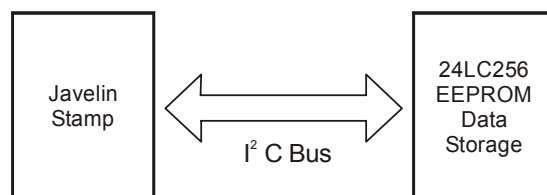


599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office/Tech Support: (916) 624-8333
Fax: (916) 624-8003

Web Site: www.javelinstamp.com
Home Page: www.parallaxinc.com

General: info@parallaxinc.com
Sales: sales@parallaxinc.com
Technical: javelintech@parallaxinc.com



Contents

Introduction to Microchip's 24LC256 I ² C Serial EEPROM.....	1
Downloads, Parts, and Equipment for the MC24LC256.....	1
MC24LC256 Example Circuit.....	3
Testing the 24LC256 Circuit.....	4
Program Listing 1.1 – The MC24LC256Test.java.....	6
MC24LC256 Data Storage Utility.....	7
Program Listing 1.2 – MC24LC256DataStorageUtility.java.....	8
MC24LC256 Demonstration.....	11
Published Resources – for More Information.....	11
Javelin Stamp Discussion Forum – Questions and Answers.....	11

Introduction to Microchip's 24LC256 I²C Serial EEPROM

The 24LC256 is a 32 kb EEPROM which includes an I²C compatible 2-wire bus. Three address lines allow for up to 8 chips on a single bus. The MC24LC256 library class will allow you to read and write directly to the 24LC256 EEPROM(s).

Downloads, Parts, and Equipment for the MC24LC256

This application note, the library file, the javadoc file, the test program (Program Listing 1.1), the demonstration program (which will demonstrate all the methods available to you in the MC24LC256 library), and an application example (Program Listing 1.2) are all available to you for free download from:

<http://www.javelinstamp.com/Applications.htm>

You can use the AppNote005-MC24LC256.exe, to install the files listed below. These files must be located in specific paths within the Javelin Stamp IDE directory. Although the path to this directory can be different, the default root path is: C:\Program Files\Parallax Inc\Javelin Stamp IDE

```
<root path>\doc\AppNote005-MC24LC256.pdf
<root path>\doc\MC24LC256.pdf
<root path>\lib\stamp\peripheral\memory\eprom\MC24LC256.java
<root path>\lib\stamp\peripheral\memory\eprom\MC24LC256BadReadException.java
<root path>\Projects\examples\peripheral\memory\eprom\MC24LC256DataStorageUtility.java
<root path>\Projects\examples\peripheral\memory\eprom\MC24LC256Demo.java
<root path>\Projects\examples\peripheral\memory\eprom\MC24LC256Test.java
```

This library relies on the I²C protocol, you will need to have the I2C.java library class file installed on your computer. This file can be obtained from:

<http://www.javelinstamp.com/Applications.htm>

The file should then be placed in the following directory:

```
<root path>\lib\stamp\protocol\I2C.java
```

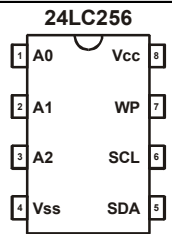

**Important for I2C.java
versions 1.1 and below**

*You will need to change the package information within the class.
This information can be found at the top of the code.*

Change this line: **package stamp.peripheral;**
To this line: **package stamp.protocol;**

Table 1.1 lists the parts you will need for this application note. You can use up to eight individual 24LC256 chips.

Table 1.1: Parts List

Quantity	Part Ordering Info and Part Description	Figure
1	Microchip IC SERIAL EEPROM 32 k x 8 2.5V 8-pin DIP Digi-Key Part #24LC256-I/P-ND	
2	Resistor 4.7k Ω $\frac{1}{4}$ W – 5% Parallax Part #150-04720	 Yellow, Purple, Red

The equipment used to test this example includes a Javelin Stamp, Javelin Stamp Demo Board, 7.5 V, 1000 mA DC power supply, serial cable, and PC with the Javelin Stamp IDE v2.01.

MC24LC256 Example Circuit

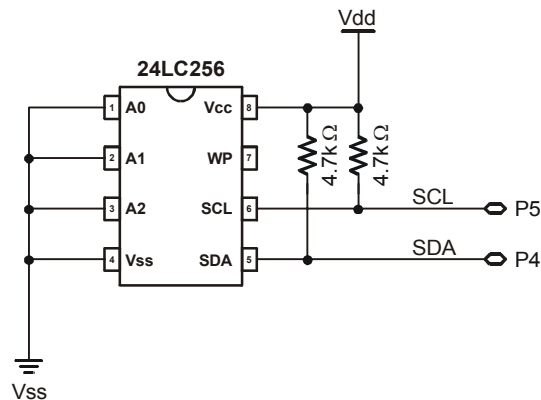
Figure 1.1 is a complete circuit for the 24LC256 EEPROM that can be used for all programs in this document. This circuit is wired as device number zero, and will communicate with the Javelin using the I2C protocol on the Javelins I/O pins P4 and P5. These two lines must each have a 4.7 k Ω pull-up resistor on them. The following list will describe the connection of each pin on the 24LC256 chip.

- The address pins A0, A1, and A2 are all connected to ground (Vss) on the Javelin for this device to be addressed as device zero. To change the address to other addresses from 0-7, you will need to connect to +5 v (Vdd). The address pins can be set as follows:

MC24LC256 Address pin map			
ID#	A2	A1	A0
0	0	0	0
1	0	0	+5
2	0	+5	0
3	0	+5	+5
4	+5	0	0
5	+5	0	+5
6	+5	+5	0
7	+5	+5	+5

- The ground pin (Vss) connects to the ground of the Javelin Stamp (Vss).
- The serial data line (SDA) connects to the Javelin's I/O pin P4. This line must also have a 4.7 kΩ pull-up resistor attached to Vdd on the Javelin Stamp.
- The serial clock line (SCL) connects to the Javelin's I/O pin P5. This line must also have a 4.7 kΩ pull-up resistor attached to Vdd on the Javelin Stamp.
- Do not connect the write protect pin (WP), it will not be needed for this application note.
- The power pin (Vcc) is connected to +5 v on the Javelin Stamp (Vdd)

Figure 1.1
Wiring diagram for the
MC24LC256 test circuit.



Tip

When connecting more than one 24LC256 chips tie together all the SDA lines. Then have one 4.7 kΩ pull-up resistor attaching the SDA line to Vdd. Also tie together all the SCL lines, and have one 4.7 kΩ pull-up resistor attaching the SCL line to Vdd.

Testing the 24LC256 Circuit

Once you have successfully wired your circuit as shown in Figure 1.1, you can program your stamp with Program Listing 1.1. This testing program will create an I²C bus with the SDA line connected to P4 and the SCL line connected to P5 (see code snippet below).

```
final int sdaPin = CPU.pin4;
final int sclPin = CPU.pin5;
MC24LC256 ee = new MC24LC256(sdaPin, sclPin);
```

Then it will write a single byte with the value of 99 to EEPROM address 20 of device number zero using the writeOne method. Notice this value must be typecast to a byte.

```
test = ee.writeOne(0,20,(byte)99);
```

The resulting true or false, indicates whether the operation was successful or not. The following will print the appropriate message to indicate a successful write.

```
if (!test) System.out.println("Write Failed, verify circuit");  
if (test) System.out.println("Write Successful");
```

Next, the program will attempt to read the EEPROM from the same address location. It does this within a try/catch routine so it can be sure to catch any errors.

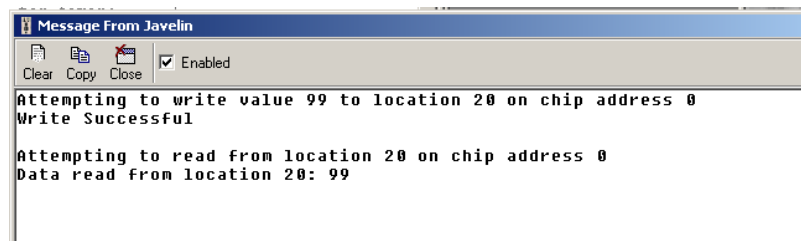
```
try {  
    num = ee.readRandom(0,20);  
    System.out.print("Data read from location 20: ");  
    System.out.println(num);  
} // end try  
  
catch (MC24LC256BadReadException bre) {  
    System.out.println("Bad Read Detected Here");  
} // end catch
```

**Tip: try and
catch**

The **readRandom** method was designed to **throw** back special information if an error occurred. This information tells the program to immediately execute the commands within the **catch**. Then, continue below the **catch** as normal. Therefore, the print lines in the **try** will never have been executed.

When testing, make sure your circuit matches Figure 1.1 exactly. Even if you are planning on connecting multiple chips, first get one chip working before adding more. When your circuit is ready you can program it with Program Listing 1.1 the output from this program will resemble Figure 1.2.

Figure 1.2
MC24LC256Test
output window



If you do not see the output window above (Figure 1.2), do the following:

- ✓ Verify your SDA and SCL lines are connected correctly
- ✓ Verify your A0, A1 and A2 address lines are all connected to ground (Vss)
- ✓ Verify you are using a 4.7 kΩ pull-up resistor on both the SDA and SCL lines.

Program Listing 1.1 – The MC24LC256Test.java

```
// Version 1.0
// This class will test the 24LC256 circuit from AppNote005
import stamp.peripheral.memory.eeprom.*;
import stamp.core.*;

//Test the 24LC256 circuit.
public class MC24LC256Test {
    public static void main() {
        final int sdaPin = CPU.pin4; // Javelin's I/O pin P4, used for data
        final int sclPin = CPU.pin5; // Javelin's I/O pin P5, used for timing
        MC24LC256 ee = new MC24LC256(sdaPin, sclPin); // Create MC24LC256 object
        boolean test = true; // validate
        int num; // value from eeprom

        // Write one byte of value 99 to memory location 20 on chip 0
        System.out.println("Attempting to write value 99 to location 20 on chip address 0");
        test = ee.writeOne(0,20,(byte) 99);
        if (!test) System.out.println("Write Failed, verify circuit");
        if (test) System.out.println("Write Successful");

        // Read from location 20 from chip 0 and test for errors.
        System.out.println("\nAttempting to read from location 20 on chip address 0");
        try {
            num = ee.readRandom(0,20);
            System.out.print("Data read from location 20: ");
            System.out.println(num);
        } // end try
        catch (MC24LC256BadReadException bre) {
            System.out.println("Bad Read Detected Here");
        } // end catch
    } // end main
} // end class
```

MC24LC256 Data Storage Utility

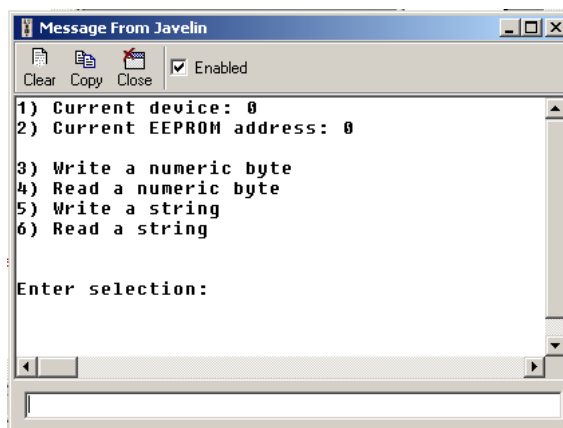
Program Listing 1.2 will allow you to read and write up to eight 24LC256 EEPROMs. The program uses many of the MC24LC256 library methods, and the code is broken up into sections that should make it easy for you to use and understand.

The program has a menu for you to interact with. The menu has six options numbered 1 through 6.

1. Menu item one will allow you to select a specific 24LC256 EEPROM device. The I²C bus allows you to address up to eight chips, these chips have addresses from 0 to 7. Notice the current device will be displayed.
2. Menu item two will allow you to set the current EEPROM address that you will be reading and writing from. Notice the current EEPROM address will be displayed.
3. Menu item three will allow you to write a single byte to the EEPROM. This byte must have a value between 0 and 255.
4. Menu item four will read this value.
5. Menu item five will write an ASCII text message to the EEPROM, you will not be allowed to write a message that exceeds address 32767.
6. Menu item six will read an ASCII text message from the EEPROM. Since there was no delimiter written to the EEPROM when you stored your message, you will need to enter the length of the message to be retrieved. If you attempt to read a value that is a non-printable character you will receive an error message for that character.

Figure 1.3 shows what Program Listing 1.2 output window should look like.

Figure 1.3
MC24LC256
Data Storage
Utility



If your output does not look like the output above (Figure 1.3), you should verify that your circuit matches Figure 1.1 exactly. And try re-running the test code (Program Listing 1.1) again.

Tip

Use the MC24LC256 javadoc file (MC24LC256.pdf located in your <"Javelin Stamp IDE\doc"> folder) as a reference to find out more about using the methods available to you.

This program uses the methods below from the MC24LC256 library class.

- **writeOne(dev, address, byte)**

Write a single byte to a specific address on a specific device.

- dev = device address (0-7)
- address = EEPROM address to write data to
- byte = byte value to be written (0-255)

- **readRandom(dev, address)**

Read a specific address from a specific device. The reason this method is called random is because with it you do not have to read from the EEPROM in a sequential fashion (like a tape drive), but you can read from the EEPROM in a random fashion (like a hard drive).

- dev = device address (0-7)
- address = EEPROM address to read data from

The **readRandom** method must be called within a try/catch function. This guarantees that the data read will be valid. The read itself should be placed inside the *try*, and if the data is in error the *try* will fail, the code within the *catch* will execute.

Program Listing 1.2 – MC24LC256DataStorageUtility.java

```
import stamp.core.*;
import stamp.peripheral.memory.eeprom.*;

/*
 * This utility will allow you to read and write to various 24LC256 chips.
 * You can use this class to better understand how to interact with this chip.
 * Version 1.0
 */
public class MC24LC256DataStorageUtility{
    static char keyboardChar;
    static StringBuffer keyboardMsg = new StringBuffer(30);

    public static void main() {
        final int sdaPin = CPU.pin4;           // Javelin's I/O pin P4, used for data
        final int sclPin = CPU.pin5;           // Javelin's I/O pin P5, used for timing
    }
}
```



```
final char CLS = '\u0010';          // Clear Screen Code
int dev = 0;                        // Device ID (default zero)
int address = 0;                    // EEPROM address to read/write
MC24LC256 ee = new MC24LC256(sdaPin, sclPin); // Create MC24LC256 object
boolean test;                       // validate read/write
int n=0,x=0;                        // misc

// Display menu
while (true){
    System.out.print(CLS);
    System.out.print("1) Current device: ");
    System.out.println(dev);
    System.out.print("2) Current EEPROM address: ");
    System.out.println(address);
    System.out.println("\n3) Write a numeric byte");
    System.out.println("4) Read a numeric byte");
    System.out.println("5) Write a string");
    System.out.println("6) Read a string");
    System.out.print("\n\nEnter selection: ");

    // Select current device
    switch (Terminal.getChar()){
        case '1':
            System.out.print("\nEnter device you wish to select: ");
            x=Terminal.getChar();
            x-=48;
            if ((x>=0)&&(x<=7)) dev=x;
            else {
                System.out.println("\nMust select a device between 0-7, device must be
currently connected");
                CPU.delay(30000);
            }//end else
            break;

        // Select current EEPROM address
        case '2':
            System.out.print("\nEnter EEPROM address (0-32767): ");
            test=true;
            x=getInt();
            if ((x>=0)&&(x<=32767)) address=x;
            else {
                System.out.println("\nAddress must be between 0-32767");
                CPU.delay(30000);
            }// end else
            break;

        // Write a byte
        case '3':
            System.out.print("\nEnter byte (0-255) to write to EEPROM: ");
            x=getInt();
            if ((x>=0)&&(x<=255)) ee.writeOne(dev,address,(byte)x);
            else {
```

```
        System.out.println("\nValue must be between 0-255");
        CPU.delay(30000);
    } // end else
    break;

    // Read a byte
    case '4':
        try {
            x = ee.readRandom(dev,address);
            System.out.print("\nData read from device ");
            System.out.print(dev);
            System.out.print(" at address ");
            System.out.print(address);
            System.out.print(" is: ");
            System.out.println(x);
        } // end try
        catch (MC24LC256BadReadException bre) {
            System.out.println("Bad Read Detected Here");
        } // end catch
        CPU.delay(30000);
        break;

    // Write a string of characters
    case '5':
        System.out.print("\nEnter message to be stored: ");
        keyboardMsg.clear();
        while ( (keyboardChar = Terminal.getChar()) != '\r' ) {
            keyboardMsg.append(keyboardChar);
        } // end while
        if (keyboardMsg.length()+address>32767) {
            System.out.println("\nThe message you have entered exceeds the end of
the EEPROM");
            CPU.delay(30000);
        } // end if
        else
            for (x=0;x<keyboardMsg.length();x++){
                ee.writeOne(dev,address+x,(byte)keyboardMsg.charAt(x));
                CPU.delay(50);
            } // end for
        break;

    // Read a string of characters
    case '6':
        System.out.print("\nPlease enter the length of the message you wish to
extract: ");
        x=getInt();
        System.out.println("\n");
        for (int y=0;y<x;y++){
            try {
                n = ee.readRandom(dev,address+y);
            } // end try
            catch (MC24LC256BadReadException bre) {
```

```
        System.out.println("\nBad Read Detected Here");
        CPU.delay(30000);
    } // end catch

    if ((n>31)&&(n<127))
        System.out.print((char)n);
    else System.out.print("<<NOT-PRINTABLE>>");

    } // end for
    System.out.println("\n\nPress ENTER to continue");
    Terminal.getChar();

    } // end switch
    } // end while
} // end method: main

public static int getInt() {
    keyboardMsg.clear();
    while ( (keyboardChar = Terminal.getChar()) != '\r' ) {
        keyboardMsg.append(keyboardChar);
    } // end while
    return Integer.parseInt(keyboardMsg);
} // end method: getInt

} // end class: MC24LC256DataStorageUtility
```

MC24LC256 Demonstration

The class, MC24LC256Demo.java, demonstrates more methods available to you that were not shown in this document. This class is included in the AppNote005-MC24LC256.exe file.

Published Resources – for More Information

This class was developed to allow the Javelin to interact with external EEPROMs. Much care has been taken in creating this class. Below you will find useful information that was used in creating this document.

“256K I²C™ CMOS Serial EEPROM”, Data Sheet, Microchip Technology Inc, 2002

This document will give you detailed information about the internal workings of the 24LC256 EEPROM.

Javelin Stamp Discussion Forum – Questions and Answers

The Parallax, Inc. Javelin Stamp Discussion Forum is a searchable repository of design questions and answers for the Javelin Stamp. To view the Javelin Stamp Forum, go to www.javelinstamp.com and follow the Discussion link. You can also join this forum and post your own questions. The Parallax technical staff, and Javelin Stamp users who monitor the list, will see your questions and reply with helpful tips, part numbers, pointers to useful web pages, etc.

Copyright © 2002 by Parallax, Inc. All rights reserved. Javelin, Stamp, and PBASIC are trademarks of Parallax, Inc., and BASIC Stamp is a registered trademark of Parallax, Inc. Windows is a registered trademark of Microsoft Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Other brand and product names are trademarks or registered trademarks of their respective holders.

Parallax, Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products.