Bluetooth Solution**

## **Course notes**

# Contents

|       |  |    |
|-------|--|----|
| 1     | Introduction.....  | 7  |
| 1.1   | Structure of these notes .....                                 | 7  |
| 1.2   | Learning outcomes.....   | 7  |
| 1.3   | Using this manual.....   | 8  |
| 1.3.1 | An introduction to the Practical implementation sections ..... | 8  |
| 2     | What do we mean by Bluetooth? .....                            | 9  |
| 2.1   | BLU2i.....   | 9  |
| 2.2   | AT Commands.....   | 9  |
| 2.3   | Hardware and software used in the course.....                  | 9  |
| 2.3.1 | E-Blocks solution .....  | 9  |
| 2.3.2 | Flowcode.....  | 10 |
| 2.3.3 | Additional PC software (EZURiO terminal).....                  | 10 |
| 2.4   | Additional useful devices.....                                 | 10 |
| 2.4.1 | Onboard Bluetooth/USB dongle .....                             | 10 |
| 2.4.2 | Bluetooth headset.....   | 10 |
| 2.4.3 | Bluetooth enabled phone .....                                  | 10 |
| 3     | Getting started .....  | 11 |
| 3.1   | Setting up the hardware .....                                  | 11 |
| 3.1.1 | E-Blocks solution .....  | 11 |
| 3.1.2 | Setting up the EB024 Bluetooth board.....                      | 12 |
| 3.1.3 | Adding the Voice Codec board.....                              | 12 |
| 3.1.4 | Standard Settings .....  | 12 |
| 3.2   | Introduction to Flowcode .....                                 | 13 |
| 3.2.1 | Starting out in Flowcode .....                                 | 13 |
| 3.3   | The Bluetooth component.....                                   | 13 |
| 3.3.1 | Properties and pin connection .....                            | 14 |
| 3.3.2 | Bluetooth component macros .....                               | 14 |
| 3.3.3 | Creating and sending AT scripts .....                          | 15 |
| 3.4   | Testing the hardware .....                                     | 15 |
| 4     | Bluetooth theory and background.....                           | 16 |
| 4.1   | Introduction to Bluetooth .....                                | 16 |
| 4.1.1 | Brief history.....   | 16 |
| 4.1.2 | Bluetooth concepts.....  | 16 |
| 4.1.3 | Bluetooth advantages .....                                     | 17 |
| 4.1.4 | Bluetooth disadvantages .....                                  | 17 |
| 4.2   | Protocols and the OSI model.....                               | 17 |
| 4.2.1 | Application .....  | 19 |
| 4.2.2 | Bluetooth layers .....   | 19 |
| 4.2.3 | Hardware details.....  | 20 |
| 4.2.4 | Profiles .....   | 22 |
| 5     | Discovery.....   | 23 |
| 5.1   | Theory: Finding other Bluetooth devices .....                  | 23 |
| 5.1.1 | The Inquiry command .....                                      | 23 |
| 5.1.2 | Additional Inquiry commands and the <devclass> parameter ..... | 23 |
| 5.1.3 | Discovery example .....  | 24 |
| 5.2   | Exercise 1: Discovering Bluetooth devices .....                | 25 |
| 5.2.1 | Introduction .....   | 25 |
| 5.2.2 | Objectives .....   | 25 |
| 5.2.3 | Pre-requisites .....   | 25 |
| 5.2.4 | Hardware/Software requirements .....                           | 25 |
| 5.2.5 | Exercise information.....                                      | 25 |
| 5.2.6 | Learning outcome.....  | 25 |
| 5.2.7 | Additional tasks.....  | 26 |
| 5.3   | Practical implementation: Discovering Bluetooth devices.....   | 27 |
| 5.3.1 | Discovering and discoverable.....                              | 27 |
| 5.3.2 | Planning the program.....                                      | 27 |

|        |   |    |
|--------|---|----|
| 5.3.3  | Required macros .....                                       | 27 |
| 5.3.4  | Initializing .....  | 28 |
| 5.3.5  | Sending a command .....                                     | 28 |
| 5.3.6  | Checking for responses .....                                | 29 |
| 5.3.7  | Running the complete program .....                          | 31 |
| 6      | Discoverability .....                                       | 32 |
| 6.1    | Theory: Discoverability .....                               | 32 |
| 6.1.1  | Configuring for start up .....                              | 32 |
| 6.1.2  | Using S registers .....                                     | 32 |
| 6.1.3  | Using AT commands .....                                     | 32 |
| 6.1.4  | S Registers or AT Commands? .....                           | 32 |
| 6.1.5  | Making a device discoverable .....                          | 33 |
| 6.1.6  | Baud rate and settings .....                                | 33 |
| 6.1.7  | Discoverable and Connectable .....                          | 33 |
| 6.1.8  | Number of rings before answering .....                      | 33 |
| 6.1.9  | Whether to allow remote command for data sending .....      | 33 |
| 6.1.10 | Saving the configuration .....                              | 33 |
| 6.1.11 | Implementing the configuration .....                        | 33 |
| 6.1.12 | A basic configuration set up .....                          | 34 |
| 6.2    | Exercise 2: Discoverability .....                           | 35 |
| 6.2.1  | Introduction .....  | 35 |
| 6.2.2  | Objectives .....  | 35 |
| 6.2.3  | Pre-requisites .....  | 35 |
| 6.2.4  | Hardware/Software requirements .....                        | 35 |
| 6.2.5  | Exercise information .....                                  | 35 |
| 6.2.6  | Learning outcome .....                                      | 35 |
| 6.2.7  | Additional tasks .....                                      | 36 |
| 6.3    | Practical implementation: Discoverability .....             | 37 |
| 6.3.1  | Continuing development .....                                | 37 |
| 6.3.2  | Configuring the Bluetooth device .....                      | 37 |
| 6.3.3  | Single commands and scripts .....                           | 37 |
| 6.3.4  | Running and demonstrating the program .....                 | 39 |
| 7      | Connecting Bluetooth devices .....                          | 40 |
| 7.1    | Theory: Connecting – Addresses .....                        | 40 |
| 7.2    | Exercise 3: Connecting to a device .....                    | 41 |
| 7.2.1  | Introduction .....  | 41 |
| 7.2.2  | Objectives .....  | 41 |
| 7.2.3  | Pre-requisites .....  | 41 |
| 7.2.4  | Hardware/Software requirements .....                        | 41 |
| 7.2.5  | Exercise information .....                                  | 41 |
| 7.2.6  | Learning outcome .....                                      | 41 |
| 7.2.7  | Further work .....  | 41 |
| 7.3    | Practical implementation: Connecting .....                  | 42 |
| 7.3.1  | Resetting the systems .....                                 | 42 |
| 8      | Passkeys and Pairing .....                                  | 43 |
| 8.1    | Theory: Passkeys and Pairing .....                          | 43 |
| 8.1.1  | Sending the Passkey command .....                           | 43 |
| 8.1.2  | Initiating pairing .....                                    | 43 |
| 8.1.3  | When to send the Passkey command .....                      | 43 |
| 8.2    | Exercise 4: Passkeys and Pairing .....                      | 44 |
| 8.2.1  | Introduction .....  | 44 |
| 8.2.2  | Objectives .....  | 44 |
| 8.2.3  | Pre-requisites .....  | 44 |
| 8.2.4  | Hardware/Software requirements .....                        | 44 |
| 8.2.5  | Exercise information .....                                  | 44 |
| 8.2.6  | Learning outcome .....                                      | 44 |
| 8.2.7  | Further work .....  | 44 |
| 8.3    | Practical implementation: Passkeys and pairing .....        | 45 |
| 8.3.1  | The basic program .....                                     | 45 |
| 8.3.2  | Advanced features: Choosing what device to connect to ..... | 45 |

|        |   |    |
|--------|---|----|
| 8.3.3  | Resetting the systems.....                            | 46 |
| 9      | Checking responses .....                              | 47 |
| 9.1    | Theory: Checking responses.....                       | 47 |
| 9.1.1  | Solicited and unsolicited responses.....              | 47 |
| 9.1.2  | Response handling macros.....                         | 47 |
| 9.1.3  | Command and response sequences .....                  | 48 |
| 9.2    | Exercise 5: Checking responses.....                   | 50 |
| 9.2.1  | Introduction .....                                    | 50 |
| 9.2.2  | Objectives .....                                      | 50 |
| 9.2.3  | Pre-requisites .....                                  | 50 |
| 9.2.4  | Hardware/Software requirements .....                  | 50 |
| 9.2.5  | Exercise information.....                             | 50 |
| 9.2.6  | Learning outcome .....                                | 50 |
| 9.3    | Practical implementation: Checking responses .....    | 52 |
| 9.3.1  | Establishing the expected sequence .....              | 52 |
| 9.3.2  | Additional sequence considerations .....              | 52 |
| 9.3.3  | Using the WaitForResponse macro.....                  | 53 |
| 9.3.4  | Error checking methodology .....                      | 53 |
| 10     | Command modes - Sending data and commands.....        | 54 |
| 10.1   | Theory of Command modes .....                         | 54 |
| 10.1.1 | Data mode, Remote and local command mode.....         | 54 |
| 10.1.2 | Local command mode +++ and ^^ .....                   | 54 |
| 10.1.3 | Guard gaps and escape sequences .....                 | 54 |
| 10.2   | Command modes: Exercise 6 .....                       | 55 |
| 10.2.1 | Introduction .....                                    | 55 |
| 10.2.2 | Objectives .....                                      | 55 |
| 10.2.3 | Pre-requisites .....                                  | 55 |
| 10.2.4 | Hardware/Software requirements .....                  | 55 |
| 10.2.5 | Exercise information.....                             | 55 |
| 10.2.6 | Learning outcome .....                                | 55 |
| 10.2.7 | Further work .....                                    | 55 |
| 10.3   | Practical implementation: Command modes.....          | 56 |
| 10.3.1 | Which device is being sent commands? .....            | 56 |
| 10.3.2 | Program overview.....                                 | 57 |
| 11     | Audio communication .....                             | 58 |
| 11.1   | Theory: Audio communication.....                      | 58 |
| 11.1.1 | Hardware requirements .....                           | 58 |
| 11.1.2 | Establishing an Audio connection .....                | 58 |
| 11.2   | Exercise 7: Audio communication.....                  | 60 |
| 11.2.1 | Introduction .....                                    | 60 |
| 11.2.2 | Objectives .....                                      | 60 |
| 11.2.3 | Pre-requisites .....                                  | 60 |
| 11.2.4 | Hardware/Software requirements .....                  | 60 |
| 11.2.5 | Exercise information.....                             | 60 |
| 11.2.6 | Learning outcome .....                                | 60 |
| 11.2.7 | Further work .....                                    | 60 |
| 11.3   | Practical implementation: Audio communications .....  | 61 |
| 11.3.1 | Audio setup and considerations .....                  | 61 |
| 11.3.2 | Solo audio testing .....                              | 61 |
| 11.3.3 | Using headphones/microphones and the Codec board..... | 62 |
| 11.3.4 | Program notes.....                                    | 62 |
| 12     | Profiles – Headsets and Telephones .....              | 63 |
| 12.1   | Theory: Profiles.....                                 | 63 |
| 12.1.1 | Profile specific features.....                        | 63 |
| 12.1.2 | Device class .....                                    | 64 |
| 12.1.3 | Major and Minor Device Classes .....                  | 64 |
| 12.1.4 | Device class values.....                              | 64 |
| 12.1.5 | Setting the Device Class .....                        | 65 |
| 12.1.6 | ATD and the <uuid> parameter .....                    | 65 |
| 12.2   | Exercise 8: Headset profile.....                      | 66 |

|        |   |    |
|--------|---|----|
| 12.2.1 | Introduction .....                                  | 66 |
| 12.2.2 | Objectives .....                                    | 66 |
| 12.2.3 | Pre-requisites .....                                | 66 |
| 12.2.4 | Hardware/Software requirements .....                | 66 |
| 12.2.5 | Exercise information.....                           | 66 |
| 12.2.6 | Learning outcome.....                               | 66 |
| 12.2.7 | Further work .....                                  | 66 |
| 12.3   | Practical implementation: Profiles.....             | 67 |
| 12.3.1 | Implementing the Headset profile .....              | 67 |
| 12.3.2 | Setting up the headset .....                        | 67 |
| 12.3.3 | Special commands for the headset profile.....       | 67 |
| 12.3.3 | Sending profile related commands .....              | 68 |
| 12.3.4 | Checking for profile related commands.....          | 68 |
| 12.3.5 | Filtering for particular devices .....              | 68 |
| 13     | Trust and Security .....                            | 69 |
| 13.1   | Theory: Trust and Security.....                     | 69 |
| 13.1.1 | Security in general.....                            | 69 |
| 13.1.2 | Authentication .....                                | 69 |
| 13.1.3 | Encryption.....                                     | 70 |
| 13.1.4 | ATD and Authentication and Encryption .....         | 70 |
| 13.1.5 | Trust.....  | 70 |
| 13.1.6 | Trusted Devices AT Commands .....                   | 70 |
| 13.2   | Exercise 9: Trust and Security .....                | 72 |
| 13.2.1 | Introduction .....                                  | 72 |
| 13.2.2 | Objectives .....                                    | 72 |
| 13.2.3 | Pre-requisites .....                                | 72 |
| 13.2.4 | Hardware/Software requirements .....                | 72 |
| 13.2.5 | Exercise information.....                           | 72 |
| 13.2.6 | Learning outcome.....                               | 72 |
| 13.2.7 | Further work .....                                  | 72 |
| 13.3   | Practical implementation: Trust and Security.....   | 73 |
| 13.3.1 | General objectives.....                             | 73 |
| 13.3.2 | Additional features.....                            | 73 |
| 14     | Project design principles .....                     | 74 |
| 14.1   | Project - Baby monitor .....                        | 74 |
| 14.1.1 | Extending the project with additional features..... | 74 |
| 14.2   | Project - Medical datalogger.....                   | 75 |
| 14.2.1 | Extending the project .....                         | 75 |
| 15     | References and Appendix.....                        | 76 |
| 15.1   | References .....                                    | 76 |
| 15.1.1 | Books.....  | 76 |
| 15.1.2 | Websites .....                                      | 76 |
|        | AT command reference.....                           | 76 |
| 15.1.3 | Common AT Command parameters.....                   | 76 |
| 15.1.4 | Important AT Commands list .....                    | 76 |
| 15.1.5 | Important S registers .....                         | 77 |
| 15.2   | Appendix.....                                       | 78 |
| 15.2.1 | How to... .....                                     | 78 |



# PART 1: General Introduction and overview

## 1 Introduction

### 1.1 Structure of these notes

These notes are set out as follows:

Part 1: General introduction to Bluetooth and the Bluetooth solution

- Getting started – introduction to the hardware and software
- Bluetooth history and general overview

Part 2: The course

A series of progressive exercises to take students through the concepts and practice required in establishing Bluetooth communications.

The chapters are broken down into 3 sections for each chapter.

- Theory section
- Exercise description
- Practical notes

Part 3: References and Appendix

Ancillary chapters providing reference information.

Includes:

- Reference section
- A “How to” Programming reference section
- AT Commands section
  - Includes:
    - Common AT Commands
    - S Registers and Parameters

### 1.2 Learning outcomes

These teacher's notes are designed to introduce the concepts and strategies required for practical Bluetooth communications. In completing the exercises in this course students will learn about the following:

- How one Bluetooth device discovers another Bluetooth device and the options concerning discoverability
- How Bluetooth devices pair and set up a communications channel
- How data of various kinds is transferred between Bluetooth devices
- How Bluetooth is used for audio transmission
- How trust and security are handled by Bluetooth.
- How profiles can facilitate communication with other Bluetooth devices
- How to implement profiles like the Headset profile, LAN, OBEX, and Serial port.

These notes will not address the radio frequency characteristics and low level transmission characteristics and protocols of Bluetooth. These are all handled by an off the shelf Bluetooth module which shields users from such issues.

These notes are structured into a number of sections that first take you through setting up, configuring and testing the hardware and software into the background of Bluetooth and then into a series of Exercises and examples that take the student through the workings of Bluetooth.

The exercises should be carried out using Flowcode V7 or later, a graphical programming language. The Flowcode Bluetooth component is designed to allow students to learn about



Bluetooth without getting bogged down with the problems of programming in C or a lower level language.

It is envisaged that a student of some competence with Flowcode will need to spend around 20 hours to complete all exercises.

### 1.3 Using this manual

The main part of this manual is structured around a three part approach:

- Theory.  
The first part of a chapter introduces the topic at hand and discusses the theory behind it, explains the commands used and the general sequences and strategies required.
- Exercise.  
The second element is an exercise. The exercise is given here so that the aims and objectives are understood and borne in mind whilst reading through the next section.
- Practical implementation.  
The third section discusses the practical implementation of the exercise. Items discussed here include Flowcode macros used, Flowcode strategies and any other bits of information or advice needed for implementing the exercise objectives.

It is intended that students should first have the theory explained to them in the form of a lecture or handout. Students can then be given the exercise. Supervisors have a choice whether the practical implementation notes are handed out or not.

We would suggest that for each exercise the students are given the Exercise sheet(s) and the Practical implementation notes. Students should build the programs in Flowcode. Initial Practical implementation notes are quite detailed and provide a good deal of information on how the program should be constructed. Later Exercises are not provided with such detailed Practical implementation notes: the student must then use his/her knowledge to complete the tasks detailed in the Exercise.

The Bluetooth module datasheet is a key part of the Bluetooth solution: students will be expected to look up the meaning of AT commands and the functions of the Bluetooth registers. Accordingly students should be given a copy of the Bluetooth module datasheet which is on the EB617 CD ROM.

For most of the exercises a complete set of solution programs are provided. These take the form of two programs – one for each Bluetooth node.

#### 1.3.1 An introduction to the Practical implementation sections

This first practical section is in the form of a worked example with all the practical information provided and an example program built. Later practical sections will contain the additional information required to accomplish the exercise objectives, and will contain code snippets for new procedures and macros, but will not include full example programs. The students are assumed at this stage to be competent enough to be able to create the programs given the practical section information supplied.

It is intended that wherever possible the programs created by the student for the previous exercise are re-used for the current exercise. This illustrates both the evolution of the program as new steps are added, and to show the growing complexity and program flow of a full working system. This approach also provides students with core code that they are already familiar with. However the Student programs may need to be assessed at various points to ensure the code is adequate for the current project, and modifications or comments made as appropriate to steer student programs through the course as a whole. Such modifications may take the form of adapting parts of programs to macros to unclutter parts of the program, or for monitored and assessed code rewrites to implement element in a more organized and efficient manner.

## **2 What do we mean by Bluetooth?**

Bluetooth can be accessed at a number of levels. Bluetooth is both a communications technology, and a communications strategy. As a communications technology Bluetooth transforms data from an application to radio signals and back again. As a communications strategy Bluetooth is about discovering and linking to other Bluetooth devices, and accessing features on those devices such as audio communications.

This course will concentrate on the communications strategies of Bluetooth. This is when the end user will experience Bluetooth technologies – linking to headsets and sending data or audio signals between devices. Considerations such as security and trust will be covered, as will elements such as profiles and classes.

The Bluetooth communications technology is generally handled automatically by the Bluetooth devices. Unless one is actively designing Bluetooth chips the lower communications layers will remain hidden from most end users. Theory and background has been provided to show how Bluetooth handles communications.

### **2.1 BLU2i**

The device used on the Matrix Bluetooth board is an EZURiO BLU2i device. The BLU2i device is a self-contained Bluetooth module that can be communicated with using a set of BLU2i AT commands. This device allows users to connect to and use other Bluetooth devices at the application level. Using the BLU2i device allows the course to concentrate on the strategies and sequences of communicating between Bluetooth devices without needlessly delving into the lower technology levels.

NOTE: The BLU2i module is sometimes called BiSM.

### **2.2 AT Commands**

For this course we will be using the BLU2i AT Commands Set. This AT commands set is designed for the BLU2i module to configure and control communications. The AT commands are essentially a method of communicating with the BLU2i module and are not a part of Bluetooth as such. The AT Commands used are also part of a BLU2i command set. Whilst this means that a different Bluetooth module may use a different set of AT commands, or not even use AT commands at all, in practice most AT command based Bluetooth systems will be similar, and anyway the strategies are what matters, not the specific syntax involved.

### **2.3 Hardware and software used in the course**

The Bluetooth course makes use of the following Hardware and Software:

#### **2.3.1 E-Blocks solution**

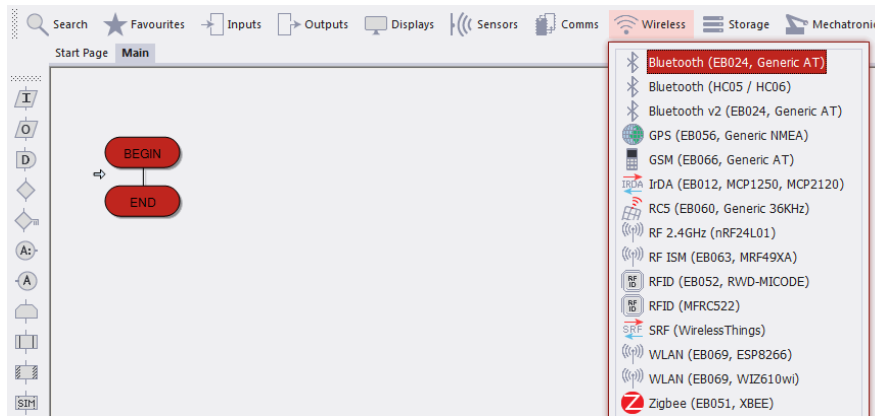
The E-Blocks solution will contain two complete kits of the following components:

- EB024 Bluetooth board
- EB006 Multiprogrammer
- EB004 LED board
- EB005 LCD Display
- EB007 Switch board
- EB032 Voice Codec board
- Headphones with attached microphone

### 2.3.2 Flowcode

The example programs used in this course require Flowcode V7 or later.

The Bluetooth component can be found in the Wireless section of the component toolbar:



All example and exercise program files supplied on the Bluetooth solution CD have been created for Flowcode V7 or later.

This course assumes a degree of familiarity with Flowcode. If necessary time should be spent using the tutorials and the Flowcode course to familiarize the student with using Flowcode.

### 2.3.3 Additional PC software (EZURiO terminal)

For basic communications tests a Communications program is required. For use with the Matrix Bluetooth boards we recommend the EZURiO terminal program which was designed for use with the BLU2i module.

Install instructions for the EZURiO terminal program can be found in the EZURiO folder on the Bluetooth solution software CD EB617

## 2.4 Additional useful devices

The following devices are not supplied with the Bluetooth solution, however if they are available they can be used in conjunction with the Bluetooth solution

### 2.4.1 Onboard Bluetooth/USB dongle

For communications to and from the PC a Bluetooth module is needed. Some recent laptops and PCs are already wired for Bluetooth. Most current PCs and Laptops though will require a Bluetooth connector such as the TDK Go Blue or EZURiO USB adapter. These plug into a spare USB port and allow the computer to function as a Bluetooth device.

Note that it is not necessary to have a Bluetooth module on your PC to complete all exercises.

### 2.4.2 Bluetooth headset

Whilst not required for the course access to a Bluetooth headset will allow testing of strategies such as Pairing and creating Headset profiles using real life Bluetooth devices.

### 2.4.3 Bluetooth enabled phone

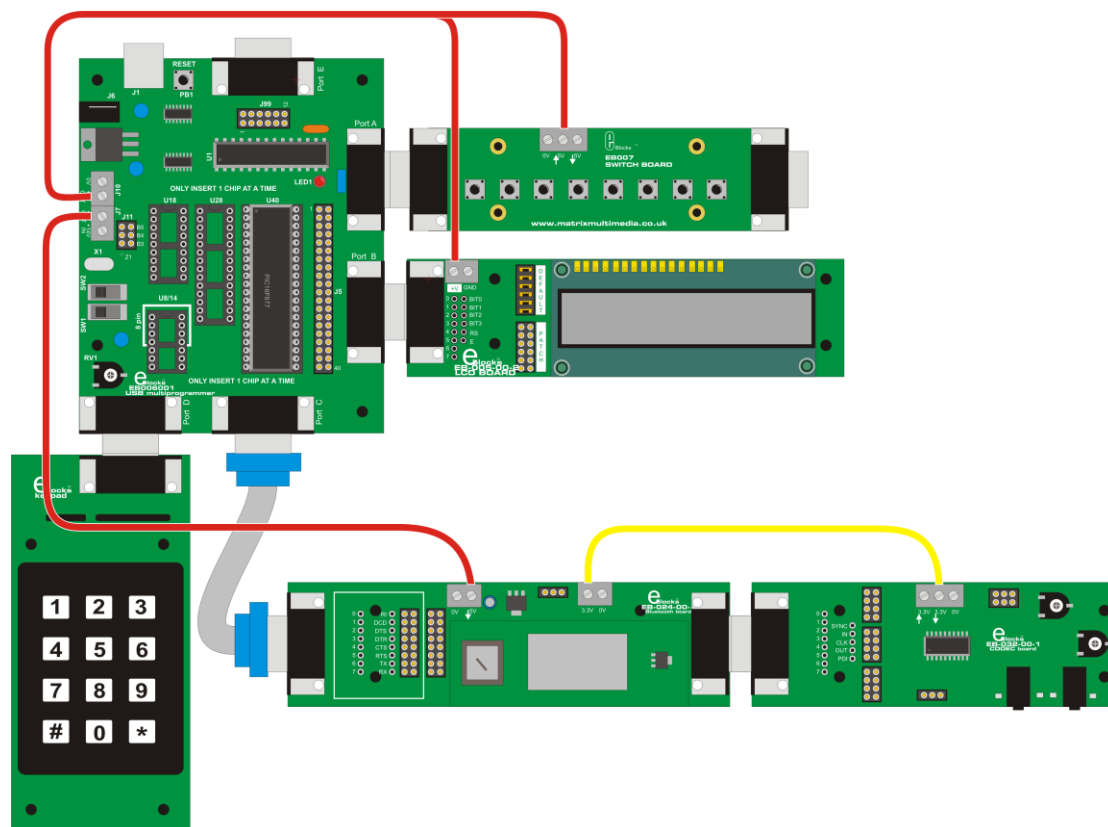
Whilst not required for the course, access to a Bluetooth enabled phone will allow testing of strategies such as Pairing and creating Headset profiles using real life Bluetooth devices.

## 3 Getting started

### 3.1 Setting up the hardware

#### 3.1.1 E-Blocks solution

The e-blocks should be connected as shown below



| PORT   | EB006 PIC Programmer                               |
|--------|--|
| PORT A | EB007 Switch Board                                 |
| PORT B | EB005 LCD Board                                    |
| PORT C | EB024 Bluetooth Board +<br>EB032 Voice Codec Board |
| PORT D | EB014 Keypad Board                                 |

### 3.1.2 Setting up the EB024 Bluetooth board

Pre-assembled Bluetooth solution kits will come with the Bluetooth boards already connected and configured correctly. Users connecting up a Bluetooth board should refer to the Bluetooth board documentation for detailed instructions on setting up and configuring the board.

| EB024 configuration | PIC               |
|---------------------|-------------------|
| Jumpers connected   | J9, J10, J11, J12 |

### 3.1.3 Adding the Voice Codec board

Bluetooth is widely used in Telecommunications making the EB032 Voice codec board a natural companion to the Bluetooth board. The Voice codec board allows audio communications to be converted to and from audio data signals that the Bluetooth board can send and receive.

Note that the Voice codec is not required for all applications, only those that use voice audio, such as phones and intercom units.

For pre-assembled Bluetooth solution kits the Voice codec boards will already be connected correctly. For user adding a Voice codec board to a Bluetooth board are advised to consult the Voice codec datasheet for connection and configuration details.

An important note to remember is that the Voice codec needs to be connected to the Bluetooth 3.3 Voltage line.

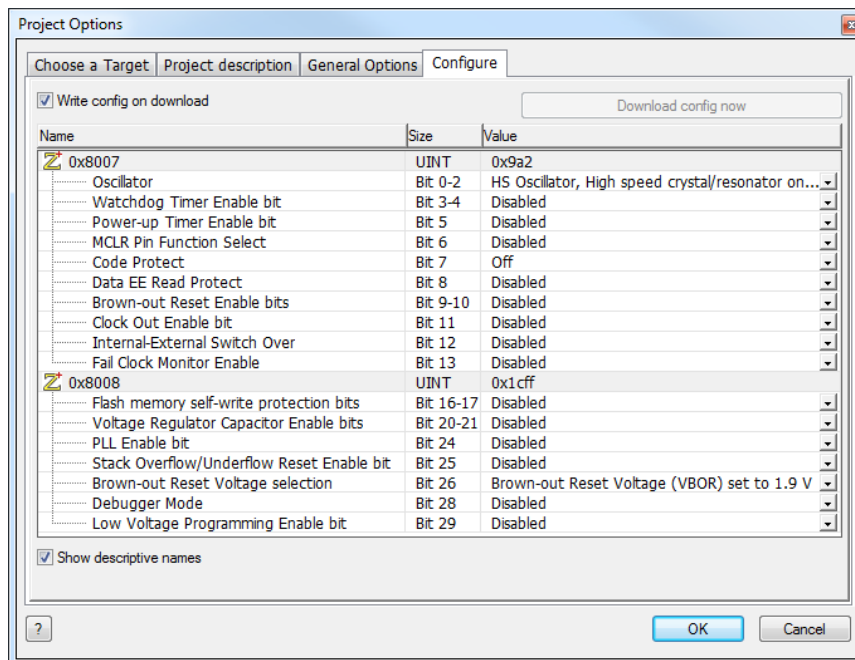
### 3.1.4 Standard Settings

The following are the standard setting used in the Projects in this course unless otherwise stated.

Ensure all E-blocks boards are wired up to power and ground if required.

#### PIC16F1937 Configuration settings:

The standard configuration used in this course is shown below. Unless specifically stated all Examples and Exercises will use this configuration. You can set the configuration options via Flowcode by using the 'Configure' tab in the 'Project Options' accessed via the 'Build' menu.



## 3.2 Introduction to Flowcode

Flowcode is a flowcharting system for microcontrollers. Generally microcontrollers are programmed in C (a hard language to learn) or Assembly language (an even harder language to program in). Teaching microcontrollers generally required a large investment of time in order to skill up students in the basics of the required languages before they were able to tackle systems such as Bluetooth.

Flowcode is a much simpler and more intuitive method of creating programs. Based on standard flowchart symbols and using drag and drop icons, a working program can be built in minutes. Icons can be configured using dialogs that remove the chance of syntax errors or invalid options being selected. Flowcharts can be visually followed, and tracked through allowing users to see the wider picture as well as the single element. Most introductory courses on programming use, or recommend creating, flowcharts as a precursor to writing your final code, due to their ability to break down the program flow in a clear and understandable manner. With Flowcode that IS writing your program.

A range of components can be used with programs that range from basic LEDs and Switches through to full communications systems such as CAN, TCP/IP and of course Bluetooth. Once again dialogs are used to remove the possibilities of syntax errors or incorrectly put together function calls. Macro icons allow the user access to component functions allowing users to perform tasks as varied as lighting a single LED to sending text to an LCD display, to checking for incoming messages on a Bluetooth system.

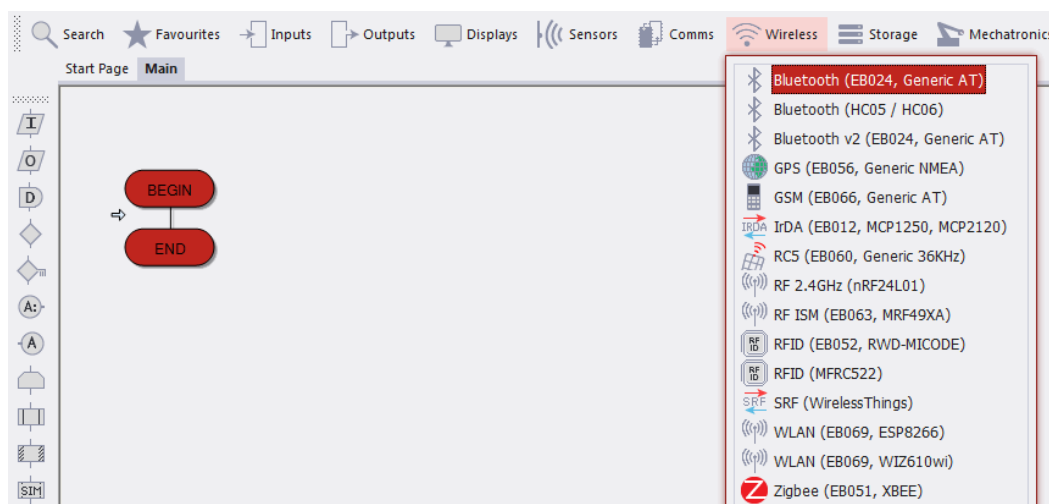
The system is designed to be easy to understand and use, but with the depth and flexibility required by today's technical, educational and industrial market places.


### 3.2.1 Starting out in Flowcode

This course assumes a degree of familiarity with Flowcode. New users to Flowcode should consult the Flowcode tutorials and introductory course prior to using Flowcode with this course. Check the Matrix TSL website [www.matrixtsl.com](http://www.matrixtsl.com) and the Matrix Learning Centre for course details and links.

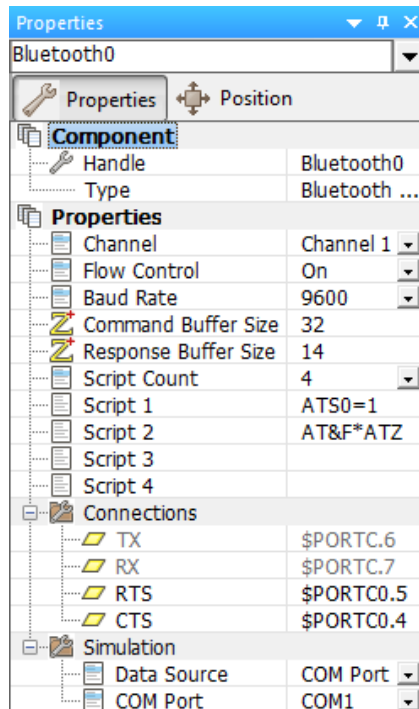
## 3.3 The Bluetooth component

The Flowcode Bluetooth component can be added to your program from the Wireless section of the Components toolbar.



 Bluetooth component icon





### 3.3.1 Properties and pin connection

The Bluetooth component has a number of properties and connections that can be set by the user.

RTS Pin, CTS Pin and Number of scripts should be self-explanatory (See the Bluetooth component Help file for further details).

Hardware On/off sets whether Hardware Flow control is used in communications. The important bit here is that the Hardware On/Off setting needs to match that of other devices it is communicating with.

The Scripts properties allow you to add up to four scripts ready to be sent. Scripts are dealt with in further detail during the course.

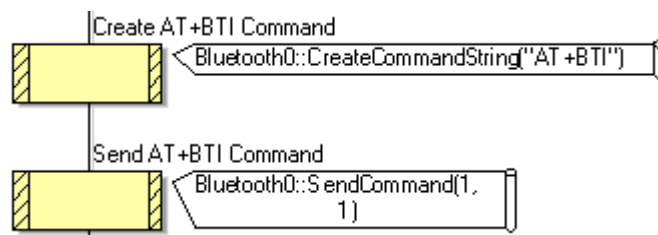
### 3.3.2 Bluetooth component macros

The Bluetooth component uses a number of macros to help program the Bluetooth device. In brief the macros are:

- Initialize  
Initialize the Bluetooth module ready for use.
- CreateCommand (Character)  
Adds the character to the Command buffer.
- CreateCommandString (Data)  
Adds the string to the command buffer
- SendCommand (ExpectEcho, SendCR)  
Sends the Command Buffer
- SendScript (idx)  
Sends Script idx
- WaitForResponse (response\_code, timeout)  
Waits for a response of type response\_code for timeout ms.
- StringReceive  
Checks to see if a response has been received.  
Returns 0 or string length
- StringRead (idx)  
Reads in char idx of the response.

Full details of the macros, their parameters and their uses can be found in the Bluetooth component help file.

The Teaching exercises section of the course will introduce and explain the use of the macros as they are needed during the exercises.





The above two icons issue the command “AT+BTI”: to issue the command you first need to create the command and then Send it.

### 3.3.3 Creating and sending AT scripts

The Flowcode Bluetooth component has a feature to aid in sending blocks of AT commands. The component has a set of four Scripts that can be sent via a single macro call. The Scripts are a feature of the Bluetooth component, not the Bluetooth specification, and have been added simply as a convenience to the user.

The Send script command Creates and Sends each of the lines in the script text box in sequence.

## 3.4 Testing the hardware

Detailed steps for a full test, including configuration and jumper settings can be found in the EB024 Bluetooth board technical datasheet. This is available on your EB671 CD ROM. More up to date versions may be available on our web site – [www.matrixtsl.com](http://www.matrixtsl.com).

Set up, configure and power up both Bluetooth solution kits.

Note the device addresses as displayed on the labels under both the Bluetooth board and under the BLU2i module board. You can read these off and make a note of them – or you can use a simple inquiry program to ascertain what the addresses of each module are. Students are tasked with making a note of the Bluetooth addresses of each of the modules in Exercise 2.

Load the program BLUETOOTH\_TEST.FCFX into both kits.

Both boards will send out an Inquiry command and will list the addresses of any Bluetooth board in the vicinity that can be found. This includes the other solution kit. Both kits should list the address of the other kit in the list of Bluetooth devices found.

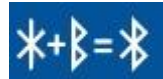
## 4 Bluetooth theory and background

### 4.1 Introduction to Bluetooth

#### 4.1.1 Brief history

“King Harald had this monument made for Gorm, his father, and Thyri his mother. That same Harald who won all of Denmark and Norway, and made the Danes Christian.” So says the Jelling Runestone. This Harald is none other than the legendary Viking king Harald Bluetooth after which Bluetooth technology is named. Bluetooth started off as nothing more than a project name, something to call the technology under development. But the name stuck. And it is a fitting name. Just as the Viking age Bluetooth brought disparate nations together, gave them a new creed and lead them on to greatness so does the modern day Bluetooth bring disparate hardware devices together, give them a new protocol and expand their capabilities. Today’s monument to Bluetooth though would not be runes carved on stone, but be the flurry of electronic signals as the world communicates via Bluetooth.

N.B. The Bluetooth logo contains the runes H and B for Harald Bluetooth:



Bluetooth was introduced in 1998 by the Bluetooth Special Interest Group, a federation of companies involved in communications, industry and business technologies. The variety and number of companies involved has helped to make Bluetooth successful and widely adopted by a number of different systems.

The original Bluetooth specification version 1.1 (1.0 and 1.0B being too restricted technology wise for major development) was immediately successful. The specification was updated to version 1.2 in 2003 with added features such as higher speeds and adaptive frequency hopping. In 2004 the version 2.0 + Enhanced Data rate (EDR) specification was released. V2.0+EDR is faster, uses less power and allows better communications between multiple devices. Bluetooth continues to grow and adapt to the rapidly changing telecommunications need of the modern marketplace.

#### 4.1.2 Bluetooth concepts

- Bluetooth is designed to enable secure wireless data transfer between hardware devices.
- By the use of a protocol data can be passed between the devices in a standard manner. This simplifies coding and testing as propriety code does not need creating.
- Bluetooth is fully specified which allows manufacturers to create Bluetooth compatible hardware without requiring them to design and implement propriety high level protocols. This also allows for network simplification as there are no problems with competing propriety protocols or specifications.
- Communication is automatic once two Bluetooth devices are within range. This means users do not need to run through set up procedures to start communicating.
- Bluetooth devices can be made secure and non-discoverable helping to eliminate security threats from automatic communication. You get to decide if you want to be on the network or not.

Bluetooth is essentially a wireless communications network with the ability to automatically discover and communicate with new devices that come within range. This allows two key benefits to the user.

### 4.1.3 Bluetooth advantages

- 1) Cables are not required.
  - No spaghetti cabling running all over the place. No unsightly messes. Nothing to trip over or damage. And for uses such as automotive technology no wiring looms to install and maintain/repair/replace.
  - No physical constraints due to positioning of cables or network points, therefore greater freedom of movement for devices.
  - Communication is distance based so devices can be mobile.
  - Devices can move from network to network without requiring infrastructure
- 2) Automatic discovery of other devices within range.
  - No need for physical changes, such as new network points, to be made to accommodate new devices. Therefore ease of use and ease of upgrading.
  - Bluetooth security features allow you to set the device to respond to or ignore other devices putting you in charge not the network.
  - New devices can be added simply by coming within range so no need to edit or change settings or anything to add new devices.

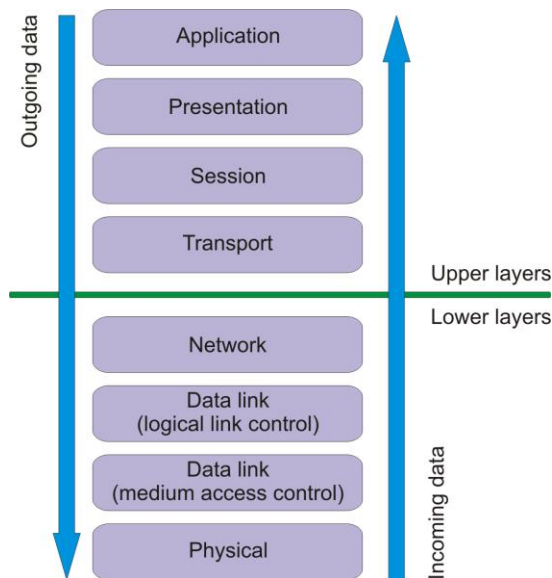
### 4.1.4 Bluetooth disadvantages

- 1) Complexity and reliability
  - Cables may not be portable but they are simple and reliable. Complexity, as in Bluetooth systems, brings with it increased risk of failure.
  - The more complex a system is, like Bluetooth, the harder it can be to find and fix faults.
- 2) Battery life
  - Mobile devices are limited by battery life and recharging facilities.
  - Loss of power during communications can occur.
  - Cables may be standardized, but plug sockets and voltage levels are not. When traveling a power socket adapter may be required.
- 3) Costs
  - Bluetooth devices are often more expensive than conventional fixed solutions such as a cable.
  - An initial outlay may be required to convert from legacy equipment to Bluetooth solutions.
  - Higher repair costs.
- 4) Speed
  - Bluetooth is slower than equivalent fixed solutions such as a direct cable.
  - Bluetooth communications speed may be restricted by either baud rate considerations, or by limitations of the data transfer speeds of the Bluetooth protocol.

## 4.2 Protocols and the OSI model

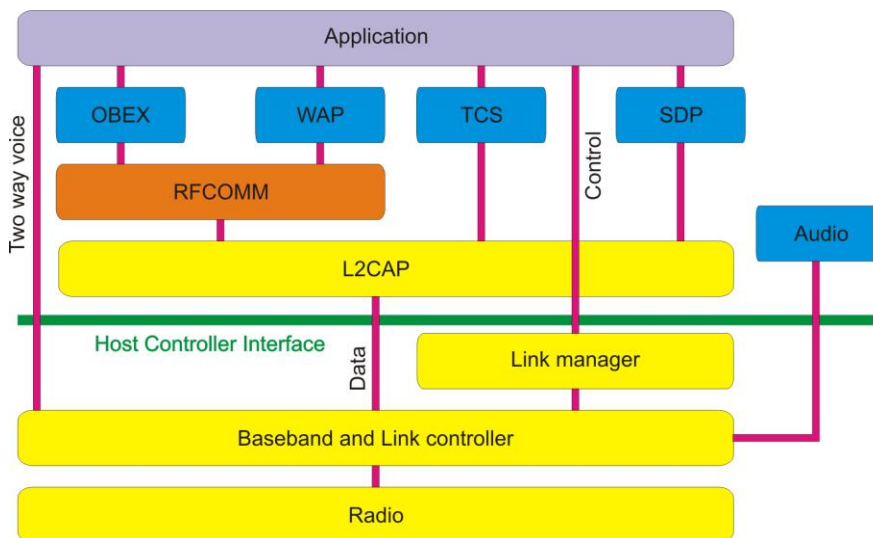
Bluetooth works in layers based on the OSI model. The OSI model is a sequence of layers used to define how data moves from the application to the physical sending of signals. This process is repeated in reverse upon reception of data to change from the signals back to application data. Each layer addresses fundamental areas of the communications processes. By abstracting out the different parts of the communications process into OSI model based layers, developers can implement their systems as a series of tasks each covering one or more OSI layers. Standards can be implemented for specific tasks or layers creating a base for developers to work on.

The basic OSI model diagram can be seen below:



The OSI model is used to aid in separating out the various elements

The Bluetooth layers can be seen below:



The basic process is simple: raw data at the top needs to be converted into bits and bytes that can be transmitted via radio signal and built back up into the raw data the application at the other end can work with.

Transmitting involves breaking the data into chunks that can be sent and wrapping the chunks of data with extra bits of information that can be used by that particular layer to check for things such as errors or missing data. A single piece of data from an application may get broken down into a number of different chunks each of which is surrounded by several different wrappers.

Receiving involves a pass-the-parcel process with each layer unwrapping the top covering and passing the packets up to the layer specified in that wrapper. At each step other functions can occur such as error checking and assembling groups of packets into a larger packet that will be passed on up once complete. Finally the packet will reach the application and the data can be displayed to the user (or put through the speaker if audio etc.).

Not all layers need to be used. Some packets can skip several layers. The process depends on the type of data being sent.

#### 4.2.1 Application

The application is the element that is creating or using the data; be that raw data, audio signals or textual information. This is the level that we users communicate with the system. If we used the Post office as an analogy for Bluetooth the application would be the words that we write or read.

Data input by us in whatever form, spoken or typed etc. is converted into a form suitable for transmission, be that ASCII data, or audio format data. The data can then be sent down the pipeline to be transmitted.

When data is received it is in a format that the application can understand. All the collating of data packets and re-assembly has been done. This is just raw data ready to be used.

#### 4.2.2 Bluetooth layers

##### OBEX/WAP/TCS/SDP/AUDIO

These are protocols for data. Rather than use custom data models it is good practice to make use of existing protocols and standards. This allows you to communicate with other applications via the same standard protocols. Data is converted to the protocol format ready to be passed on down, or is converted from the protocol to the raw data used by the application. By using such common protocols programming tasks are simplified and code is made easier to transfer to other devices.

Some protocols, such as SDP and TCS, can be used to help configure Bluetooth operations. SDP for instance allows applications to query the Bluetooth device as to what services it can provide.

The protocols commonly used include:

OBEX – **O**bject **E**xchange

WAP – **W**ireless **A**ccess **P**rotocol

TCS – **T**elephony **C**ontrol **S**pecification

SDP – **S**ervice **D**iscovery **P**rotocol

AUDIO – direct audio signal as used for headphones

##### RFCOMM

RFCOMM, **R**adio **F**requency **C**OMMunication, is a special protocol. RFCOMM enables users to use a standard serial COM port connection rather than a Bluetooth radio connection. These are virtual COM ports, not physical COM ports. RFCOMM tricks the PC into believing its virtual COM ports do exist in the same way as the real hardware ones. But when it receives data it quietly sends it to the Bluetooth radio link. The end application is none the wiser; to it it's just a serial port it sent to. When it receives data it is just data from a serial COM port, the Bluetooth side is totally hidden from it.

The great advantage of this is that the myriad of existing serial COM port technologies can be used directly with no need of special adapters or brand new software. This can have considerable effect on the cost and effort involved in upgrading or developing products that currently use Serial COM port technology. The best Post Office analogy for RFCOMM is email. It's a letter-style communication but is not a letter. Emails are in the same format as a letter, we write and receive them, but the process of sending them is not handled by the Post Office. Similarly the process of sending data in RFCOMM serial communications is not really serial communications even if it appears to be to the host controller or PC.

##### RFCOMM and the BLU2i module

The BLU2i module sits between the RFCOMM layer and the application layer. The module handles the actual RFCOMM side of things, but requires the AT Commands as outlined in the AT Command set to be sent to it so it not quite at the application level. This allows

programmers to concentrate on the communications strategy whilst leaving the RFCOMM specifics to the BLU2i module.

**L2CAP**

The **Logical Link Controller and Adaptation Protocol** controls and coordinates data flow creating virtual channels, sessions and file transfers. L2CAP is also the main packet assembly areas breaking down the data into the individual packets ready to be sent to the Baseband layer for preparation for transmission. On the return trip the L2CAP is the primary reassembly area collating all the individual packets and reassembling the data from them. In our Bluetooth Post Office this layer would be the pages of the letter, and the envelope it comes in.

**Host Controller interface**

The Host Controller Interface links the PC or system to the Bluetooth hardware. Elements of the HCI include device drivers required to run the Bluetooth hardware and its interface. Where the Bluetooth hardware meets the data handling hardware is where the HCI resides.

**Link manager**

Link manager is a low level language for configuring and controlling the links.

Given the sophistication of Bluetooth transmissions with adaptive frequency hopping and a wide range of available frequencies, the Link manager is a very useful tool to have. Link manager aids at the network and data link levels. Link manager is like the Zip code or Post code that the Post Office uses to automate much of its sorting.

**Baseband and Link controller**

The Baseband and Link controller, corresponding to the Data link level of the OSI model, is a packet controller. It assembles data sent down to it into data packets ready for transmission by the radio layer. The contents of the data and who sent it are irrelevant. All that matters is that the Baseband layer can assemble it into data packets for transmission.

Data passed up to it from the Radio layer has its packet tags examined and is then shunted upwards to the appropriate layer. The Baseband layer does not care what the data contains, just who to deliver it too. In our Post Office analogy this layer is the primary sorting office and the address on the letter.

**Radio**

The Radio layer is the Physical layer, the lowest level available in Bluetooth. At this level the only real concern is with the actual radio signal. A radio connection is made and the data sent across. The data packets assembled by the Baseband are sent out, and incoming data packets received and passed on up to the Baseband layer.

**4.2.3 Hardware details**

Bluetooth hardware uses a number of common hardware and software features in its communications that are useful to know.

**Piconets**

A Bluetooth device can be part of a network of up to 8 Bluetooth devices. The network is formed of one device selected to be the Master device and up to seven other Slave devices. This mini network is called a Piconet. Slaves can only send to the Master device not the other Slaves. The Master communicates to the Slaves in a 'Round robin' system talking to each Slave device in turn. So signals can be passed to another Slave device via the Master device, but not directly.

Forthcoming Bluetooth devices will have the ability to expand this by communicating between Piconets. One device will be the Master in one ring and a Slave in another allowing communication between the two rings.

**Paging and inquiry**

All Bluetooth devices have the ability to become Master or Slave devices. The Master device in a Piconet is the one that initiated contact, the Slaves those that responded. Contact is



initiated by sending out Inquiry signals to determine what devices are nearby. At the same time devices are listening for contact from another device. If a device is found it can be paged to establish contact and a Piconet either established, or joined if one already exists. If a Master exists as is the case when joining a Piconet then the new device will be a Slave. If no Piconet is already in existence then the device that initiated contact becomes the Master device. Which one it is depends on who received the inquiry signal. As devices can join or leave the network at will mechanisms exist to replace a master device that leaves the Piconet.

### Radio specification

Bluetooth operates in the 2.45GHz ISM radio band. The band is split into 79 different channels which Bluetooth hops between at 1600 times a second (Specification 1.1 and 1.2). Later Specification 2 Bluetooth devices can be up to 3 times faster. Bluetooth hops frequencies to prevent both interference from or to other systems on that same frequency (e.g. a baby monitor), and to prevent Bluetooth devices hogging one particular channel and thus jamming it for other devices. Obviously the system needs to know which channel to change to. This is done at the radio level with the transmitter and receiver automatically keeping pace with the frequency hops of the other devices in the network. Signals sent via the radio level contain embedded data about the frequency hop pattern the device is using that the other devices can use to work out which to go to next. Fortunately it is all done automatically far down in the layers and so is not something you need to configure or program for with the Bluetooth E-blocks board.

Due to the low power of the radio signal (about 1 milliwatt) the range is limited to about 30 feet, but this is more than enough for most Bluetooth devices.

### Audio transfer

Due to its telecommunications roots Audio signals are a major part of many Bluetooth devices.

In order to simplify Audio transfer, Bluetooth can fast track digitized audio signals through the system to the audio output on the receiving device. Audio data can be sent via the normal methods, and applications such as a Bluetooth MP3 player would generally use the normal Application to Application route as it has better data integrity, and buffering data at the receiving end will handle most latency issues. For direct communication such as a telephone though the direct route is preferable as data quality – i.e. noise, is less of an issue than latency – i.e. waiting to hear what was said as the data takes time to work through the system. Signals are fed down to the lower layers directly, processed and compressed at the Baseband level and sent to the receiver which sends the data direct from its Baseband level to the audio output device.

### Authentication and encryption

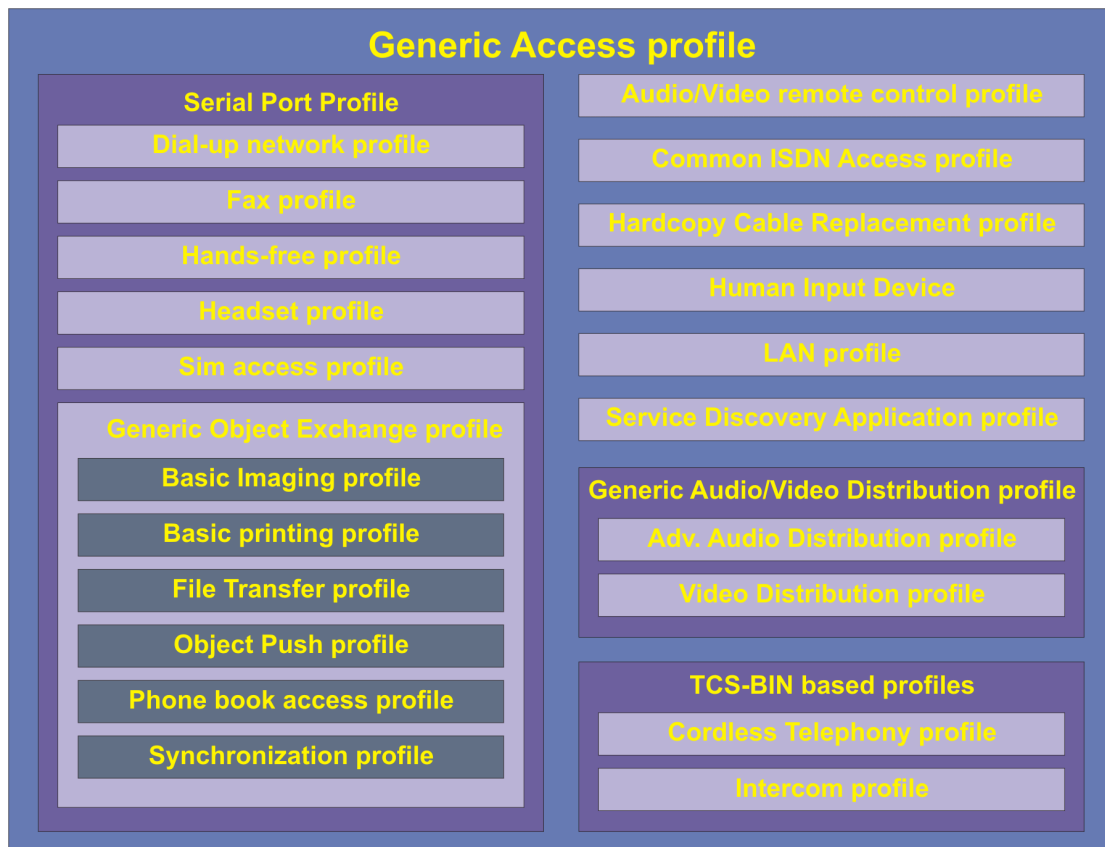
Bluetooth is a radio device. This makes it inherently unsecure as any device within transmission range can listen in to the signal. Therefore security is a prime concern for Bluetooth.

The primary security level is pairing. Devices need to be paired up which requires the address and pass key for the device being paired with. Access to the address can be blocked by devices, thus making pairing impossible, and the Pass key is not accessible by communications so must be known for the device to pair. In practice this will be a PIN number supplied with the product. So to pair a phone and a headset the phone you will need to enter the headsets PIN number into the phone when prompted. No PIN – No communication.

Once paired communications can begin. Bluetooth devices can be set to both encrypt and authenticate communications, thus making sure that data is from an approved device, and is encoded making listening in fruitless. Any code can be cracked, but the Bluetooth encryption and close range needed for listening in make it a very secure system.



## 4.2.4 Profiles



Profiles are code based implementations for use with standard hardware structures such as headsets or mobile phones. Conforming to a profile allows the hardware to make use of features and behaviors appropriate for that type of device. For instance using the Headset profile allows other units to communicate to the Bluetooth device using standard calls that are part of the Headset implementation. The Headset profile will contain a number of functions that are also in a Telephone implementation, but many of the Telephone functions will not be in the Headset profile. This is because the Headset profile is a subset of the Telephone profile.

The layering of profiles allows higher level profiles to use the same features as lower level ones. The Fax profile for instance can use the File Transfer features that are part of the File Transfer profile. It can also use the Synchronization profile features as well as this too is also a sub set of the Generic Object Exchange profile that is a sub set of the Serial Port profile that the FAX profile resides in.

## Part 2: Teaching exercises

### 5 Discovery

#### 5.1 Theory: Finding other Bluetooth devices

To be able a Bluetooth device to communicate using Bluetooth there needs to be another device to communicate to. With Bluetooth this can be any device within communications range. The first step to establishing communications is to find that other device. The BLU2i module has an inquiry command, AT+BTI that can be used to inquire about other devices in range. The AT+BTI command sends a signal that is picked up by any Bluetooth device within range. These Bluetooth devices can then elect to respond to this inquiry to let the inquiring device know that they are out there. Devices respond to an inquiry by sending their device address that can then be used later on to contact the receiving Bluetooth system.

##### 5.1.1 The Inquiry command

The Inquiry command is `AT+BTI<devclass> {Inquire}`.

Full details of the command can be found in section 2.2.34 of the BLU2i AT Command set document. This is included on the EB617 CD ROM. Whenever a new AT Command is introduced refer to the AT Commands set document to gain an understanding of the new command. Building up knowledge of the BLU2i AT Commands set is an important body of knowledge. The AT+BTI command causes the Bluetooth module to transmit an Inquiry signal to which other Bluetooth devices can respond. The optional <devclass> parameter is used to filter which devices to check for and will be dealt with later. For now the base AT+BTI command can be used. The AT+BTI command uses two S Registers to set the delay and maximum number of responses. These can be left at their default values for the exercises and examples in this course.

The AT+BTI command can receive the following responses:

- 123456789012 – The 12 digit Hexadecimal address of the Bluetooth device answering the inquiry. Note that not all devices within range may answer. Bluetooth devices can be set to ignore the Inquiry command. This will be covered next chapter.
- OK – Received when the Inquiry is complete.
- ERROR 14 – An error has occurred. The Bluetooth module is not configured correctly for sending the Inquiry command. (See the AT Command Set document for details).

A number of responses can be received: one address response from each device answering the inquiry, and an OK message once all have been received. Checking can be done and if it is not an OK response then the address can be displayed.

##### 5.1.2 Additional Inquiry commands and the <devclass> parameter

In addition to AT+BTI there are a number of other related commands such as AT+BTIV and AT+BTIN.

AT+BTIV <devclass> functions the same as AT+BTI except that the 6 digit <devclass> of the responding device is added to the response message.

For example: 123456789012,123456

AT+BTIN <devclass> functions the same as AT+BTI except that the 6 digit <devclass> of the responding device, and the name of the device is added to the response message.

For example: 123456789012,123456,"EZURIO blu2i RS232"

The Inquiry commands take an option <devclass> parameter that can be used to filter responses. The <devclass> parameter is a 2 or 6 digit hexadecimal character value. To check for device of a specific type use the full 6 character <devclass> for that device type. For

instance to check for all headsets within range you would use the <devclass> value 200404 i.e. AT+BTI200404. The 2 character major class code can be used to check for types of devices such as Audio devices or PC peripherals. For example 20 will search for Audio devices such as headsets phones and music players.

### 5.1.3 Discovery example

There are 4 Bluetooth devices within range, named Blue1-Blue4. The devices are set up as follows:

| Device | Address      | Configuration                    |
|--------|--------------|----------------------------------|
| Blue1  | 00809872F3D4 | Accepts Inquiry commands         |
| Blue2  | 00809864DD44 | Accepts Inquiry commands         |
| Blue3  | 00809894E620 | Does not accept Inquiry commands |
| Blue4  | 00809894E5D5 | Accepts Inquiry commands         |

Device Blue1 transmits the AT+BTI inquiry command.

The following responses are received:

- 00809864DD44
- 00809894E5D5
- OK

## 5.2 Exercise 1: Discovering Bluetooth devices

### 5.2.1 Introduction

To be able to communicate to a Bluetooth device it is necessary to know that the device is there. The first step in Bluetooth communications is therefore to inquire as to what devices are present, and to be able to identify them so that communications can be established with them.

### 5.2.2 Objectives

- Develop a program that performs a Bluetooth device Inquiry and displays the addresses of any devices found on the LCD display.
- Display a 'Finished' message once all the responses have been received.

### 5.2.3 Pre-requisites

- An understanding of standard Flowcode Components and icons, such as Decision icons and the LCD component and macros.
- An understanding of sending Bluetooth commands (See the Bluetooth Component Help file in Flowcode for details on the macros involved)
- An understanding of receiving and retrieving message data (See the Bluetooth Component Help file for details on the macros involved, and an outline of the relevant strategies)

### 5.2.4 Hardware/Software requirements

The following items of hardware are required:

- Bluetooth solution for the Exercise program.
- 1 or more additional Bluetooth devices for demonstrating the program.  
Note: The second Bluetooth board from the Bluetooth solution can be used for this Exercise.  
The program BT\_EX1\_NODE\_A.FCFX can be used to set up the board.
- Set hardware jumpers as specified in the Getting starting section.
- Set up Flowcode to configure the microcontroller as specified in the Getting starting section.

### 5.2.5 Exercise information

The Inquiry command is AT\_BTI. See the EZURiO AT Command set document (supplied on CD ROM) for details on the AT+BTI command.

The expected responses are:

- A 12 digit device hexadecimal address from devices that respond.
- An ERROR message.
- OK once the Inquiry is complete.

The objectives can be broken down into the following:

Tasks

- Send the Inquiry command.
- Check for responses.
- Display the device address of any device that responds.
- Check for OK response.
- Display 'Finished' once the inquiry is complete.

Control structures:

- If the Inquiry is complete display a 'Finished' message.  
Otherwise continue checking for further responses.

### 5.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- Understanding basic AT commands
- Understanding the discovery process
- Creating and sending commands
- Checking for and retrieving responses
- Checking for specific responses (e.g. OK)

#### **5.2.7 Additional tasks**

The following are additional tasks that can be implemented to add extra features or improved functionality to the basic program:

- Add a COUNT for how many addresses were found.
- Perform basic error checking on the CreateCommandString and SendCommand macro.
- Check for an ERROR 14 response.
- Retrieve and display the device name(s) using AT+BTIM.

## 5.3 Practical implementation: Discovering Bluetooth devices

### 5.3.1 Discovering and discoverable

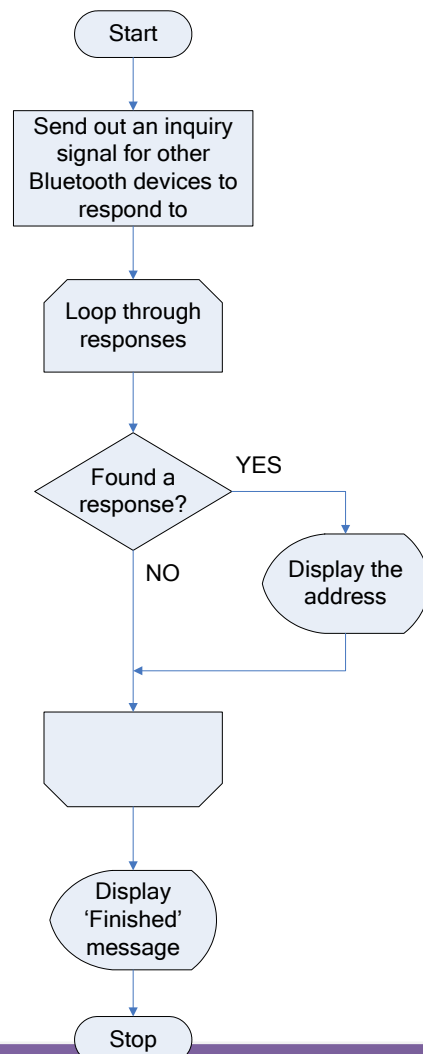
Note that the programs created for Exercise 1 needs to be used in conjunction with the program created in Exercise 2 to form a discoverable and discovering pair of boards. Using the two exercises as a linked pair allows students to explore discovering and discoverability using the same program from Exercise 1 in both Exercises, giving them continuity and the ability to see the earlier Exercise in action.

The two programs from Exercise 1 and Exercise 2 will form the core part of most of the subsequent exercises, additional commands being added in as the course progresses. As such a continuity approach can be taken where the final programs from previous Exercises can often be used as the starting point for the next. Using this approach allows students to see how the process flows and grows as they progress through the exercises.

The program students need to write in Exercise 1 is to discover any other Bluetooth devices in range. This necessitates a program in the corresponding Bluetooth device being active and that Bluetooth device being discoverable. Before starting, students should download the program BLUETOOTH\_TEST.fcfx into both nodes. This will ensure that both nodes are discoverable.

### 5.3.2 Planning the program

Before writing a program there needs to be an idea of what the program objectives are, and an outline of how this will be implemented. Below is a rudimentary plane for an Inquiry program.



The plan is to send out the inquiry signal, then to loop through checking for any responses, displaying the address of any device that responds, and a 'Finished' message once the inquiry is complete.

Hardware requirements to achieve the plan include a Bluetooth module (obviously), and a display device that is capable of displaying the entire 12 digit address of the responding devices. Here we will be using the EB005 LCD display.

Software requirements include:

- Sending an Inquiry signal
- Checking for responses
- Obtaining the response data
- Displaying the data.

### 5.3.3 Required macros

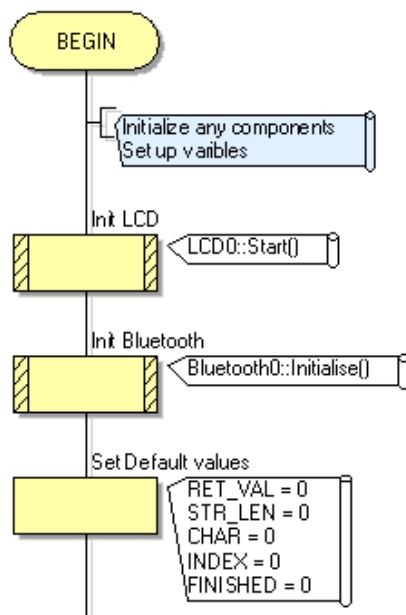
Assuming the student has a basic familiarity with Flowcode and basic components such as the LCD display then only Bluetooth specific macros need to be covered.

The following Bluetooth component macros will be needed for the program:

- Initialize
- CreateCommand
- CreateCommandString
- SendCommand
- StringReceive
- StringRead

The macros break down into a Send command set – CreateCommand / CreateCommandString and SendCommand, and a Receive response set – StringReceive and StringRead.

### 5.3.4 Initializing



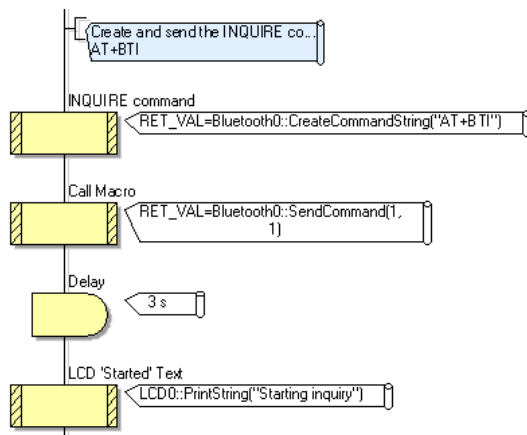
The basic program flow is to start off by initializing any components, and setting up any default variable values. Add in an LCD Start macro, and a Bluetooth Initialize macro.

The Bluetooth Initialize macro is required for all programs that use the Bluetooth component, and needs to be added to all Bluetooth programs. The best place to add initialization macros is right at the start of the program.

Add a calculation icon and enter default variable values in here. This can be updated whilst constructing the program to add in any new variables that are needed. Setting default values is good practice and can help avoid difficult to trap errors due to erroneous values.

Once all the components and variables are set up the first task can be considered, that of sending the Inquiry command.

### 5.3.5 Sending a command



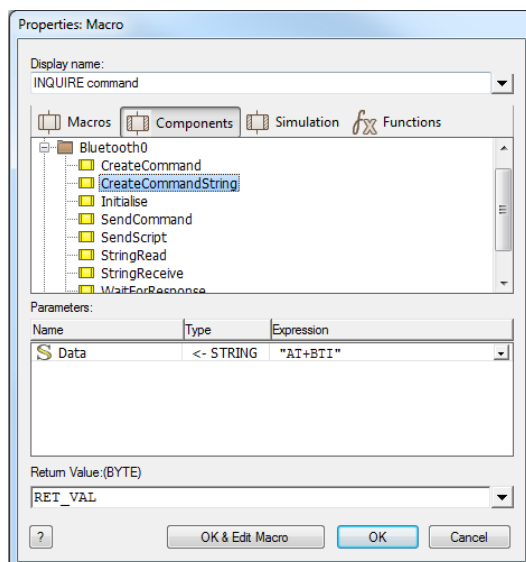
Before a command can be sent it needs to be created.

The CreateCommandString macro takes the *Data* parameter. Characters entered into the CreateCommandString macro are added to the Command buffer. Commands can be built using a single CreateCommandString macro, or by using several to build the Command up.

The Return value is 1 for success, or 0 for failure to create the command. The Command buffer is 32 characters in length, sufficient for the needs of the BLU2i command set. If the Command buffer is full CreateCommandString will return 0 for a failure to create the command.

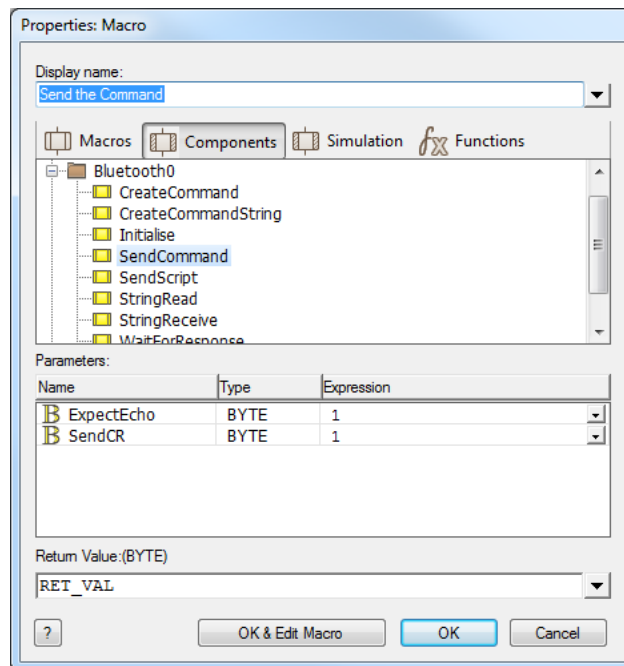
In this case the inquiry command AT+BTI needs to be sent. This can be built using a single CreateCommandString macro. Add a CreateCommandString macro to the flowchart.

The Data parameter is set to "AT+BTI" and a RET\_VAL variable is used for the return value. The return value will not be checked in this example to avoid making this first program over complex. However in general it is good practice to use the return value for basic error checking to ensure commands are created correctly.





Once the Inquiry command is created it can then be sent using the SendCommand macro. Add a SendCommand macro to the flowchart.



The SendCommand macro sends the command that is currently in the CreateCommand buffer. The return value is 1 for a successful send, or 0 for failure to send the command. Once again this particular example will not check the return value in order to simplify the program.

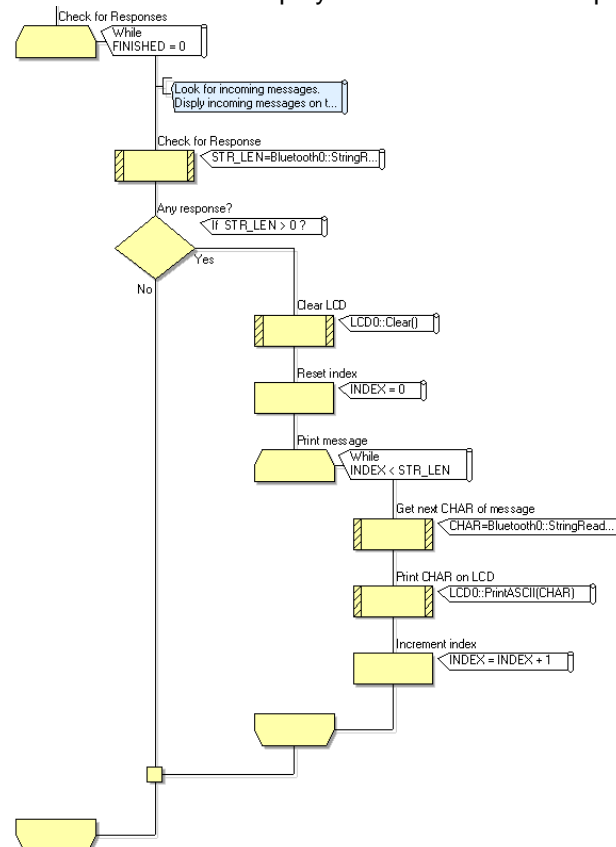
The parameter ExpectEcho is set to 1 as the Bluetooth module generally echoes sent data back to the transmitting module. This will generally be set to one when sending commands in this course. Exceptions will be noted where they occur.

The parameter SendCR is set to 1 to send a carriage return with the command. This will generally be set to one when sending commands in this course. Exceptions will be noted where they occur.

Finally add an LCD PrintASCII macro to display a message, such as "Starting inquiry", to say that the Inquiry command has been sent.

### 5.3.6 Checking for responses

Once the inquiry has been sent the next step is to listen for responses. There can be a number of responses so a loop structure is needed to go through them all. If an OK response is received then the inquiry is finished and the loop can be ended.



The basic flow is a Loop icon set to loop until a variable, FINISHED, is > 0.

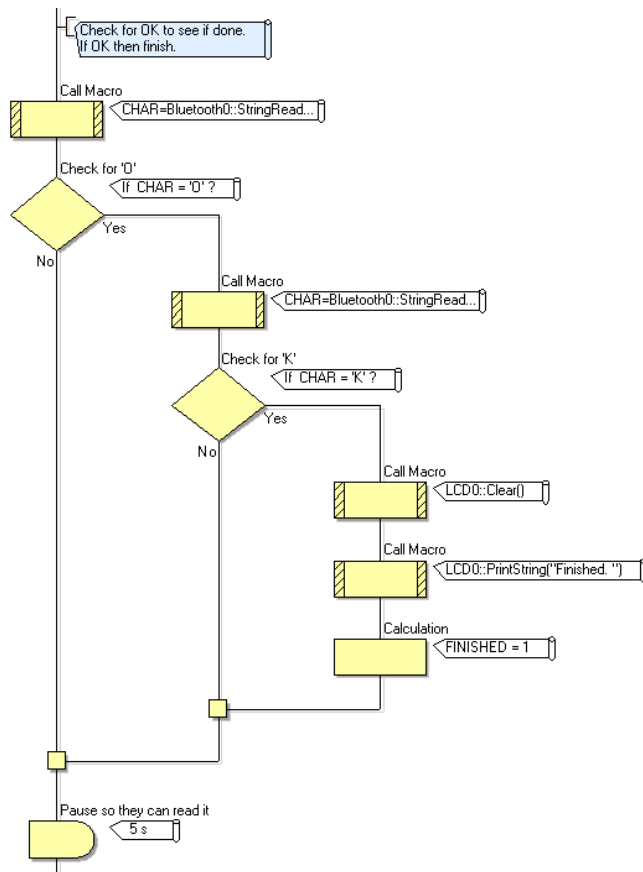
To check for a Response a StringReceive macro is used. The StringReceive macro checks to see if a message has been received and returns the length of the message string or 0 if no message string has been received. The return value variable, such as STR\_LEN, can be tested to see if it is greater than 0. If it is a message has been received and can be displayed.

Clear the LCD ready for the message. The STR\_LEN variable has the string length so we can display the message stepping through for STR\_LEN number of times and printing the character. Create an INDEX variable set to 0 and loop through incrementing INDEX while INDEX is less than STR\_LEN – the size of the message.

The StringRead macro returns the ASCII value of the character idx of the message. Using INDEX for the idx parameter will return the message characters which can then be displayed using the PrintASCII LCD macro.

A delay is required so that the response can be read.

Later programs will require us to use or interact with the addresses gathered here, but for this example all we need to do is display them. A delay of 5 seconds should be sufficient to demonstrate the addresses being received.



Responses need to be checked to see if they are an “OK” response, at which point the Inquiry is done and we can finish the loop by setting FINISHED to 1.

An OK command will have an ‘O’ and a ‘K’ as the first and second characters of the response. Retrieve the first character – idx 0 using the StringRead command. If it is an ‘O’ then check the second character – idx 1 of the message to see if it is the ‘K’. If so set FINISHED to 1 to end the loop.

To round off the program a ‘Finished’ message of some sort is required to inform us that the Inquiry has completed.

### 5.3.7 Running the complete program

The complete program can be found in example BT\_EX1\_NODE\_B.FCFX.

For testing a Bluetooth solution board is required. See the Getting started section for details on setting up the board and configuring the microcontroller.

In addition at least one other Bluetooth device needs to be active and discoverable. This can be the other Bluetooth solution board, or a third party Bluetooth device. Be warned though that not all Bluetooth devices will respond. When demonstrating the program it may be prudent to test the Bluetooth devices prior to using them for the demonstration to check if they can be discovered or not.

For example, if you have a Bluetooth-enabled mobile phone you will need to turn its Bluetooth on and set the phone's visibility so that it is not hidden.

The Bluetooth test file as detailed in the Getting started section is configured to respond to Inquiries and can be used to set up the second Bluetooth board.

Compile and download the program to the microcontroller on one of the Bluetooth boards. Briefly unplug the power, plug the power back in and press the Reset switch on the Programmer board. This ensures that the Bluetooth module resets and restarts correctly.

The LCD will display "Starting Inquiry". Next the LCD will display the 12 digit address of any devices found, with a 5 second pause between each one. Finally a "Finished" once an OK response is received indicating that the Inquiry is complete.

## 6 Discoverability

### 6.1 Theory: Discoverability

The inquiry command covered in the previous chapter is sent to all Bluetooth devices. However, not all devices will respond to the Inquiry signal. This is because Bluetooth devices can be set to hide themselves from other devices. Devices can also set themselves to be discoverable, in which case they will respond to the Inquiry command.

#### 6.1.1 Configuring for start up

Bluetooth devices have a range of configuration options that cover everything from Passkeys and encryption to baud rate and the number of rings before answering. Defaults are generally used so that devices will normally work 'out of the box'. However it is good practice to configure devices with the correct settings upon start up to ensure the device is operating as desired.

In the BLU2i module these options can be configured using S registers, or via other AT commands.

#### 6.1.2 Using S registers

The EZURiO BLU2i module uses a set of onboard registers to store operational parameters such as number of rings before answering, whether to echo a message, whether the device is connectable and discoverable etc. The registers used to store the parameters are called S Registers.

The S Registers are accessed using the AT command `ATS<register>=<value>`. For example `ATS512=3` will set the set S Register 512 to the value 3. Looking up S Register 512 in the AT Command Set document, along with a parameter value of 3, informs us that the device will be configured to be connectable but not discoverable. To make the device discoverable only we can check the parameter values for S Register 512 and see that a value of 2 should be suitable.

There are a lot of S registers, details of which can be found in the AT Command Set document.

The major S registers are listed in the Appendix under `ATS<register>` in the AT commands section. There are a number of invaluable S Registers that will be used in most programs. Taking the time to learn these S Registers and their values will aid in not only correctly configuring devices, but in understanding how Bluetooth operates.

#### 6.1.3 Using AT commands

There are a set of AT commands that configure the system in a specific mode. For instance `AT+BTO` opens the device and makes it discoverable. These AT commands can be used to quickly configure a device.

Examples of AT command configurations are:

| Command | Description                                       |
|---------|---|
| AT+BTG  | Makes the device Connectable but not Discoverable |
| AT+BTO  | Makes the device Discoverable                     |
| AT+BTP  | Device is Connectable and Discoverable            |
| AT+BTQ  | Responds to enquiries only.                       |

#### 6.1.4 S Registers or AT Commands?

Whilst both approaches are viable this course will primarily use S Registers for configuring the BLU2i module. Using S registers allows the user flexibility in that they can be set individually unlike the general mode configurations that single AT command configurations use. Users also have the ability to examine and look up the individual S Register settings to understand

what exactly is being configured - making them more transparent than the single AT commands.

### 6.1.5 Making a device discoverable

For this exercise we are interested in making the device answer automatically and to be discoverable and connectable, and for us to be able to send data to it. We also need to set this configuration on the device so that we can turn it on and off without needing to reconfigure every time.

Breaking the required configuration details down the following list is obtained:

### 6.1.6 Baud rate and settings

There are a number of S Registers that deal with the Baud rate and other hardware connection details. However, these should be left at their default values for the exercises in this course.

### 6.1.7 Discoverable and Connectable

- Discoverable sets whether the device will respond to an Inquiry signal
- Connectable sets whether the device can be connected to.

Both are set by the same S Register so need to be considered together. The S Register 512 takes a value 0-7 that specifies the power up state of the device. There are a number of complex states, however for now we can simplify it down to 3 states that could be useful for us.

- 2: set the device to be discoverable only.  
Useful when a device needs to be the one that initiates contact, not receive it. Such as a paging system.
- 3: set the device to be connectable but not discoverable.  
This state allows a device to hide from random Inquiries, but to be connectable by devices that already know its connection details (i.e. its Bluetooth address). Useful for secure communications.
- 4: set the device to be both discoverable and connectable.  
As the plan is to connect to the Bluetooth device in subsequent exercises an S Register value of 4 will be used for most exercises in this course.

### 6.1.8 Number of rings before answering

Just like an answering machine waits for a number of rings before cutting in automatically, so does Bluetooth. S Register 0 specifies the number of rings before a call is answered. Setting S Register 0 to a value of 1 configures the device so that the call is answered on the first ring – i.e. immediately.

### 6.1.9 Whether to allow remote command for data sending

One important configuration item is whether the device will allow remote commands or not. Remote commands are when another device issues commands direct to this device. This allows the other device to send data, instructions, and to configure this device. S Register 536 set to 1 enables Remote commands. 0 disables it.

### 6.1.10 Saving the configuration

The AT command AT+W causes the current S Register values to be written to Non Volatile Memory so that they are retained when the power is turned off. This allows for a device to be configured for a particular task, and to retain those settings.

### 6.1.11 Implementing the configuration

Once the configuration details are set, the device can be rebooted using the ATZ command to allow the new settings to take affect and configure the device accordingly.

### 6.1.12 A basic configuration set up

A simple set of commands to make the device discoverable are:

AT&F\*  
ATS0=1  
ATS512=4  
ATS536=1  
AT&W  
ATZ

| AT Command | S Register | Description  |
|------------|------------|--|
| AT&F*      |            | Return to factory settings   |
| ATS0=1     | 0          | Number of rings before automatically answering. 1 = 1 ring i.e. straight away.   |
| ATS512=4   | 512        | Specifies power up state. When set to 4 the device will be connectable and discoverable.   |
| ATS536=1   | 536        | Allows the device to enter Remote Command Mode. I.e. to be connected to so that AT commands can be sent to it.                           |
| AT&W       |            | This command causes the current S registers to be written to Non Volatile memory so that they are retained when the power is turned off. |
| ATZ        |            | Reboot to allow the earlier settings to activate and configure the device.   |

## 6.2 Exercise 2: Discoverability

### 6.2.1 Introduction

For a device to be found it needs to be discoverable. A Bluetooth device that is discoverable is able to respond to an Inquiry command from another Bluetooth device.

### 6.2.2 Objectives

- Develop a program that configures the Bluetooth device to be discoverable. This is the complimentary exercise to Exercise 1.

### 6.2.3 Pre-requisites

- An understanding of the Discovery process as detailed in Exercise 1. In particular the ability to be able to recognize an Inquiry command.
- An understanding of receiving and retrieving message data (See the Bluetooth Component Help file for details on the macros involved, and an outline of the relevant strategies)

### 6.2.4 Hardware/Software requirements

The following items of hardware are required:

- Bluetooth solution for the Exercise program.
- 1 or more additional Bluetooth devices for demonstrating the program.  
Note: The second Bluetooth board from the Bluetooth solution can be used for this Exercise.  
The program from Exercise 1 can be used to send the Inquiry command, or the test program BLUETOOTH\_TEST.fcx can be used to set up the board. See the Getting started section for details.
- Set hardware jumpers as specified in the Getting starting section.
- Configure the microcontroller as specified in the Getting starting section.

### 6.2.5 Exercise information

The configuration data to be sent:

```
AT&F*  
ATS0=1  
ATS512=4  
ATS536=1  
AT&W  
ATZ
```

The objectives can be broken down into the following:

Tasks

- Send the configuration commands.
- Check for messages.
- Display any messages.

Control structures:

- Checking for and displaying responses.

### 6.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- Understanding the discovery process.
- An understanding of how to configure the BLU2i device.
- An understanding of S Registers.
- Checking for and retrieving responses.



**6.2.7 Additional tasks**

- Referring in the Bluetooth module manual S Register settings and make the Bluetooth module non-discoverable.
- What implications does this have for the person using the equipment after you?
- Using the programs in Exercise 1 and 2 make a note of the Bluetooth addresses of each of the Bluetooth modules – you will need these in the next exercise.

### 6.3 Practical implementation: Discoverability

Exercise 2 accompanies Exercise 1 in that having shown how Discovering devices work in Exercise 1 the next step is to show how to make devices discoverable. In addition, configuration methods and the use of S Registers are discussed. It is important to stress that Just as the AT commands used here are particular to the specific module used (The EZURiO BLU2i device) so are the S Registers are also a feature of the BLU2i chip and not an inherent part of Bluetooth. Different chips may have different registers, or even use none at all. However the general principles are the same i.e. setting how many rings to wait, or electing to make a device discoverable.

Before starting connect the USB lead to the second Bluetooth system – you can leave the program from Exercise 1 in the first Bluetooth system and we will use it to test if the program created here works.

#### 6.3.1 Continuing development

The Students programs created for Exercise 1 can be used in conjunction with the program created in this exercise to form a discoverable and discovering pair of boards.

The two programs from Exercise 1 and Exercise 2 will form the core part of most of the subsequent exercises, additional commands being added in as the course progresses. As such a continuity approach can be taken where the final program from previous exercises can often be used as the starting point for the next.

#### 6.3.2 Configuring the Bluetooth device

For this Exercise the Bluetooth device needs to be configured to be discoverable and connectable. We will not be connecting to it in this particular Exercise; however we may as well set up this basic configuration now.

For this exercise use the following configuration commands:

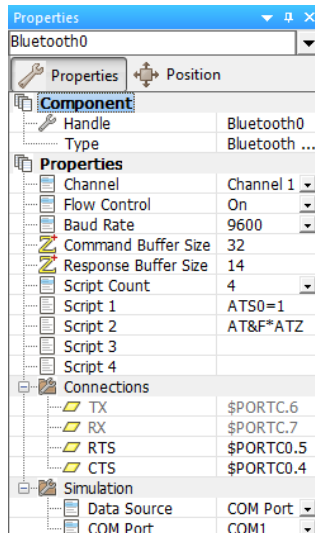
```
AT&F* - optional
ATS0=1
ATS512=4
ATS536=1
AT&W
ATZ
```

These commands will set the device up to answer immediately, and to be both discoverable and connectable.

#### 6.3.3 Single commands and scripts

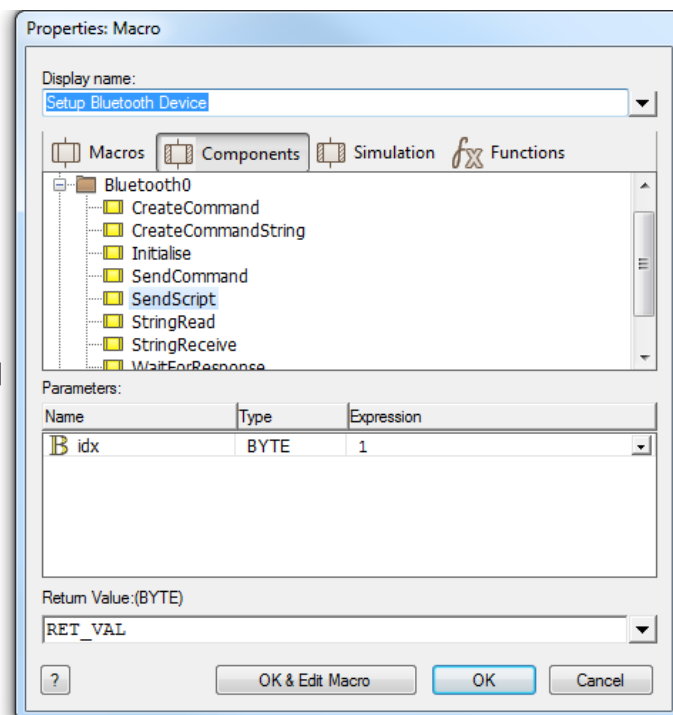
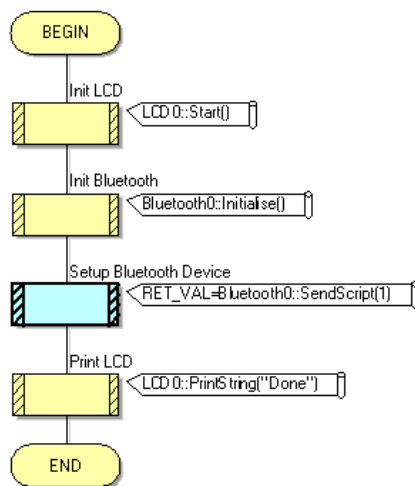
The Commands can be implemented one of two ways: Single commands or via scripts. Single commands can be created and sent using the *CreateCommand* and *SendCommand* macros detailed in Exercise 1. In addition scripts can be used.

Scripts are a set of command entered into a text box. Scripts can be accessed from the 'Properties Panel' when the Bluetooth component is selected. There are 4 separate script properties. The Script Count property can be set from 0-4 allowing use of 0-4 scripts in the program.

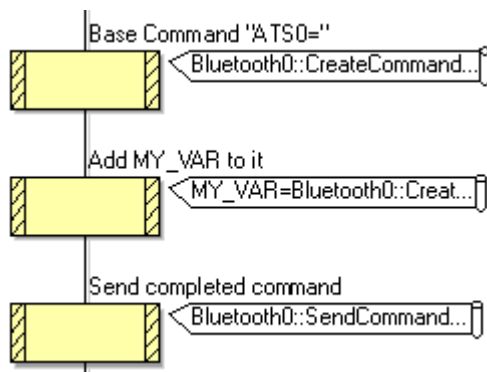


Text can be entered into the text boxes on the script properties. Any text can be entered, AT commands or data for instance. The script is sent one line at a time as if each line were a separate command that had been created and sent using *CreateCommand* / *CreateCommandString* and *SendCommand*.

Scripts are sent using the *SendScript* macro. SendScript takes a single parameter idx, the number of the script to send. The idx value will be 1-4 depending on which script is to be sent. The optional return value is 0 for success or 255 for any problems.



The advantage of using scripts is that you can send a whole group of commands with one single macro. The disadvantage is that the commands in the scripts are static and cannot be built up in the program.



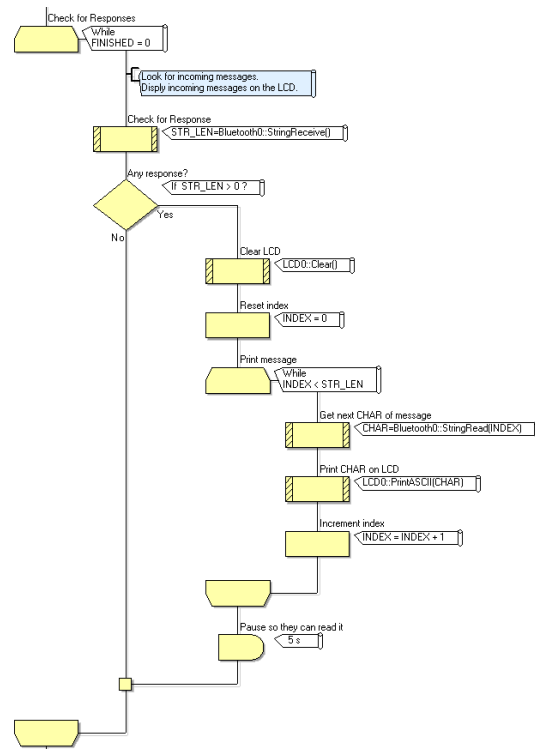
For instance we can set the device to answer immediately using `ATS0=1` in a script, but could build a command such as `ATS0=<MY_VAR>` using a variable `MY_VAR` to set S Register 0 to a program variable that depends on user input. Scripts are extremely useful for blocks of commands that will not vary with program start up. This makes scripts ideal for startup configuration settings.

Adding the text...:

```

ATS0=1
ATS512=4
ATS536=1
AT&W
ATZ
    
```

...to a script allows us to send the whole set of configuration setting commands in one go.



Once the device has been configured to be discoverable nothing else is actually needed for this Exercise. However to allow users to see that the device has received an inquiry command a loop can be added to check for messages and to display them on the LCD. This is a simplified version of the loop created in Exercise 1. There is no need to check for the OK message, or to do anything other than simply display any incoming messages.

### 6.3.4 Running and demonstrating the program

For best effect the program built here can be used with the Inquiry program built in Exercise 1.

The program from Exercise 1 can be used in the first Bluetooth board, and the Exercise 2 program can be used with the second Bluetooth board.

- Create the program, noting as you do the 12 digit device address of the Bluetooth board that will be used with the Exercise 2 program.
- Download and run Exercise 2 on the Bluetooth board whose address you have noted.
- Download and run the Inquiry program created in Exercise 1 to the other board.
- Monitor the LCD Display on the board to be discovered. When the *AT+BTI* Inquiry command is sent it will be displayed on the LCD.
- Monitor the LCD on the Inquiry program board. When the discoverable board is sent the Inquiry command it will respond by returning its 12 bit address (the number you noted down earlier). Check the LCD display to see that the address is displayed.

## **7 Connecting Bluetooth devices**

### **7.1 Theory: Connecting – Addresses**

As no physical connection exists between two devices all communications are capable of being picked up by any other Bluetooth device. An immediate problem is how to communicate with one particular device and not inadvertently communicate with other devices.

All Bluetooth devices have an address, a unique 12 digit hexadecimal number. This address is the same number that is returned in response to an inquiry command. This address can be either retrieved via an inquiry command, or can be stored in the program if a specific device address is to be used and is known in advance.

If a Bluetooth device is connectable and the address is known then other Bluetooth devices can initiate connection to that device. Generally devices will be discoverable and connectable so that the address can be determined from an Inquiry command. However it is possible for devices to be connectable, but not discoverable. For instance a set of phones that are designed to communicate only with each other may be set to be connectable but not discoverable. The pair of phones have the address of each phone in memory so that a discovery command is not needed, and the phone is ready to start communicating with its partner device.

The basic connection command is:  
`ATD<bt_addr>`

Where <bt\_addr> is the 12 digit hexadecimal address of the device to initiate connection with.

The Bluetooth addresses of the BLU2i modules are set in the factory and cannot be changed. The address is written on both the underside of the Bluetooth E-Block and the underside of the BLU2i module itself. Note that if the BLU2i modules are replaced or swapped for any reason, the address displayed on the underside of the Bluetooth E-Block will not be the same as the address of the BLU2i module itself.

## 7.2 Exercise 3: Connecting to a device

### 7.2.1 Introduction

If the address of a Bluetooth device is known, and the device has been configured to be connectable, then that device can be connected to.

### 7.2.2 Objectives

- Develop a program in one Bluetooth system (node B) that connects to another Bluetooth system (node A), and displays the reply to show that the connection has succeeded.
- Develop a program in one Bluetooth system (node A) that allows another Bluetooth system (node B) to connect to it. This program is supplied for you in BT\_EX3\_Node\_A.fcfx.

### 7.2.3 Pre-requisites

- An understanding of the Discovery process as detailed in Exercise 1 and Exercise 2.

### 7.2.4 Hardware/Software requirements

The following items of hardware are required:

- Bluetooth solution for the Exercise program.
- 1 or more additional Bluetooth devices for demonstrating the program.  
Note: The second Bluetooth board from the Bluetooth solution can be used for this Exercise.  
Download the program BT\_EX3\_Node\_A.fcfx into this second system.
- Set hardware jumpers as specified in the Getting Starting section.
- Configure the microcontroller as specified in the Getting Starting section.

### 7.2.5 Exercise information

The Initiate Connection command is:

ATD<bt\_addr>

Where <bt\_addr> is the address of the second Bluetooth board.

Set up two programs: one in Node B to Dial a second Bluetooth device in Node A. When you issue the ATD command the receiving node will send the acknowledgement CONNECT 123456789012. Display this on the LCD display of node A.

### 7.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- Connecting to another Bluetooth device.

### 7.2.7 Further work

The LCD display only has 16 characters. Devise an extension to the code in BT\_EX3\_Node\_A.FCFX so that all any text overflow is displayed on line 2 of the display.

## 7.3 Practical implementation: Connecting

Like previous exercises students actually need to develop two programs – one in the Bluetooth device initializing the communication, and the second in the device receiving the initial communication.

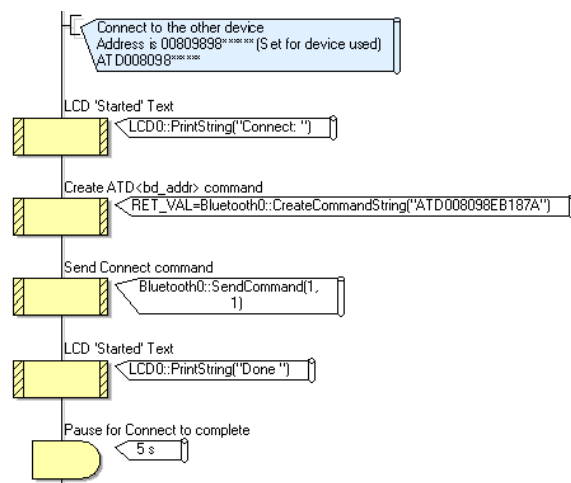
This exercise is a quick and simple one from the workload point of view, but has a number of technical pitfalls to be aware of.

1) Device address. The example program BT\_EX3\_Node\_B.FCFX uses the address of a Bluetooth device used for testing here at Matrix. This will need to be edited to match the address of the board that is to be used with the example program. For this you should use the address you found in Exercise 2.

Be clear about which is the sending node and which is the receiving node and what their addresses are.

2) The Bluetooth device to be connected to needs to have been configured to be connectable. Once again the program BT\_EX3\_Node\_A.FCFX has been set up with this issue in mind.

3) Once a connection has been made there needs to be some way of verifying to ourselves that the connection has been made. The program BT\_EX3\_Node\_A.FCFX is set up to display any messages sent, so we can simply send a message and check that it is received by the other device. The program you write should display the returning message from node A.



A delay is needed after the connection command has been sent to allow time for the connection messages to be handled. No error checking is done at present. The program assumes that the connection will succeed, which may not always be the case. A better method would be to use responses to error check the process. Responses and error checking will be dealt with in a later chapter.

### 7.3.1 Resetting the systems

The Bluetooth board EB024 is not equipped with a reset button. The only mechanism of issuing a reset is to remove power. If you have programmed the Bluetooth module to carry out some activity then pressing the reset button on the Multiprogrammer board will not necessarily reset the Bluetooth module.

For this reason when developing pairs of programs it **may** be necessary to remove power from the system and reboot the Bluetooth modules.

For this exercise we recommend that you remove power from both systems each time you download a program. Then power up the receiving system, press reset and give it a few seconds to set up. Then power up the transmitting system, press reset and give it a few seconds to set up.



## 8 Passkeys and Pairing

### 8.1 Theory: Passkeys and Pairing

Bluetooth communicates via radio which is an inherently unsecure transmission medium. Any device within transmission range can receive the signals sent to any other device. Using unique address solves the problem of specifying which device you intend to communicate to, but other methods are needed to prevent unauthorized access to a device.

The basic security system used with Bluetooth is called Pairing. Pairing is when two devices connect to each other using both a device address and the device's secret "link key" known as the Passkey. The Passkey cannot be retrieved from another device in the same way the address can. To be able to connect to a device you need to know the Passkey number for that device.

#### 8.1.1 Sending the Passkey command

To set up a device so that it can be paired with a Passkey needs to be set up for the device. The command to set up a Passkey is:

```
AT+BTK=<passkey>
```

Where <passkey> is the Passkey PIN number. The Passkey is a 0-8 digit number. Setting the Passkey to a blank string clears the current passkey. Generally Passkey PIN numbers are supplied with documentation that accompanies a particular device. Device PIN numbers should be kept safe, just like you would with bank card PIN numbers.

For the programs in this course a simple Passkey of "1234" will be used for all exercises

#### 8.1.2 Initiating pairing

To pair to a device the device initiating contact needs to send two commands

A passkey command

```
AT+BTK=<passkey>
```

Where <passkey> is the passkey value of the device to be paired with.

An Initiate Pairing command is:

```
AT+BTW<bt_addr>
```

Where <bt\_addr> is the address of the Bluetooth device to be paired with.

#### 8.1.3 When to send the Passkey command

The Passkey command can be sent before or after the Initiate Pairing command. If the Passkey command is sent before then Pairing will be initiated upon sending the Passkey command. If the Passkey Command has not been sent when an Initiate Pairing command is received the device to be paired to will send an unsolicited PIN? response which will need to be checked for by the program. Pairing will not proceed until the Passkey is sent. Responses will be covered in a later chapter, so for now send the Passkey command before the Pairing command. An Initiate Pairing command on its own is not enough. A passkey command needs to be sent as well.

In many instances, such as when pairing a mobile phone with a headset, the Passkey will be in the product documentation and the program will prompt you to enter the Passkey.

## 8.2 Exercise 4: Passkeys and Pairing

### 8.2.1 Introduction

Devices can be paired with each other for communications. Pairing requires the address of the device to pair with, and a Passkey value to establish contact. Here you will need to develop two programs – one that establishes what the pass key is in a system, and the other that connects to that system and sends data to it.

### 8.2.2 Objectives

- Develop programs for two Bluetooth systems that allow pairing to take place. Develop a program for Node A that assigns a Passkey of “1234”, and make it discoverable. Develop routines that show any data sent to the node on the LCD. Develop a program for node B that Pairs with node A using Node A's address and Passkey.
- Once Paired establish a simple communication between the systems that shows communication is taking place – i.e. a counter on the sending count data as a single byte which is also displayed on the receiving node.

### 8.2.3 Pre-requisites

- An understanding of the connection process as detailed in Exercise 3.
- An understanding of creating, sending and receiving commands as detailed in Exercises 1 and 2.

### 8.2.4 Hardware/Software requirements

The following items of hardware are required:

- Both Bluetooth solutions are required for this Exercise.
- Solution 1 will initiate pairing.
- Solution 2 will be paired to.
- Set hardware jumpers as specified in the Getting starting section.
- Configure the microcontroller as specified in the Getting starting section.

### 8.2.5 Exercise information

The Initiate pairing commands are:

- `AT+BTK=<passkey>`  
Where <passkey> is the passkey value of the device to be paired with.
- `AT+BTW<bt_addr>`  
Where <bt\_addr> is the address of the second Bluetooth board.

Once a connection has been established loop through and send the numbers 0-9 to be displayed on the other device.

### 8.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- Understanding of the pairing process.
- Understanding of the role of the Passkey.
- Understanding of the security implications of the Passkey.

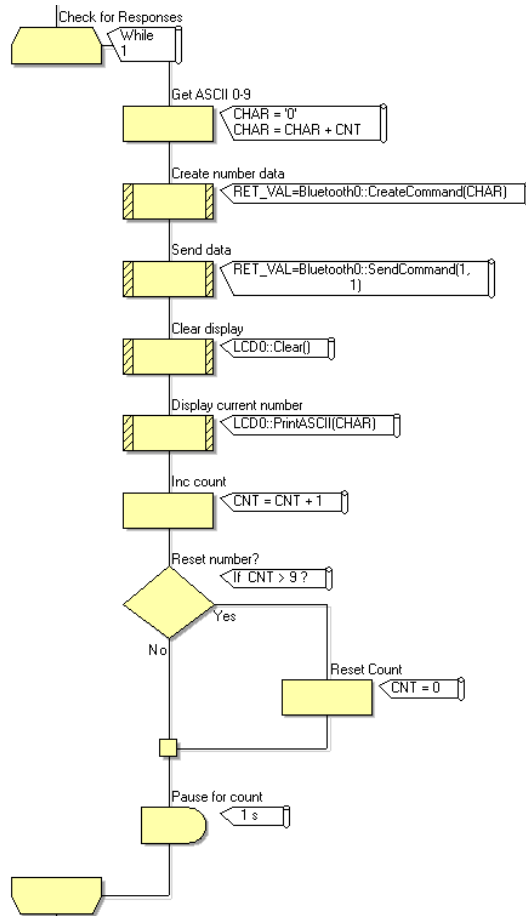
### 8.2.7 Further work

- Using a keypad and an array variable, develop two programs that allow the passkey of both the receiving system to be set, and that allows the passkey used by the transmitting system to be set.
- Develop a program that displays the ‘friendly name’ of all other Bluetooth devices in the locality.

## 8.3 Practical implementation: Passkeys and pairing

### 8.3.1 The basic program

Pairing is a key process in Bluetooth operations. The pairing process is relatively straight forward on one level, but complex on another. Pairing involves the sending of both the Passkey and an initiate pairing command.



AT+BTK=<passkey>  
AT+BTW<bd\_addr>

For Example:  
AT+BTK="1234"  
AT+BTW012345678912

If the details are known in advance then they can be simply entered into the program and the commands built and sent. If the passkey and address are not known in advance then they need to be established.

The boards in the Bluetooth solution have the device addresses on labels on the bottom of both the BLU2i module board, and on the bottom of the E-block. You may wish to note down the addresses for the various boards.

To demonstrate the connection is working a simple 0-9 repeating signal can be sent to be displayed on the receiving device.

Note: The delay icons used in this chapter allow for the devices to finish communicating with each other. A better method would be to know when a process is complete, or more importantly if it has failed. This can be done with responses, which will be covered next chapter.

### 8.3.2 Advanced features: Choosing what device to connect to

Device addresses can be retrieved using the Inquiry command. However knowing if we need to connect to device 007284972989 or 00234869030 is another matter. The AT+BTI Inquiry command can be extended to retrieve extra information above and beyond the 12 digit address. AT+BTIN retrieves not only the device address, but the friendly name of the device as well. The friendly name is a device description string that is easier for users to understand than the device address.

For example:  
012345678912, "TDK BLU2i RS232"  
012345678912, "TDK Headset S102"

Displaying the friendly address can help users select the correct device out of a list. The practicalities of displaying the friendly name are a simple modification to the display response code from Example 1. Adding in the command space the first 14 characters can be displayed on the top line of the LCD, and the rest on the bottom line by moving the cursor when the first 14 characters have been displayed.

In addition we can add the 12 address characters into an array, storing the current address for further use. This can be done at the same time as we are printing out the characters. Add the first 12 characters to the current address.

Obtaining the passkey is more difficult as this it cannot be retrieved via an AT command. For the examples used in this course the Passkey "1234" is used throughout. In practice a passkey may need to be obtained from the device documentation and either entered in the program code, or entered into the program manually.

### **8.3.3 Resetting the systems**

The Bluetooth board EB024 is not equipped with a reset button. The only mechanism of issuing a reset is to remove power. If you have programmed the Bluetooth module to carry out some activity then pressing the reset button on the Multiprogrammer board will not necessarily reset the Bluetooth module.

For this reason when developing pairs of programs it may be necessary to remove power from the system and reboot the Bluetooth modules.

For this exercise we recommend that you remove power from both systems each time you download a program. Then power up the receiving system, press reset and give it a few seconds to set up. Then power up the transmitting system, press reset and give it a few seconds to set up.

## 9 Checking responses

### 9.1 Theory: Checking responses

In the previous exercise the pairing worked but only with a large delay put in to allow the devices to work through the pairing messages and responses. This is all well and good if we know that the devices will pair correctly, and can afford to wait until it is definitely done. However what happens if there is a problem, or the connection cannot be established in the delay we set? A more useful method would be to monitor the process checking for the right response at the right time.

When a command is sent a response is generally sent to that command. For many commands and occasions the response will be an 'OK' message. However many commands have specific responses. The Discover command for instance sends addresses as responses, and once finished then it sends an OK response.

#### 9.1.1 Solicited and unsolicited responses

When a response is the direct result of sending a command it is a solicited response i.e. a response that is expected. For instance sending an AT+BTK="1234" command will result in a solicited response of "OK".

Sometimes a response is sent that is not in response to a specific command. This is an 'unsolicited' response. For instance when an initiate pairing command is sent an expected response (solicited) of "OK". will be sent. However if the Passkey has not been provided an unsolicited response of "PIN?" will also be sent. This is an unsolicited response as it is not in response to the command sent but is instead a prompt to the device to provide extra information, in this case the Passkey value. Unsolicited responses can be sent at various times to indicate either a need for further interaction, or to provide additional feedback. Examples include the CONNECT and RING responses sent during the establishment of communications to inform the other device of the current status of the operation.

Unsolicited responses are sent when required. Using the pairing example from above, if the Passkey command had been sent before the Initiate pairing command then an unsolicited "PIN?" Response would not be needed and would not therefore be sent.

Unsolicited responses are detailed in the AT Commands Set document.

#### 9.1.2 Response handling macros

There are two macros available to help with handling responses: StringReceive and WaitForResponse.

The StringReceive macro checks to see if a response has arrived and returns 0 or the length of the response message. *StringReceive* was used in the section on Discovery and details of using *StringReceive* can be found in the Help file in Flowcode. *StringReceive* is useful for general message monitoring when messages may be present. Exercise 1 demonstrates *StringReceive* in action monitoring incoming messages for Addresses.

*WaitForResponse* pauses program flow until a response message is received.

*WaitForResponse* takes the parameters response\_code and timeout.

The response\_code parameter indicates the type of response to wait for. This takes the form of a 1-7 value that refers to the following response types:

| Response type | response_code value | Notes  |
|---------------|---------------------|--|
| OK            | 1                   | Can means no more than confirming the command has been successfully received.<br>For Example An OK with AT+BTW<bt_addr> does not confirm pairing occurred; only that |

|                    |   |  |
|--------------------|---|--|
|                    |   | the command to initiate pairing was received.  |
| ERROR <xx>         | 2 | See AT Command Set document for details of ERROR code.<br>Examples include:<br>09 = Invalid Bluetooth address<br>16 = Pairing in progress<br>24 = Remote address same as local address<br>33 = S Register value is invalid |
| CONNECT <bt_addr>  | 3 |  |
| NO CARRIER         | 4 |  |
| AUDIO <string>     | 5 | <string> indicates the status of the Audio channel. Can be:<br>ON, OFF or FAIL   |
| PAIR <n> <bt_addr> | 6 | Values for <n>:<br>0 = success<br>1 = timeout occurred<br>2 = Other unsuccessful outcome.  |
| RING <bt_addr>     | 7 |  |

The timeout parameter is the number of milliseconds to wait before timing out and assuming that no response is coming. Normally this will be used to help indicate if a command failed or not.

The return value for *WaitForResponse* is as follows:

- Returns zero to indicate that the specified response has been received.
- Returns 0xFF for timeout or an illegal response code
- Returns the number of characters received if an unexpected response is received

*WaitForResponse* can be used for both error checking and sequence control. Sequence control can be accomplished by checking for the expected response type. For example an Initiate pairing command should return a PAIR <bt\_addr> response (response\_code value 6). Error checking can be done by checking for timeouts or unexpected responses. If a timeout occurs the expected response has not arrived and appropriate action needs to be taken. Generally a timeout will indicate a communications failure at that stage of the sequence. If an unexpected response occurs it can be further checked to see if it requires action (e.g. it is an unsolicited response) or if it is an error message that needs processing or reporting.

The *WaitForResponse* command does not look for a specific error number, address or audio state. *WaitForResponse* indicates if a response of the specified type was received or not. To find check the specific details of a response the *StringReceive* command can be used to retrieve the response for further checking.

For example, to find out a specific Bluetooth address after receiving zero (success) from a *WaitForResponse* (3) command, send the *StringReceive* command. The received string will then be the Bluetooth address.

### 9.1.3 Command and response sequences

When a command is sent one or more responses are returned. For example when a set of Pairing commands are sent. The sequence will be something like:

|                     |   |
|---------------------|---|
| AT+BTW0123456789012 | - Command sent (request to initiate pairing)  |
| OK                  | - Response (Pair command received and is ok). |
| PIN?                | - Unsolicited response (PIN number request)   |
| AT+BTK="1234"       | - Command sent (Passkey)                      |
| OK                  | - Response (OK)                               |
| PAIR 0 012345678912 | - Unsolicited response (Pair succeeded)       |

Note the two unsolicited responses. The AT protocol is a Command/Response protocol. A command is responded to. In the case of the AT+BTW<bd\_addr> command the response is OK to indicate that the command has been received. However the command itself requires further action when it is processed. These actions cause the device to issue an unsolicited response. In this case firstly a request for the passkey to be sent. Later, once the Passkey has been sent and responded to with an OK to say that the command has been received; a further unsolicited response is sent to inform the device that Pairing was successful.

*Side note: For some commands, such as AT+BTW<bt\_addr> the OK response confirms that the command was received correctly. It does not mean that the request was successful. Subsequent responses may need to be examined to indicate that the command was successful.*

Details of the expected responses can be found in the AT Command Set documentation. By checking that the expected response has arrived it is possible to determine whether an error has occurred, or when something unexpected has occurred. For instance a return value of 255 (0xFF) indicates that a timeout has occurred, possibly due to a connection issue. A response to an Inquiry with AUDIO or CONNECT 123456789012 would indicate that something has gone wrong with the sequence as a different response was expected.



## 9.2 Exercise 5: Checking responses

### 9.2.1 Introduction

When a device receives a command it returns a response. By checking these responses it is possible to both monitor for any errors that occur and to wait for expected responses before continuing a sequence of commands.

### 9.2.2 Objectives

- Develop programs for two Bluetooth systems that allow pairing to take place. Develop a program for Node A that assigns a Passkey of “1234”, and make it discoverable. Develop routines that show any data sent to the node on the LCD. You can use the program from Exercise 4 for this.
- Develop a program for node B that Pairs with node A using Node A’s address. Don’t use the Passkey with the Pair command – alter the program you wrote in Exercise 4 to include routines that provide the Passkey when prompted. Once Paired wait for and check the responses to ensure that the pairing completes successfully.
- Report any errors that occur on node B

### 9.2.3 Pre-requisites

- An understanding of the Pairing process as detailed in Exercise 4.
- An understanding of the WaitForResponse macro as detailed in the Bluetooth component help file.
- An understanding of the Response types as detailed in the Bluetooth component help file.

### 9.2.4 Hardware/Software requirements

The following items of hardware are required:

- Both Bluetooth solutions are required for this Exercise.
- Solution 1 will initiate pairing.
- Solution 2 will be paired to.
- Set hardware jumpers and configuration as specified in the Getting starting section.

### 9.2.5 Exercise information

The Initiate pairing commands are:

- `AT+BTK=<passkey>`  
Where <passkey> is the passkey value of the device to be paired with.  
Expected response is OK.
- `AT+BTW<bt_addr>`  
Where <bt\_addr> is the address of the second Bluetooth board.  
Expected response is OK.
- Once Pairing has occurred an unsolicited response PAIR <bd\_addr> will be sent.
- If the Initiate connection command is successful a CONNECT <bd\_addr> response will be returned.
- Load the program BLUETOOTH\_TEST.FCFX on Bluetooth Solution 2.
- A list of response types can be found below.

### 9.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- Command and response sequences.
- Response types.
- Error checking and sequence monitoring.
- Unsolicited responses.

| Response type | response_code value |
|---------------|---------------------|
| OK            | 1                   |
| ERROR <xx>    | 2                   |

|                         |   |
|-------------------------|---|
| CONNECT <bd_addr>       | 3 |
| NO CARRIER              | 4 |
| AUDIO <n> <audio state> | 5 |
| PAIR <bd_addr>          | 6 |
| RING <bd_addr>          | 7 |

## 9.3 Practical implementation: Checking responses

Two important bits of information to note down are the Bluetooth Device addresses, and the Passkeys. For this exercise it is assumed that the general Passkey “1234” will be used. However the Bluetooth devices will need to be modified accordingly.

The task requires two programs, one to initiate pairing, and one to display data sent to the paired device. The Pairing programs developed in Exercise 4 can be used as the base of Exercise 5 allowing the student to see how their code progresses.

The display program only needs to display data sent, so the display program from Exercise 4 can be used as is.

The main focus will be with the Pairing program, and expanding that program to include response checking.

### 9.3.1 Establishing the expected sequence

Before a sequence can be error checked the pattern of commands and expected responses must be established.

|                      |                               |
|----------------------|-------------------------------|
| AT+BTK="1234"        | - Passkey command             |
| OK                   | - Response. Type 1.           |
| AT+BTW0123456789012  | - Initiate pairing command    |
| OK                   | - Response. Type 1.           |
| PAIR 0 012345678912  | - Response. Type 6.           |
| ATD0123456789012     | - Initiate connection command |
| CONNECT 012345678912 | - Response. Type 3.           |

The sequence, with types and suggested timeout values is as follows:

```
SendScript(1): ATZ – Reset
                WaitForResponse(1,200)
                OK response from ATZ

SendScript(3): AT+BTK="1234"
                WaitForResponse(1,10)
                OK response from Passkey command

SendScript(2): AT+BTW00809894E5DF - Pairing command
                WaitForResponse(1,10)
                OK response from Initiate pairing command
....
                WaitForResponse(6,200) – Unsolicited PAIR command sent when pairing complete.

SendScript(4): ATD00809894E5DF
                WaitForResponse (3,200)
```

### 9.3.2 Additional sequence considerations

The above sequence is a simple one as all the Response type are covered and can be waited for. However if the Passkey command has not been sent when the Initiate pairing command is sent then an unsolicited PIN? Response will be received after the solicited OK response. The PIN? Response will be registered as an unexpected response and will need to be trapped and the appropriate action taken (namely sending the Passkey command). It is recommended that students send the Passkey first to simplify the program initially. Advanced students can then tackle the unsolicited PIN? response as both an exercise in handling unsolicited responses and in advanced error checking.

### 9.3.3 Using the WaitForResponse macro

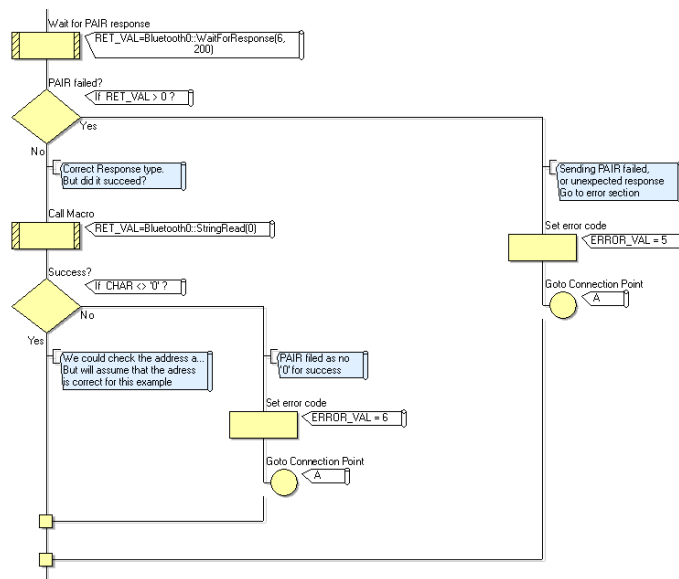
Once a sequence is known adding WaitForResponse macros is relatively straightforward. Add the macro and edit the properties so that the expected response code parameter uses the correct response type, and an appropriate timeout value is used. Set a return value (RETVAL is the generic return value variable used throughout this course) for the error checking. The timeout value can vary depending on factors such as baud rate and speed of operation. The example values set out below are the ones used successfully when developing this course and can be used as guides to setting the values.

Example timeouts values used for the example programs is:

| Response                     | Value in milliseconds |
|------------------------------|-----------------------|
| Awaiting general OK response | 10 - 100              |
| ATZ reset OK response        | 200                   |
| PAIR command                 | 200                   |
| CONNECT response             | 200                   |
| RING                         | 200                   |
| AUDIO                        | 200                   |

### 9.3.4 Error checking methodology

A basic error checking methodology has been adopted here. As the commands are created and sent they are first checked using the return value from the functions. A failure here indicates that there is a problem with either creating or trying to send the command.



Once a command has been sent the WaitForResponse return value can be checked to see if it received a response of the specified type (0 for success). If not an error message is displayed on the LCD detailing the stage at which the sequence failed. A connection icon is used to jump to the end of the program to exit without running the rest of the program.

If a successful response has been received this too may need further checking to determine that the action was both successful and that the result was what was expected.

The WaitForResponse (6,200) command will only wait for the “PAIR” response (plus a following space character), which on its own does not indicate a successful pairing. Immediately following the reception of “PAIR” is a single digit representing the outcome of the pairing attempt. 0 means success, 1 means timeout and 2 means another unsuccessful outcome. Following this single digit is a space, then the 12-digit hexadecimal string representing the address of the other Bluetooth device. In this example both the outcome and the address need to be checked to ensure that we have successfully paired to the correct device.

Advanced error checking could involve separate user macros testing the responses and giving more detailed information – e.g. displaying ERROR <nn> codes, or checking for a NO CARRIER and giving users the option to turn on the other device and try again etc.

## 10 Command modes - Sending data and commands

### 10.1 Theory of Command modes

Once a connection has been made data and commands can be sent to the device. Data sent by the sending device will be received on the receiving device and can be retrieved with `StringReceive`. `StringRead` can then be used to access the raw data. Raw data is not handled by Bluetooth but by the program code itself.

#### 10.1.1 Data mode, Remote and local command mode

- Data mode should be used when sending data – not when sending commands.
- Local command mode is where commands are sent to a remote device.
- Remote command mode is where commands are issued to the receiving device.

Data mode is established by default upon connection. To return to Data mode from one of the other command modes, send the AT command `ATO`. Data mode is basically the sending of raw data that the application on the receiving device can use. The sending program and the receiving program are responsible for understanding, handling and formatting this raw data.

Local command mode is established by sending the escape sequence `^^` whilst in Data mode.

Entering Local Command Mode allows your program to alter settings and perform specific Bluetooth actions such as ending the communication link or establishing an audio channel.

Remote command mode is established by sending the escape sequence `!!!` whilst in Data mode. In remote command mode commands subsequently sent are processed by the receiving device. For instance an `ATS0=3` command would set the S Register 0 on the receiving device to value 3. The local sending device would not be affected, only the remote receiving device.

Remote command mode allows for the configuration of the remote device and/or the querying of the device, but only if the remote device is set up to allow remote capture. Remote capture is controlled by the value of S Register 536. The default S Register 536 value of 0 blocks remote capture. Setting S Register 536 to 1 allows remote capture. The AT command for setting the register to allow remote capture is:  
`ATS536=1`

#### 10.1.2 Local command mode `+++` and `^^^`

AT commands were originally designed for use with PC Modems. The local command mode escape sequence for modems is generally `+++`. In order to avoid confusion with Bluetooth to modem signals the Bluetooth escape sequence is generally `^^^`. The escape sequence character is set in S Register 2. This allows the character to be modified if required. However using the default `^` character is highly recommended.

#### 10.1.3 Guard gaps and escape sequences

Local and remote command modes are established by sending an escape sequence. An escape sequence differs from a command by sending a particular character a set number of times. For Bluetooth the escape character needs to be sent three times. To guard against a command mode being activated by the inadvertent sending of the three escape sequence characters, a guard gap is used. The guard time is set in S Register 12 and can be from 40-5000ms (with a resolution of 20ms). The default guard time is 100ms. The three characters must be sent with a gap between them of at least `<guard time>` ms or more.

## 10.2 Command modes: Exercise 6

### 10.2.1 Introduction

As well as sending data Bluetooth devices can send commands either to themselves, or to another device. This Exercise will cover sending Data, local and remote commands.

### 10.2.2 Objectives

- Develop a program that pairs two Bluetooth devices and enters Local command mode.
- Send the AT+I4 command to retrieve that retrieves the address of the active device.
- Examine which address was retrieved. Was it the address of the device receiving the AT+I4 command or sending AT+I4?

You should use the programs you have previously developed as a starting point.

### 10.2.3 Pre-requisites

- An understanding of the Pairing process as detailed in Exercise 4.

### 10.2.4 Hardware/Software requirements

The following items of hardware are required:

- Both Bluetooth solutions are required for this Exercise.
- Solution 1 will initiate pairing.
- Solution 2 will be paired to.
- Set hardware jumpers and configuration as specified in the Getting starting section.

### 10.2.5 Exercise information

The Local Command mode command is:

- `^L` Local command mode.

Note that Local command mode needs to be sent with guard time delays:

`<guard time><Esc char><guard time>< Esc char ><guard time>< Esc char ><guard time>`

Load the program BLUETOOTH\_TEST.FCFX on Bluetooth Solution 2.

Once Local command mode has been established send an AT+I4 command to retrieve and display the Bluetooth address of the device sent the commands. This will help show which device responded to the command.

### 10.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- Command modes.
- Escape characters and guard times.
- Local command mode.

### 10.2.7 Further work

Develop your program further to set the number of rings the slave device answers on to 5. Use your previous experience to check that you have made the settings correctly.

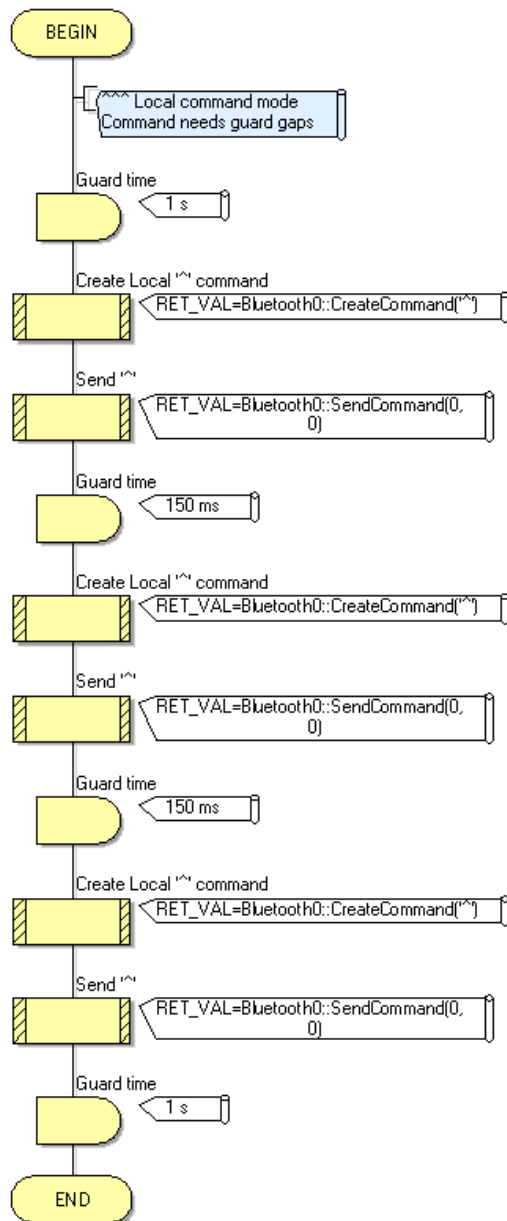
## 10.3 Practical implementation: Command modes

Due to the guard time requirement, the !!! or ^^ command cannot be sent in a script. The script will not wait the required guard gap time between characters and so will fail to be recognized as the command mode escape sequence.

A command mode escape sequence will be structured as follows:

<guard time><Esc char><guard time>< Esc char ><guard time>< Esc char ><guard time>

Enter LOCAL command mode



response number.

For this exercise *ATI4* is ideal as it returns the device address allowing us to determine which device responded to the command. By sending the *ATI4* command in local command modes we can see which device responded.

Delay 1s

Create + send ^

Delay 150ms

Create + send ^

Delay 150ms

Create + send ^

Delay 150ms / WaitForResponse – OK

Note that a delay of at least guard time is required. Assuming the default 100ms delay the 150ms delay used here should suffice. If the guard time were longer the delays would need adjusting accordingly. The ^ escape character is then sent, and the next guard time delay added until all three escape characters have been sent. An OK response will be sent once the command mode escape sequence has been sent, so the *WaitForResponse* will need to wait for a type 1 (OK) response.

### 10.3.1 Which device is being sent commands?

When first investigating Local and Remote command modes it can be confusing to figure out which device is being sent the command. Is local command mode affecting Device A? Or is the command affecting Device B that Device A is communicated with? One command that can help us figure that out is the *ATI<n>* command.

*ATI<n>* requests information from the device. The information requested depends on the value of <n>. The full list of <n> values, and the information retrieved can be found in the AT Command Set document. Below are a few that you may find useful.

- 0 – Returns the product name/variant.
- 4 – Returns the 12 digit device address.
- 9 – Returns 1 if connected otherwise returns 0.
- 12 – Returns the last ERROR

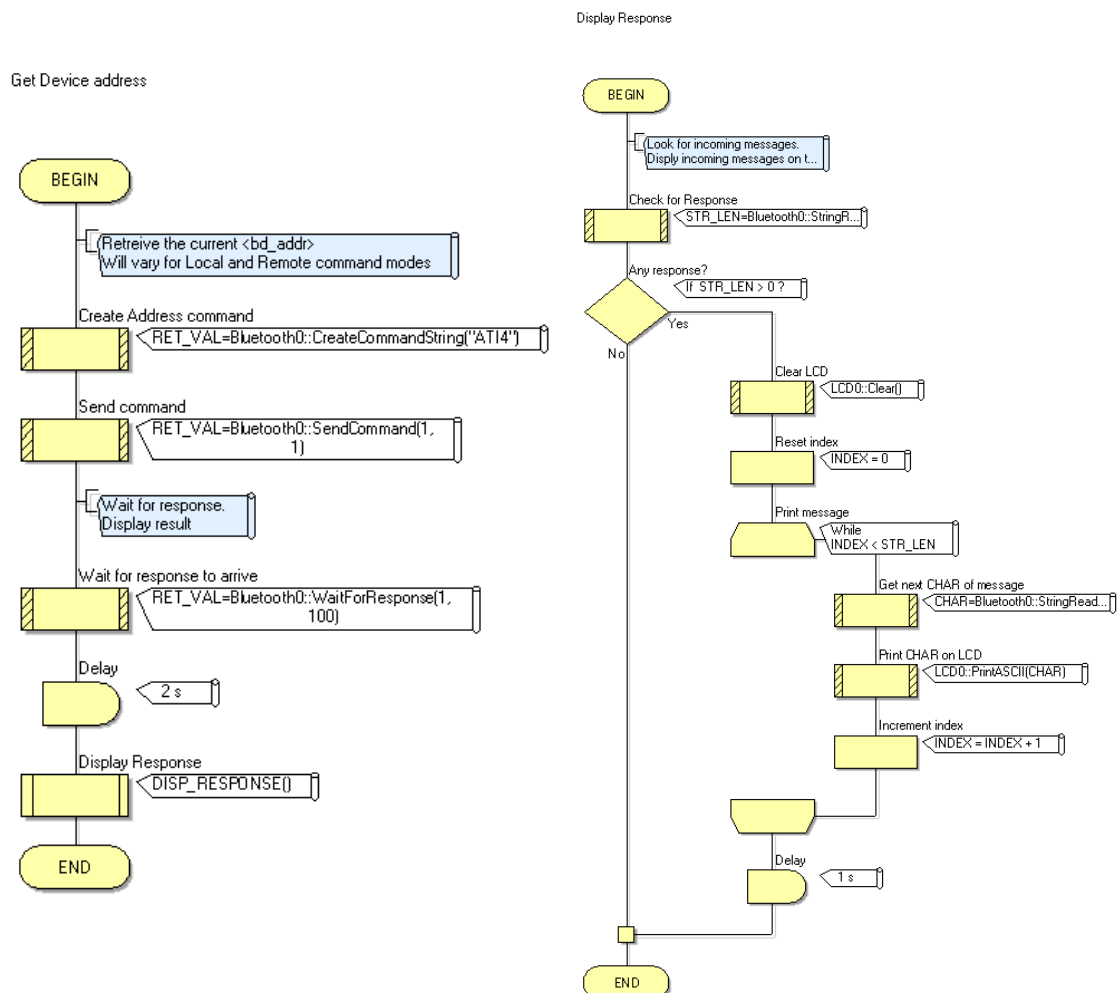


When in Local command mode send the AT+I4 command and display the resulting device address. Before running the program note down which device address you expect when and see if you get it right.

### 10.3.2 Program overview

The objective is to check the Local command mode and request the address of the device executing commands. Data or information is requested and the results monitored. A simple task list outline is shown to the left. The Send Local command and Send Remote commands need to have the guard gaps and separate send commands. As such they are prime candidates for implementing as user macros.

In order to investigate which device is currently receiving commands the information request, AT+I4, is sent to retrieve the currently active device's address. Once this has been retrieved it can be displayed on the LCD Display. The code snippet below shows the outlines of a macro designed to get the device address. Note the use of a Display response macro. Putting the display response code in a separate macro allows it to be called to show the result of specific actions, such as here, and to be used in a general response monitoring loop at the end of the program.



## 11 Audio communication

### 11.1 Theory: Audio communication

Audio is handled in a special way in Bluetooth. Bluetooth has the ability to fast track audio signals through the system allowing for low latency communications i.e. voice communications. An audio specific AT command is available that opens or closes the fast track audio channel.

Latency is the delay between the data being sent and the data arriving. For real time voice communications this is a major issue as waiting several seconds to hear what the other person said is confusing and difficult to work with. The Bluetooth fast track allows for much lower latency speeds but at a cost in quality as higher quality means more data to send. For voice communications the quality is normally more than adequate.

This assumes that the application is for voice communications such as a phone call, or walkie-talkie, where low latency is more important than quality. For quality based systems where latency is not an issue, such as sending music data for an MP3 music player standard data transfer would generally be used in preference to the audio fast track, with the data being buffered at the receiving end before being played.

#### 11.1.1 Hardware requirements

The Blu2i Bluetooth device is capable of sending audio signals but is not an audio device. The raw audio signals need to be converted to a signal that the Bluetooth device can handle. In the Bluetooth solution this is accomplished by the Voice Codec board. To send and receive audio a microphone and or headphone/speaker is required. The Voice Codec board has connections for both a microphone and a headset via music industry standard 3.5mm jack plugs. Two sets of headphones with attached microphones are supplied with the Bluetooth solution for use with the Voice Codec boards allowing voice communication to be tested.

#### 11.1.2 Establishing an Audio connection

Before audio can be accessed, the receiving device needs to be in Local command mode. Local command mode is covered in Exercise 6. The Local command mode is entered using `^^`. Remember that Guard gaps are required between the escape characters. Once Local command mode has been established the audio commands can be sent.

The audio command is:

`AT+BTA<n>`

Where `<n>` is 0 for Audio channel closed, or 1 for Audio channel open.

The first step is to connect to the other device. Once data communication between the two devices has been established Local command mode can be entered using `^^`.

Once in local command mode the audio channel can be enabled using `AT+BTA1`

An OK response is sent confirming that the audio command has been received. Next the device will attempt to open the audio channel and will send an additional AUDIO response message indicating the result of the audio channel command.

The AUDIO response will be either AUDIO ON or AUDIO OFF indicating the audio state, or AUDIO FAIL if the audio state could not be set for any reason. The response will need checking to ensure that the audio channel is now on.

Once the audio channel is on audio signals are fast tracked through the Bluetooth system by the Bluetooth devices. Assuming both devices have a microphone and a headset voice communications will now be possible. Check headset and microphone connections and volume controls to set appropriate volume levels. The Bluetooth audio channels do not have volume controls so the attached hardware volume controls will need to be used.

Once the audio channel is established it will run automatically until turned off. Further commands are not required. At this stage you can re-enter data mode if you also intend to transmit other data by sending the Data mode command ATO.

To terminate the audio connection you need to be in Local command mode. If the device has been switched out of Local command mode it will need to be reestablished (^^^).

In Local command mode the audio channel can be closed with the command AT+BTAA0. An OK response is sent indicating that the audio command was received, followed by an AUDIO response of AUDIO ON, AUDIO OFF or AUDIO FAIL indicating the new audio channel state.

## 11.2 Exercise 7: Audio communication

### 11.2.1 Introduction

A prime use of Bluetooth technologies is in telecommunications. For many Bluetooth is synonymous with the Mobile Phone and the Bluetooth headset. In this exercise we will be setting up voice communications as used by mobile phones.

### 11.2.2 Objectives

- Develop a pair of programs that create a voice communications link between two Bluetooth devices.

### 11.2.3 Pre-requisites

- An understanding of the Pairing process as detailed in Exercise 4.

### 11.2.4 Hardware/Software requirements

The following items of hardware are required:

- Both Bluetooth solutions are required for this Exercise.
- Solution 1 will activate the voice communications.
- Solution 2 will receive the voice communications.
- Set hardware jumpers and configuration as specified in the Getting starting section.
- The headphone/microphone sets are required for voice communications.
  - See the Voice Codec board datasheet for connection information
- Configure the Codec boards as detailed in the Codec board test procedure.

### 11.2.5 Exercise information

The Audio command commands are:

- `AT+BTA1` – Open Audio channel.
- `AT+BTA0` – Close Audio channel.

For audio communications the initiating device needs to set the receiving device to Local command mode (^^^ - See Exercise 6).

### 11.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- Principles of Audio communications.
- Opening the Audio channel.
- Closing the Audio channel.

### 11.2.7 Further work



- The basic program outlined above may work well for direct voice communications. However it has no way to turn the audio on or off. Use two switches to function as Audio On and Audio Off switches allowing voice communications to be controlled by the microcontroller.
- The code examples discussed assume that the audio channel is functioning and is enabled or disabled correctly via the `AT+BTA<n>` commands. This may not always be the case. An option for advanced users is to extend the program to include error checking. The `WaitForResponse` macro can be used to wait for `response_type` `AUDIO`, but does not detail if it is `FAIL ON` or `OFF`. Testing can be done by waiting for an `AUDIO` response. If the response is other than `AUDIO` and error has occurred and needs to be dealt with. Check the rest of the response to ensure that an `ON (AUDIO ON)` was received and the audio channel is enabled. If `AUDIO OFF` or `AUDIO FAIL` are received report the problem to the user on the LCD.

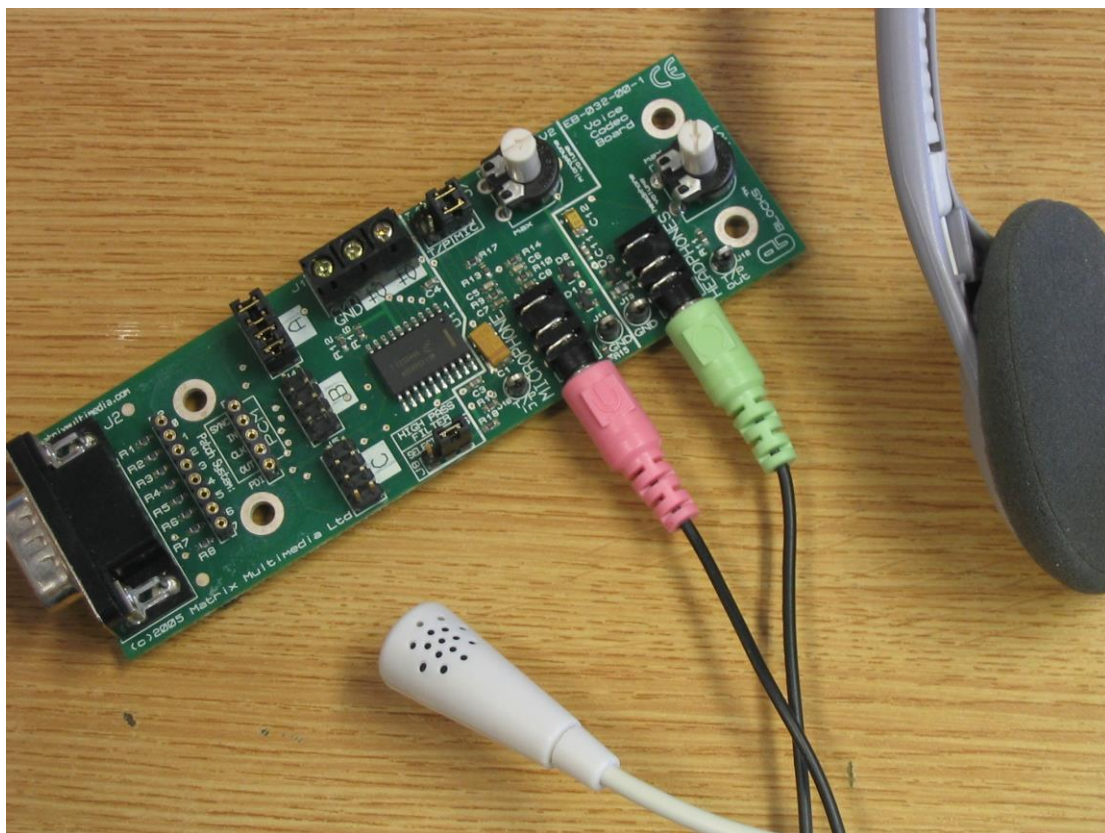
## 11.3 Practical implementation: Audio communications

### 11.3.1 Audio setup and considerations

The Voice Codec boards and headphones/microphones are required. Connect the Voice Codec boards to the Bluetooth boards using the Voice Codec documentation, and the notes from the Getting started section, to ensure that the boards are connected and configured correctly.

The combined headphone and microphone sets supplied with the Bluetooth solutions use industry standard 3.5mm stereo jack plugs. The jack plugs are marked with standard

Headphone  and Microphone  icons. They are also colour coded to match the standard connection colours of Green for headphones and Pink for Microphones. The jack plugs need to be firmly pushed in until they can be pushed no further. Use the volume controls to set the volumes for the headphones and microphone. The T/P MIC jumper needs to be in MIC for the headphone and microphone to be used. Note that mono jack plugs use the headphone slot.



### 11.3.2 Solo audio testing

Whilst two people allows voice communications to be tested using both devices in the same way as a regular phone call there is a trick to enable one person to test voice communications. Something that is useful for home study, or testing. On one board use the standard set up with a set of Headphones and microphone. On the other board put the T/P MIC jumper in T/P (Test Point mode). Connect the Headphones T/P OUT test point to the Microphone T/P IN test point. Then when an audio signal is sent (i.e. you speak into the microphone) it is received at the other end, output through the headphones test point and back in through the microphone test point to be sent back to the first board and received in the Headphones.

### 11.3.3 Using headphones/microphones and the Codec board

Before you start to develop programs please make sure that the volume potentiometers are turned up to maximum and that the headphone / microphone jacks are in the correct sockets.

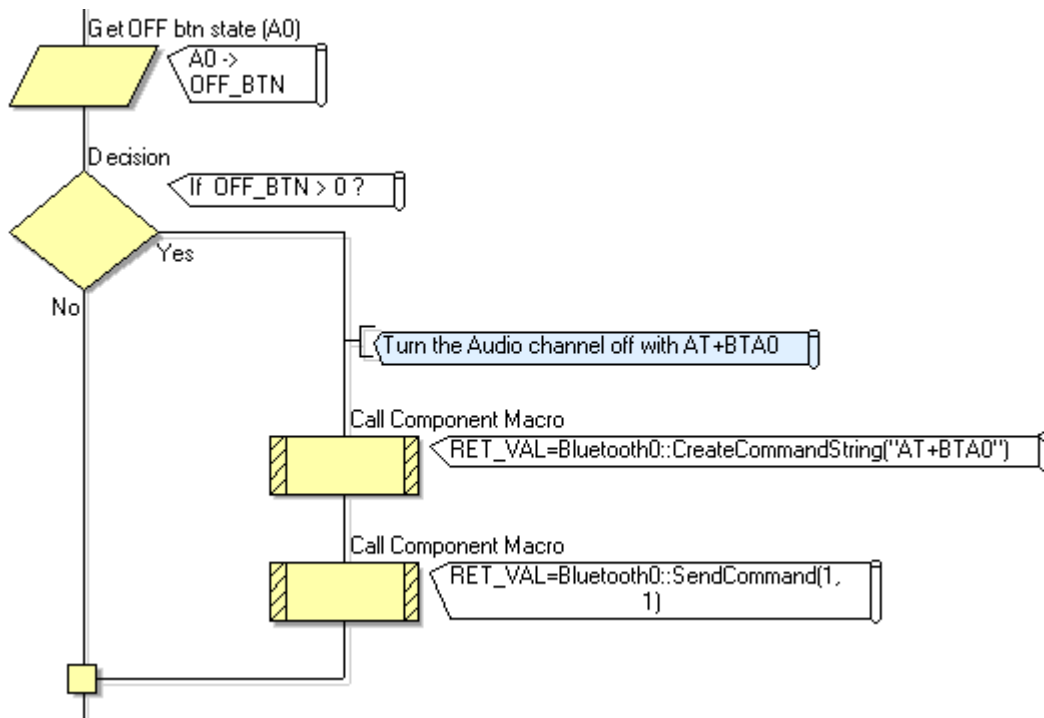
### 11.3.4 Program notes

Other than the audio board set up there is nothing in this exercise that has not been covered before. Therefore the exercise should be relatively straightforward.

The Data and Command mode programs can be used as basis for the programs required for this exercise.

Establish a connection; enter Local command mode and send the AT+BTA1 command to enable the audio channel.

For this example use Switch A0 on port A as a signal to end the audio communications. Monitor Switch A0 and send the Audio channel off command AT+BTA0 when it has been pressed.





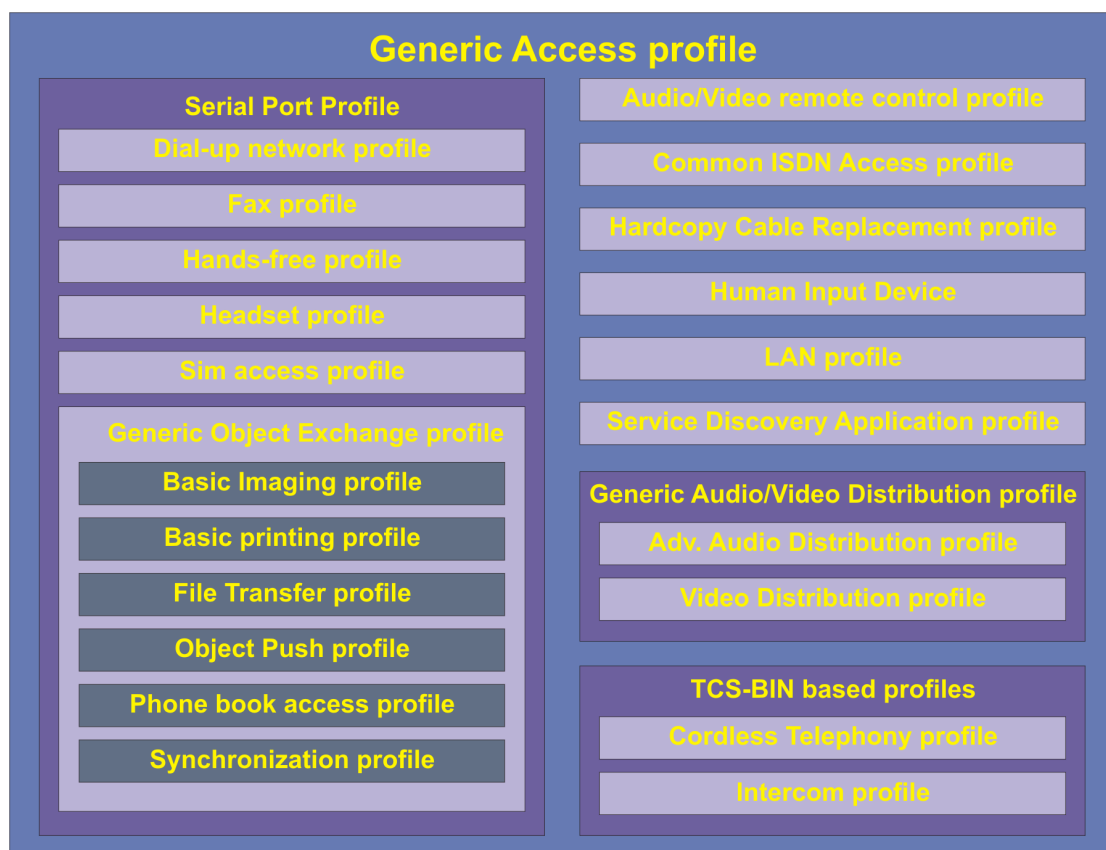
## 12 Profiles – Headsets and Telephones

### 12.1 Theory: Profiles

Bluetooth devices can use Profiles. Profiles are a specified implementation of Bluetooth with known features and functions. Declaring a Bluetooth device as conforming to a Profile allows the device to implement the profile specification, meaning that it will operate in a standardized manner. This in turn allows other devices to communicate with it in this standardized manner.

The Profiles are based around the requirements of a standard device for accomplishing a set task. The Headset profile for instance is based on an Audio device with a volume control and a command button. Standardized commands, features and functions allow devices to communicate to any Bluetooth headset in the same manner. It does not matter who made the headset, and what chip or other hardware they used for it, it is a headset and so can be accessed as one.

Profiles are layered so that common shared features are implemented in a lower layer of the profile system. This allows for commonality amongst devices easing design and use. For instance the Camera is part of the Basic Imaging Profile so will be expected to implement the Basic Imaging Profile specification elements. It is also a part of the Generic Object Exchange profile, so will need to implement that specification also. The next layer down is the Serial Port profile, which too will need implementing. Finally all the profiles are part of the Generic Access profile which has elements which also need implementing.



#### 12.1.1 Profile specific features

Part of the profile implementation includes profile specific features. The exact nature and details of these can be found in the Profile specific documentation. The headset profile document is included on the Bluetooth Solution CD to show the kind of profile specific features available.



For example, the Headset profile provides for the implementation of microphone and speaker volume control. In the case of the Headset several profile specific AT commands are specified for the implementation:

- AT+VGS=<n> Sets the Headset Speaker gain to <n> (<n> = 0 to15)
- AT+VGM=<n> Sets the Headset Microphone gain to <n> (<n> = 0 to15)
- AT+CKPD=200 Command used to indicate Headset Command button has been activated.

A copy of the Headset profile document is available on the accompanying CD in the Datasheets folder. The document contains details of the headset profile implementation including the additional AT commands available for the headset.

These additional commands aid in configuring and using the headset. Their precise implementation may differ between Headsets, but all Headsets are required to implement them. By this we mean that a headset manufacturer can choose to not have a Headset Microphone gain control on the physical hardware, but he must provide functions that can receive and handle the AT+VGM signal, if only to respond with an OK and ignore it otherwise. This allows other Bluetooth devices to be able to use the profile specific features knowing that which Headset it is and how it handles it does not matter.

### 12.1.2 Device class

The profile that a device is implementing is set in the devices Device class. The Device Class informs inquiring devices what classes the device conforms to. This information can then be used to decide if further communications and a possible connection are needed.

### 12.1.3 Major and Minor Device Classes

Profiles allow the device to specify major and minor classes that the device belongs to. Examples of major classes include:

- Miscellaneous
- Computer (Desktop, Laptop, PDA etc.)
- Phone
- LAN and Network Access Point
- Audio (headsets, speakers, stereos etc.)
- Peripherals (Mouse, Keyboard Joystick etc.)
- Imaging (Printer, scanner, camera etc.)
- Unclassified

The number and type of minor classes depends on the major class.

Examples for the Audio major class include:

- Hands-free
- Headset
- Loud speaker
- Head phones
- Car audio
- Camcorder

### 12.1.4 Device class values

The Device Class details are recorded in a 6 hexadecimal character value. The Device Class number is created from adding together various bits of a binary pattern from various tables for the separate parts. The creation of Device Class codes is quite complicated as a number of tables are required. (Details can be found in the Bluetooth specification documents.)

Example major device codes include:

- |                             |    |
|-----------------------------|----|
| • Miscellaneous             | 00 |
| • Computer                  | 01 |
| • Phone                     | 02 |
| • LAN /Network Access point | 03 |

- Audio/Video 04
- Peripheral 05
- Imaging 06
- Wearable 07
- Toy 08
- Unclassified 1F

Examples of full Device class codes are:

- Headset 200404
- Loudspeaker 200414
- Mobile phone 400204
- Camera 080520

### 12.1.5 Setting the Device Class

The AT Command AT+BTC<devclass> is used to set the Device Class.

The AT Command AT+BTC? is used to retrieve the Device Class value.

The value for the Device Class is stored in S Register 515.

The default value for the Device Class on the BLU2i chip is 001F00 – Unclassified.

Further details on the AT+BTC commands can be found in the AT Command Set document.

To set the device Class to a Headset for instance you would use the command:

AT+BTC200404

### 12.1.6 ATD and the <uuid> parameter

In addition to the Device Class the UUID parameter also requires to be set. The UUID parameter is used with the Instigate Connection command ATD to help form the search parameters for devices of that type.

The full description of the Instigate Connection command ATD is:

ATD<U><Y><bd\_addr>,<uuid>

<U> and <Y> are authentication and encryption parameters, which we shall be discussing next chapter.

<bd\_addr> is the device address, which has been covered previously.

<uuid> is the Universally Unique Identifier parameter used to select which class of device to connect to. If not specified the value stored in S Register 101 is used (default is: 1101 – Serial Port profile device).

The traditional UUID parameter used in programming is 128 bits long, but is shortened in use with Bluetooth to just 32 bits - 4 hexadecimal characters long. The following is a list of some useful UUID values:

- Serial Port 1101 (Default)
- LAN Access Using PPP (Point-to-Point Protocol) 1102
- Dialup Networking 1103
- IrMC Sync 1104
- OBEX Object Push 1105
- OBEX File Transfer 1106
- IrMC Sync Command 1107
- Headset 1108
- Cordless Telephony 1109
- Intercom 1110
- Fax 1111
- Audio Gateway 1112
- WAP 1113
- WAP\_CLIENT 1114

To set the UUID parameter use the ATS<n>=<m> command.

For example to set the UUID for an Intercom the command ATS101=1110 would be used.

## 12.2 Exercise 8: Headset profile

### 12.2.1 Introduction

A prime use of Bluetooth technologies is in telecommunications. For many Bluetooth is synonymous with the Mobile Phone and the Bluetooth headset. In this Exercise we shall look at the second of these devices - the Headset.

### 12.2.2 Objectives

- Develop a program which implements the Headset profile and sets up a communication between two audio enabled Bluetooth systems.

### 12.2.3 Pre-requisites

- An understanding of the Pairing process as detailed in Exercise 4.
- An understanding of Bluetooth Audio communications as detailed in Exercise 7.

### 12.2.4 Hardware/Software requirements

The following items of hardware are required:

- Both Bluetooth solutions are required for this Exercise.
- Solution 1 will be the phone part of the system.
- Solution 2 will be the headset part of the system.
- Set hardware jumpers and configuration as specified in the Getting starting section.

### 12.2.5 Exercise information

The headset related commands are:

- AT+BTI<devclass> to filter the Inquiry for specific Device Classes
- AT+BTC<devclass> to set the Device Class
- ATS101=<uuid> to set UUID value.
- AT+VGS=<n> Headset specific Speaker gain command <n> = 0-15
- AT+VGM=<n> Headset specific Microphone gain command <n> = 0-15
- AT+CKPD=200 Headset specific Command Button command

Configure one Bluetooth as a headset device.

Set up Switch A0 to be the Headset Command button.

Use the Headset button as an Answer/Hang-up button (enable/disable Audio channel).

Set up Switches A1 and A2 to be Headset speaker volume +/- controls respectively.

Use an initial Speaker volume level of 8.

Display the current Speaker volume on the LCD display.

### 12.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- An understanding of profiles.
- An understanding of the Headset profile.
- Filtering for particular profiles.

### 12.2.7 Further work

Use the keypad to develop a system that provides the functionality a two way intercom. You should use the keypad or switch module to develop functions for 'call', and 'terminate'. Use the LCDs on each system to show system status.

## 12.3 Practical implementation: Profiles

### 12.3.1 Implementing the Headset profile

A classic example of a profile is the Headset profile. Profiles are widely used so easily understood. The Headset is a relatively straightforward profile to implement. It has several additional AT commands that demonstrate how profiles can have profile specific command. Being AT commands the commands are easy to use with the BLU2i device. In general the Headset is well suited as a first example of profiles.

### 12.3.2 Setting up the headset

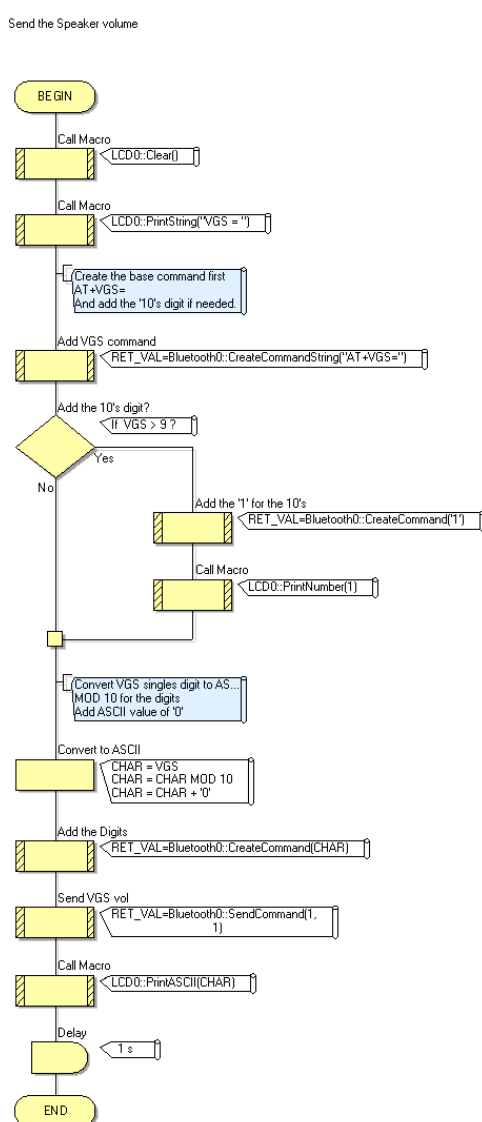
There are a number of settings to be implemented to set up a Headset profile. The <devclass> and <uuid> parameters both need setting. Also items such as how many rings before answering may need setting, allowing users the options to answer via a handset, not the headset, or to check caller ID etc. on a mobile phone before deciding to answer.

The <devclass> is set via the AT+BTC<devclass> command. For the headset use <devclass> = 200404.

In a similar manner the <uuid> can be set in S Register 101. The Headset <uuid> is 1108.

An example script to make the headset device discoverable and connectable would be:

Send the Speaker volume



ATS0=10 (10 rings to answer)  
 ATS536=1 (Remote command mode enabled)  
 ATS512=4 (Connectable and Discoverable)  
 ATS101=1108 (set uuid to Headset)  
 AT+BTC200404 (set devclass to headset)  
 AT+BTK="1234" (set Passkey)  
 AT&W (store settings in NVM)  
 ATZ (reset)

### 12.3.3 Special commands for the headset profile

The Headset profile for instance has a number of extra commands that can be used that are particular to the headset profile.

AT+CKPD=200 - Headset button command. Note the constant value of 200 used for this particular button.

AT+VGS=<n> - Headset Speaker gain (i.e. volume control)

AT+VGM=<n> - Headset Microphone gain

The two volume controls have obvious functions, but the headset button control simply reports the button press. It is up to the users own code to check for and react to this command. One obvious use for the Headset button is as an Answer/End call button.

### 12.3.3 Sending profile related commands

The profile related commands are triggered programmatically. The program needs to monitor switches or sensors to detect the appropriate input and then send the corresponding command. For the Headset command button the command is already set, but for the two volume commands a value <n> is required.

Whilst the actual value of <n> can be stored as a basic BYTE variable the value needs to be converted to ASCII for sending. A simple method is to add the character '1' if needed (i.e. <n> is equal to or greater than 10) and to add the ASCII value of the 0-9 digit.

### 12.3.4 Checking for profile related commands

Checking for profile related commands is relatively straightforward. Use the StringReceive command to get the initial message and its length. StringRead can then be used get characters for checking. If characters 3-6 of the message are CKPD, VGM= or VGS= then one of the Headset commands has been sent. If VGS or VGM the remaining characters are the value for <n> and will need to be converted from ASCII to a numeric BYTE value. The length of the message (as returned by StringReceive) will show if <n> is one or two digits.

### 12.3.5 Filtering for particular devices

In order to narrow down an Inquiry command search to only those devices that are required, such as when searching for any Headsets the optional <devclass> command can be used as a filter mechanism. When the Inquiry is sent only those devices that match the filter will respond. I.e. a Camera will ignore an Inquiry sent with a Headset parameter even though it is in range and discoverable.

<devclass> is an optional parameter to the AT\_BTI command. It can be either 2 or 6 hexadecimal characters in length. The full 6 characters are used to filter for devices that match that specific device class. If a particular class such as headsets or cameras is required then the full device class for the required device class can be used. If a major class is required, such as Audio or Computer, then the 2 character Major device class value can be used instead. Any device that corresponds to the major class will respond. For instance if an Audio class inquiry was sent not only would Headsets respond, but also other Audio devices including Speakers, microphones and Camcorders etc.

#### Examples:

To send an Inquiry command to find Headsets profile devices the <devclass> parameter would be set to: 200404 the Device class value for Headsets. E.g. AT+BTI200404

To send an Inquiry looking for any Audio based profile device, including Headsets and other Audio devices the <devclass> parameter would be set to: 04. E.g. AT+BTI04

## 13 Trust and Security

### 13.1 Theory: Trust and Security

Radio communications are inherently vulnerable. Any device capable of receiving radio signal from the same frequency bands can listen in to the signal. Radio communications companies have over the years put considerable effort into ways to send signals securely. And just as much effort has been devoted at times to breaking that security. Enigma and Bletchley Park may be the two most famous names from the age of communications warfare, but their descendants still live on. Sadly the most significant security threat is not from Allied intelligence, but from the more prosaic but insidious hackers and virus writers that plague the internet. Also whilst the French resistance may not be monitoring your signals maybe your business rivals would like to? Details of contacts, deals, offers and prices are priceless ammunition to the cut and thrust of the board room, and the humble phone, with its array of Bluetooth add-ons, is right in the front line.

#### 13.1.1 Security in general

So how does Bluetooth go about ensuring that you are secure?

There are three complementary systems in place.

- Hardware
- PIN numbers
- Authentication, Encryption and Trust

At the hardware level Bluetooth employs frequency hopping techniques that leap from frequency to frequency. Listening to one frequency will get you just minute parts of the message. Trying to listen into a full conversation requires you to know the frequencies to hop too, information that is not shared to those devices not included in the communication.

PIN numbers are often used by devices such as Mobile phones and Headsets etc. The PIN number is supplied with a device and is required to be entered to be checked by the program. Note: This is not the same as the Passkey that is passed programmatically, but a User entered PIN Number. PIN numbers are generally used to inform the device that you are the legitimate owner to help prevent thieves from using the device. PIN numbers should not be stored with the device for security reasons, just like you would not leave your PIN number with your bank cards.

At the program and device level the communications can be set to be Authenticated and or Encrypted, and the device can be recorded as a trusted device.

#### 13.1.2 Authentication

Once pairing has been complete the two devices can elect to use Authentication. If authentication is required any communication between the two devices may (depending on the authentication settings) provoke a response and counter response wherein a challenge key is sent and an appropriate response needs to be returned for the communications to be authenticated. Fortunately this all happens behind the scenes and the exact details need not concern us. What does concern us though is how to enable authentication.

The Authentication settings are held in two S Registers: S Register 500 for authentication on outgoing calls and S Register 502 for authentication on incoming calls. A value of 1 enables authentication, and a value of 0 disables it. Both authentication registers are set to 0 (disabled) by default.

To set the registers use the `ATS<n>=<m>` command.

For instance `ATS500=1` enables Authentication on outgoing calls.

| S Register | Description   | Default value |
|------------|---|---------------|
| 500        | Authentication mode for outgoing calls. Set to 1 to enable. | 0             |
| 501        | Encryption mode for outgoing calls. Set to 1 to enable.     | 0             |
| 502        | Authentication mode for incoming calls. Set to 1 to enable. | 0             |
| 503        | Encryption mode for incoming calls. Set to 1 to enable.     | 0             |



### 13.1.3 Encryption

If at least one device requires authentication the communications can also be encrypted. The two devices work out the encryption details between them. Once again fortunately behind the scenes so we don't need to worry about it. Encryption is handled in the same way as Authentication, the only difference being that the two registers involved are. S Register 501 for encryption on outgoing calls and S Register 503 for encryption on incoming calls.

### 13.1.4 ATD and Authentication and Encryption

As mentioned last chapter the full ATD command is:

ATD<U><Y><bd\_addr>,<uuid>

<U> and <Y> are authentication and encryption parameters and can be used instead of the default S Register values to enable authentication and encryption for a particular connection.

If <U> is Specified (ie. ATDU<bd\_addr>) then the connection will be authenticated. If <U> is not specified then the value of S Register 500 is used instead to determine if authentication is to take place.

If <Y> is Specified (ie. ATDU<bd\_addr>) then the connection will be encrypted. If <Y> is not specified then the value of S Register 501 is used instead to determine if encryption is to take place.

For example:

ATDUY008924800352,1108 - Instigate connection to the headset device 008924800352 with both authentication and encryption.

### 13.1.5 Trust

When two Bluetooth devices have instigated pairing and established communications between themselves they are said to be in a trusted pair. Once a trusted pair has been established they can communicate with each other without requiring discovery or authentication. As this will obviously speed up the connection and communication processes there are a number of commands and options available to control recording which devices you trust.

Trusted Devices are cached and stored in a list. The Trusted Devices List is stored in Non Volatile Memory allowing it to be retained during power off. The cache on the other hand is temporary listing only the last device to pair and gain trust.

### 13.1.6 Trusted Devices AT Commands

The following AT Commands are for use in adding to, removing from and inquiring about the Trusted Devices List:

AT+BTT - Add device to Trusted Devices list.

Adds the currently paired device in the Trusted devices list.

AT+BTT? - List trusted devices

Returns a list of the trusted devices by sending a series of responses comprising the addresses of trusted devices and an OK device once the list has been sent. e.g.

012345678912

678901234567

OK

AT+BTW? – list current cached trusted device.

Returns the address of the device currently in the trusted device cache. Generally the device currently paired to but not necessarily depending on the state of pairing.

AT+BTD<bd\_addr> - remove device <bd\_addr> from the Trusted Devices list.

Returns OK if successful. If device <bd\_addr> is not in the list an OK will still be returned.



AT+BTDP\* - remove all Trusted Devices.

This command clears all the devices from the Trusted Devices list. No confirmation is asked for, so use with caution. Using this command can cause authentication problems with currently paired devices so it is highly recommended to send an ATZ Reset command after using AT+BTDP\* to prevent any communications problems.

## 13.2 Exercise 9: Trust and Security

### 13.2.1 Introduction

Good communication is both secure and efficient. Security is required to block listeners and prevent bogus communications. However security must not be too intrusive, and devices that are known to be trusted need to be able to communicate without too much interference from the virtual security guards.

### 13.2.2 Objectives

Create a program that pairs with another Bluetooth device.  
Ensure that the connection is both Authenticated and Encrypted.

Give the device three options controlled by Switches on Port A:

- 1) List devices currently in the Trusted Devices List
- 2) Add the current device into the Trusted Devices List
- 3) Remove the current device from the Trusted Devices List.

Use the program to demonstrate use of the Trusted Devices List.

### 13.2.3 Pre-requisites

- An understanding of the Pairing process as detailed in Exercise 4.
- An understanding of Responses as detailed in Exercise 5.

### 13.2.4 Hardware/Software requirements

The following items of hardware are required:

- Both Bluetooth solutions are required for this Exercise.
- Solution 1 will be used to implement to objectives.
- Solution 2 will be used as the device to add or remove from the Trusted Devices list.

### 13.2.5 Exercise information

The Audio command commands are:

- `AT+BTT` – Add Device to Trusted Devices List.
- `AT+BTD<bd_addr>` – Remove Device from Trusted Devices List.
- `AT+BTT?` – List Devices on the Trusted Devices List.

For details of the AT Commands please see the AT Commands Set document.

### 13.2.6 Learning outcome

Primary learning outcomes for this exercise are:

- Principles of Security including:
  - Authentication
  - Encryption
- Using the Trusted Devices list including:
  - Adding Devices
  - Removing Devices
  - Listing Devices

### 13.2.7 Further work

If you can get access to a Bluetooth phone, use your experience gained so far to determine the address of the Bluetooth module, and set up a two way audio link with the mobile phone using the Headset profile.

## 13.3 Practical implementation: Trust and Security

### 13.3.1 General objectives

The general program resembles the Inquiry program created in the initial Discovery exercise chapter. The main differences being that we are listing the Trusted Device list, not the Inquiry responses. Tack on a Pairing at the start, which should be a familiar exercise by now, and we have the majority of the program already sorted.

Authentication and Encryption are called for. These can be set using the S Registers. However it be worth using ATDUIY<bd\_addr> as well to demonstrate both this method and to make the point that S Registers can be changed programmatically unlike ATD<U><Y>.

The main loop needs to check the switches and respond to any inputs, sending the appropriate AT command as needed. Once again this should be a relatively simple task at this stage.

For this exercise use Switch A0 as 'Add Device', Switch A1 as 'Remove Device', and Switch A2 as 'List Trusted Devices'.

The commands that need to be sent are:

| Switch    | Command         | Description                             |
|-----------|-----------------|---|
| Switch A0 | AT+BTT          | Add device to Trusted Device List       |
| Switch A1 | AT+BTD<bd_addr> | Remove device from Trusted Device List  |
| Switch A2 | AT+BTT?         | List Devices in the Trusted Device List |

A simple way of determining if the Add and Remove Device from the Trusted Device List worked is to list the devices and look for the device address.

The LCD can be used to show not only the Trusted Device List, but to show status messages for the other options.

### 13.3.2 Additional features

As an optional extra the base program can be extended to clear the Trusted Devices list, and to display the current trusted device cache.

## 14 Project design principles

Once students have become familiar with Bluetooth and have completed the exercises in this course the next stage is to use that knowledge to design and implement a project.

The following two projects are examples of simple systems that can be designed with Bluetooth.

Projects are for students to design and implement. The two examples here are just that: examples, not a definitive list. It is always best for students to decide what project to develop, get approval for the project, and to discuss the design implications before starting the project.

Design considerations are discussed below, but no code or instructions are given. They can be used as a starting point for creating the project objectives and specifications.

### 14.1 Project - Baby monitor

A baby monitor is basically an audio system that has a transmitting device with a microphone (the monitor by the baby), and a receiving device (the monitor by the parents with a speaker output).

The basic blocks are two Bluetooth devices paired in a network. One functions as a microphone monitoring the baby, whereas the other is a speaker transmitting the noises to the parents.

Pairing will be required as otherwise you could be picking up signals from other baby monitors and missing the one you actually need.

To accomplish the project you will need to pair the devices. Establish one as an audio microphone and one as an audio speaker. The basic baby monitor program will be relatively simple as it does not require much more than direct audio communications. To make the project more effective, you should consider implementing additional features.

#### 14.1.1 Extending the project with additional features

Consider ways to extend or improve the Baby monitor consider items such as the following:

- **Multiple transmitters and receivers**

Do you have more than one child to monitor? If so would multiple monitors be useful? Will you be in the same room all the time or will you be popping back and forth between the kitchen and living room? Maybe multiple receivers would be a good idea. How to do this though? Will it be a set number of stations with set ID's? Or will you need to search for and discover the devices?

- **2-way communication**

Do you want to talk to baby? Maybe sing her a lullaby or just let them know you are near

What you need is a two-way system. However you don't want the baby listening to you, or being disturbed by the movie you are watching. So you will need to be selective in how to implement the signal to baby's monitor. Whilst this may seem simple it involves coordinating the audio signals maybe across multiple receivers.

- **Volume indicator**

It would be bad if baby was crying her eyes out but you failed to realize because the monitor was too quiet. Also what about your dinner guests getting annoyed because they can hear every gurgle the baby makes? Does it need turning down a bit? Maybe a volume control and indicator would help?

Tasks to consider include implementing an analogue dial or digital Up/Down buttons for the volume, how to handle default just turned on settings if digital and working out how to display a volume bar rather than a count.

## 14.2 Project - Medical datalogger

One useful task for Bluetooth would be medical datalogging. Patients need monitoring but wires and plugs tie the patient to the bed and create extra work for staff. Should a patient need to go for tests, or even to the bathroom then equipment needs unplugged and moved, or it needs to be mobile. And what is more mobile than Bluetooth? For patients, the ability to move around gives them greater freedom and independence. For staff the system would provide 24 hour datalogging, and ease of patient movement. Also data alerting can be used allowing staff to be alerted to problems as they occur.

Such dataloggers would also aid in medical tests, such as in sports related datalogging where traditional wired sensors would just get in the way.

The first thing to consider for the remote datalogger is what data do you wish to log? For a simple datalogger temperature could be monitored. Alternatively you could add other sensors to the mix such as heart rate and respiration monitors.

Next you need to consider how frequently the data needs to be updated. For something like temperature it could be every 5 or 10 minutes. For something more critical such as monitoring heart rates you may need readings every few seconds. After deciding how frequently to send data you need to decide upon a strategy for sending the data. Keep continuous contact? Contact and send the data? Collect data for a while, and then batch send the data?

Next, are there any special cases to consider? Is a sudden rise in the heart rate important? What about temperature? Should we be warned with flashing lights and alarm bells? Should it set our beepers ringing so we know there is a problem. Or is a simple LED alert and a message on an LCD display enough for us? Do we need to know now or in 5 minutes at the next scheduled data broadcast?

### 14.2.1 Extending the project

Two obvious problems exist with the medical monitors compared to fixed monitors that restrict movement.

- What happens when the patient walks out of network range?
- Where is the patient when the alarm goes off?

How can these problems be addressed? Can they be addressed?

Could layered networks be the key?

One network set up with alarm lights to indicate a patient in trouble nearby?

Security networks at the exits to warn patients they are leaving the network area?

Maybe additional technologies could be brought into play such as GPS to track patients?

Some of these issues are too big for addressing in a project, but would show how they have considered if written up by the student in an 'Extended/Additional features' section of the project report.

## Part 3: Reference and Appendix

### 15 References and Appendix

Useful books and online links to aid in investigating and implementing Bluetooth.

#### 15.1 References

##### 15.1.1 Books

|                  |  |      |               |
|------------------|--|------|---------------|
| Morrow, Robert.  | Bluetooth operation and use              | 2002 | McGraw Hill   |
| Gratton, Dean A. | Bluetooth Profiles: The definitive guide | 2002 | Prentice Hall |

##### 15.1.2 Websites

|                               |   |
|-------------------------------|---|
| Bluetooth official site:      | <a href="http://www.bluetooth.com/">http://www.bluetooth.com/</a>                           |
| Wikipedia Bluetooth entry:    | <a href="http://en.wikipedia.org/wiki/Bluetooth">http://en.wikipedia.org/wiki/Bluetooth</a> |
| EZURiO BLU2i Device web site: | <a href="http://www.ezurio.com/">http://www.ezurio.com/</a>                                 |

#### AT command reference

##### 15.1.3 Common AT Command parameters

|             |   |
|-------------|---|
| <bd_addr>   | 12 character hexadecimal Bluetooth address.   |
| <dev_class> | 6 character hexadecimal Bluetooth device class code.  |
| <uuid>      | 4 character hexadecimal Bluetooth UUID code.<br>Common <uuid> values include:<br>Serial Port: 1101<br>Headset: 1108<br>Cordless telephone: 1109<br>Fax 1111 |
| <U>         | Authentication method. Value of S Register 500 if not specified.  |
| <Y>         | Encryption method. Value of S Register 501 if not specified.  |
| <n>         | Positive integer value  |
| <m>         | Integer value that can be positive or negative. Hexadecimal numbers can be used with a \$ prefix. E.g. \$4DE3   |

##### 15.1.4 Important AT Commands list

| AT Command | Description  | Response                                  |
|------------|--|---|
| ^^^        | Enter local Command mode   | OK  |
| !!!        | Enter Remote Command mode  | OK  |
| AT         | Test command   | OK  |
| ATS<n>=xxx | Set S Register 'n' to value 'xxx'  | Value of S Register <n>                   |
| ATS<n>?    | Request value of S Register <n>  | Value of S Register <n> or ERROR <string> |
| ATI<n>     | Request information from the device. Generally used to retrieve the <bd_addr> and S register values.<br><br>Useful values for <n> include: | Information or OK as appropriate.         |

|                           |  |  |
|---------------------------|--|--|
|                           | 0 – Product name/variant<br>4 – <bd_addr> of the device<br>5 – Manufacturer name   |  |
| ATD<U><Y><bd_addr>,<uuid> | Connect to the device with the address <bd_addr>. The other options add options such as Encrypted, Authenticated or specify profile types to connect to <uuid> | CONNECT<br><bd_addr> AE<br>A= Authenticated<br>E = Encrypted |
| ATH                       | Disconnect. Used to end communications for Audio signals and data sending.   | -  |
| ATZ                       | Reset the Device   | OK   |
| AT&W                      | Save all S registers to Non Volatile Memory so that they are preserved.  | OK   |
| AT+BTA<n>                 | Control the Audio channel.<br>0 = off<br>1 = on  | OK   |
| AT+BTK=<string>           | Set Passkey to <string>.<br><string> is a 0-8 numeric character string.<br><string> length 0 deletes the current Passkey.                                      | OK   |
| AT+BTW<bd_addr>           | Initiate pairing with device <bd_addr>   | OK   |
| ATO                       | Note: letter 'O' not number 0.<br>Return to data mode  | OK   |
| AT+BTI                    | Inquiry for device class code.   | <string>   |
|                           |  |  |

#### 15.1.5 Important S registers

Given how important the S Register settings are for the BLU2i device we have listed a number of important S registers and their default values here. A full list can be found in the AT Commands document on the CD supplied with the Bluetooth solution.

| S Register | Description   | Default value |
|------------|---|---------------|
| 0          | Number of rings before answering  | 1             |
| 101        | Default <uuid> value  | \$1101        |
| 500        | Authentication mode for outgoing calls. Set to 1 to enable.   | 0             |
| 501        | Encryption mode for outgoing calls. Set to 1 to enable.   | 0             |
| 502        | Authentication mode for incoming calls. Set to 1 to enable.   | 0             |
| 503        | Encryption mode for incoming calls. Set to 1 to enable.   | 0             |
| 505        | Delay before abandoning connection attempt. Used in conjunction with the ATD command.   | 5             |
| 512        | Specify power up state.<br>Value 0-7 sets state. See AT Commands document for details.<br>Examples:<br>3 – connectable but not discoverable<br>4 – connectable and discoverable | 1             |
| 536        | Enable Remote Capture. Used for the !!! command<br>0 = Remote capture disabled.<br>1 = Remote capture enabled.  | 0             |



## 15.2 Appendix

### 15.2.1 How to...

#### Inquiry Command

Command:

Inquiry command

AT+BTI

Response:

List of device addresses of discoverable devices followed by OK

012345678912

012343567778

OK

#### Setting Discoverability (General Initialization commands)

Command:

Initialization Script

ATS0=1

ATS512=4

ATS536=1

AT&W

ATZ

#### Initiate Pairing and connection

Command:

Passkey command

AT+BTK="1234"

Response

OK

Command:

Initiate pairing command

AT+BTW0123456789012

Response

OK

PAIR 0 012345678912

Command:

Initiate connection command

ATD0123456789012

Response

CONNECT 012345678912

#### Remote and local command modes

Remote command mode = !!!

Local command mode = ^^

Sequence is sent with guard gaps:

<guard time><Esc char><guard time>< Esc char ><guard time>< Esc char ><guard time>

#### Audio commands

AT+BTA0 = Audio channel disabled

AT+BTA1 = Audio channel enabled

Unsolicited Response sent of:

AUDIO <string>

Where <string> is ON, OFF or FAIL indicating current Audio channel status.

#### Trusted devices

AT+BTT – Add to trusted devices list

Response

OK or ERROR

AT+BTT? – List trusted devices

Response is Device address list of Trusted devices followed by OK

01234566789012

65478990002020

OK

### Authentication and Encryption

ATD<U><Y><bd\_addr>,<uuid>

Where:

U (ATDU...) enables Authentication.

Y (ATDY...) enables Encryption.

### Implementing profiles (Headset example)

Set Device Class and UUID to match profile.

ATS101=<uuid>

AT+BTC<dev\_class>

Headset example:

ATS101=1108

AT+BTC200404

Note additional profile related commands.

Headset example:

AT+VGS=<n> - Speaker volume (<n> = 0-15).

AT+VGM=<n> - Microphone volume (<n> = 0-15).

AT+CKPD=200 – Headset control button event.