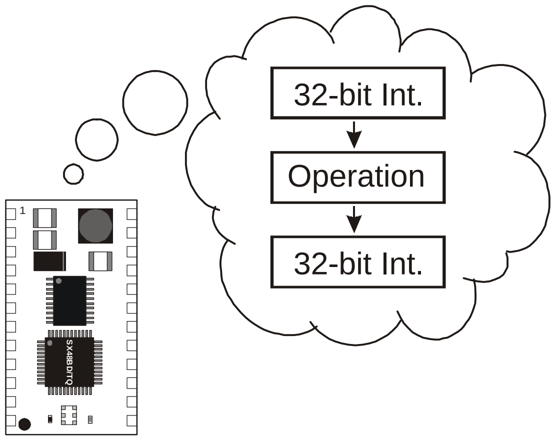
599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office/Tech Support: (916) 624-8333
Fax: (916) 624-8003

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 the 32-bit math library

The Int32 library class allows you to add, subtract, multiply, divide, compare and shift 32-bit integers. Support is included for signed and unsigned 32-bit integers, 16-bit integers, and direct access to the high and low fields of the 32-bit integer object.

A 32-bit signed integer has a value range from -2,147,483,648 to +2,147,483,647. An unsigned integer has a value range from 0 to 4,294,967,295. The Javelin natively has a signed 16-bit integer, which has a value range from –32768 to +32767.

Downloads, Parts, and Equipment for the 32-bit math library

This application note (AppNote011-Int32.doc), the Int32 library file (Int32.java), the library's javadoc file (Int32.pdf), the test program (Int32Test.java), the demonstration program (Int32Demo.java) which will demonstrate all the methods available to you in the Int32 class, and an application example (Int32ElapsedTime.java) are all available to you for free download from: http://www.parallax.com/javelin/
Once there select the "applications" menu, and it will take you to the application page.

You can use the AppNote011-Int32.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

The file list below is organized by directory, then by filename, please verify that your file list is organized in the same way.

```
<root path>\doc\AppNote011-Int32.pdf
<root path>\doc\Int32.pdf
<root path>\lib\stamp\math\Int32.java
<root path>\Projects\examples\math\Int32Demo.java
<root path>\Projects\examples\math\Int32ElapsedTime.java
<root path>\Projects\examples\math\Int32Test.java
```

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.

Int32 Library and How it Works

The Int32 library allows you to perform a variety of mathematical functions, which allow for 32-bit math. It does this by using the standard 16-bit integer that is available to the Javelin. The explanation in this section assumes you are familiar with the debug feature of the IDE (see Javelin Stamp Users Manual), as well as hexadecimal numbers. If you are not interested in how the Int32 library works, skip this section and simply go directly to *Testing the Int32 Library*.

What's an Integer?

An integer is a whole number and can be positive or negative. The Javelin's signed 16-bit integer (`int`) allows for values between –32768(0x8000) and +32767(0x7FFF). Positive numbers between 0 and 32767 are stored in RAM just like you would expect, with values from 0 to 32767. But, negative numbers are different. The Javelin's compiler, as with all computers, needs a special way to represent the negative sign, called "two's compliment". The Javelin's compiler does this by taking the absolute value of the negative number and subtracting it from 65536. For example, we have an integer with the value of –1, we take the absolute value of –1, which is 1, then subtract this from 65536 to get 65535. This value 65535, which is unsigned, is what is actually stored in RAM for an integer with the value of –1. Let's try another example, if we have an integer with the value of –31234, we would take the absolute value of this number, which is 31234 then subtract it from

65536, which would give us 34302. This is the value stored in RAM to represent –31234. Table 1.1 shows how integers are represented within the Javelin.

Table 1.1: Integer Representations	
Hexadecimal Value (0x)	0000, 0001, 0002, … 7FFE, 7FFF, 8000, 8001, … FFFD, FFFE, FFFF
Unsigned Integer	0, 1, 2, … 32766, 32767, 32768, 32769, … 65533, 65534, 65535
Signed Integer	0, 1, 2, … 32766, 32767, -32768, -32767, … -3, -2, -1

Let's say you wish to pass a value of 45000 into an Int32 object called **`uX`**, which will take an integer as a parameter. In order for you to pass a value larger than 32767, you would have to subtract 45000 from 65536, which will give you 20536, then negate it. This value, -20536, is what you would pass into **`uX`**.

```
uX(-20536);
```

The Javelin's compiler will convert this negative number (-20536) into the Javelin 16-bit signed integer equivalent (45000). If you do not want to worry about pre-converting the number this way don't worry, there is a shortcut! This shortcut is a special data type called **`short`**. To use this shortcut, and place the value of 45000 directly into the variable RAM, simply do this:

```
uX((short)45000);
```

The object **`uX`** only accepts signed integers; the Javelin does not have unsigned integers. It is an integer that we are using as an unsigned integer. So, when you print the value of **`uX`**, the compiler will convert 45000 into –20536. The reason for explaining this is because it is the quickest way to store values into the Int32 object. If you are not worried by speed, you can simply use a string. This will be discussed later in this application note.

Inside The Int32 Object

The Int32 object contains an Int32 data type, which has two integer fields, a high (**`Int32.high`**), and a low (**`Int32.low`**) as well as all of the mathematical methods. A 16-bit unsigned integer can contain a value from 0 to 65535, which in hexadecimal is 0x0000 to 0xFFFF. Two integers, 32-bits, can contain a value from 0 to 4,294,967,295, which in hexadecimal is 0x00000000 to 0xFFFFFFFF. These values are stored as unsigned values. They can be accessed as signed values from -2,147,483,648 to +2,147,483,647.

If an Int32 object contained the value of 10,000,000 (0x00989680), it would be stored in the following way, the lower 16-bits of 0x00989680, which is 0x9680, would be stored in the **`Int32.low`** integer, and the high 16-bit portion, 0x0098, would be stored in the **`Int32.high`** integer.

There are four constructors for the Int32 library, which you can use to:

1) Create a new 32-bit integer initialized to zero.
2) Create a new 32-bit integer initialized to value of another 32-bit integer.
3) Create a new 32-bit integer initialized to a 16-bit value (sign extends to high 16-bits).
4) Create a new 32-bit integer initialized to 32-bit value.

There are five methods, which set the value of the Int32 type. These are:

1) Set the 32-bit integer equal to the value of another 32-bit integer.
2) Set the 32-bit integer to a 16-bit value (sign extended).
3) Set the 32-bit integer to a 32-bit value.
4) Set the 32-bit integer to the converted value of a String.
5) Set the 32-bit integer to the converted value of a StringBuffer.

There are three methods, which allow you to enter data by using another 32-bit integer, using a 16-bit integer or directly accessing the high and low portions of the 32-bit integer by passing in two 16-bit integers. Each of these methods is available to you when using:

- addition
- subtraction
- signed or unsigned division
- multiplication
- signed or unsigned compare

The Int32 object also includes methods for:

- shifting bits to the right and left
- equality
- absolute value
- negation
- converting signed and unsigned values to a string

Testing the Int32 Library

Program Listing 1.1 is a short program that will perform a simple test of the Int32 library class. This program will perform a single multiplication.

The first line of code creates a new 32-bit object called **`num1`** with a value of 12239.

```
Int32 num1 = new Int32(12239);
```

Then we will display this value by using the **`toString()`** method within the Int32 object.

```
System.out.print(num1.toString());
```

Next we will multiply **`num1`** by 73 by passing 73 in as a parameter. The value in **`num1`** will be multiplied by the input value (73) and the result will be stored in **`num1`**. The original value of **`num1`** (12239) will be overwritten.

```
num1.multiply(73);
```

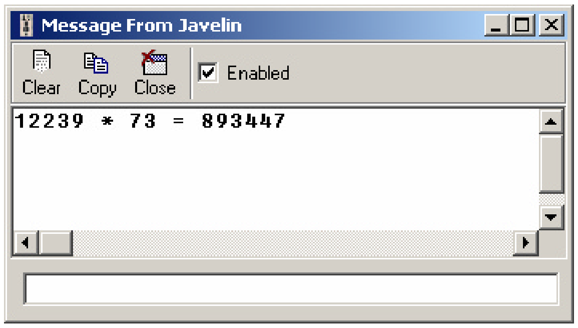
Then we will print a message to the display, which explains what we are multiplying.

```
System.out.print(" * 73 = ");
```

And then we print the result by using **`toString`** again.

```
System.out.println(num1.toString());
```

The test program (Program Listing 1.1) will display output, to the Javelin's IDE message window, similar to Figure 1.1.



```
12239 * 73 = 893447
```

Figure 1.1
Output for Int32Test

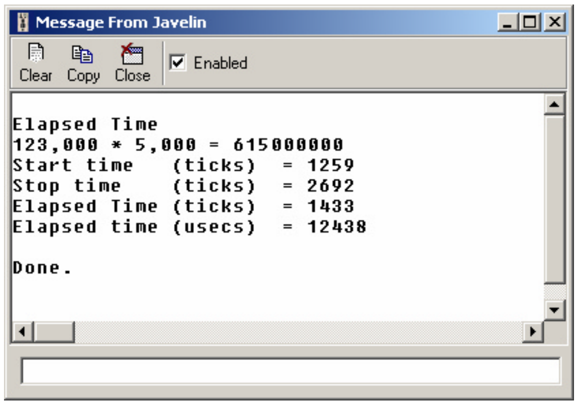
Program Listing 1.1 – The Int32 Test

```
import stamp.math.Int32;
/*
 * This is a simple math test for Application Note #11
 * @version 1.0 - February 10, 2003 (AppNote011)
 */


public class Int32Test {

  public static void main() {

    Int32 num1 = new Int32(12239);

    System.out.print(num1.toString());
    num1.multiply(73);
    System.out.print(" * 73 = ");
    System.out.println(num1.toString());
  }
}
```

A 32-bit Stopwatch

Program Listing 1.2 uses the timer library as a stopwatch to time events. In this example we will be timing the speed of the Int32 multiplication method. This is done by reading the timer before and after an event, calculating the difference between each reading, and converting the value into time in microseconds.

The output shown in Figure 1.2 should appear in the Messages from the Javelin Window/Debugger.



```
Elapsed Time
123,000 * 5,000 = 615000000
Start time   (ticks)  = 1259
Stop time    (ticks)  = 2692
Elapsed Time (ticks)  = 1433
Elapsed time (usecs)  = 12438

Done.
```

Figure 1.2
Output for
Int32ElapsedTime

Program Listing 1.2 uses the methods below from the Int32 library class.

- **`Int32()`**
  Create a new 32-bit integer initialized to zero.
- **`Int32(,)`**
  Create a new 32-bit integer initialized to 32-bit value.
- **`set( , )`**
  Sets the 32-bit integer to the 32-bit value.
- **`multiply()`**
  Multiplies the 32-bit integer by a 16-bit value.
- **`toString()`**
  Convert the signed 32-bit integer to a String.
- **`subtract()`**
  Subtracts another 32-bit integer from the 32-bit integer.
- **`add()`**
  Adds a 16-bit value (sign extended) to the 32-bit integer.
- **`udivide()`**
  Divides the 32-bit integer by a 16-bit value (unsigned divide).

Tip	Use the Int32 javadoc file (Int32.pdf in your <"Javelin Stamp IDE\doc"> folder) as a reference to find out more about using the methods available to you.

Program Listing 1.2 – An Elapsed Time Example

```
import stamp.core.*;
import stamp.math.Int32;

/*
 * The following example demonstrates the use of Int32 objects.
 * The example takes a 32-bit timer reading before an activity and another
 * 32-bit timer reading after an activity. It displays these start and stop
 * times in ticks, then calculates and displays the elapsed time in ticks and
 * in microseconds.  The Timer methods return the number of 8.68 microsecond
 * ticks since the timer was started.
 *
 * To convert ticks to microseconds we will use the formula:
 * elapsedTime = ((elapsedTime * 868) + 50) / 100
 *
 * @author Cam Thompson, Micromega Corporation
 * Copyright © Micromega Corporation 2002-2003. All rights reserved
 *
 * @version 1.0 - February 10, 2003 (AppNote011)
 */

public class Int32ElapsedTime {

  public static void main() {

    Timer  t = new Timer();
    int    nHigh;
    int    nLow;

    // Create 2 new Int32 object with initial value of zero.
    Int32 num1 = new Int32();
    Int32 num2 = new Int32();
    Int32 num3 = new Int32(0x0001, (short)0xE078);   // set num2 = 123,000

    System.out.println("\r\nElapsed Time");

    // get the start time and store in num1
    nLow  = t.tickLo();
    nHigh = t.tickHi();
    num1.set(nHigh, nLow);

    // the activity to be timed goes below here
    num3.multiply(5000);
    // the activity to be timed goes above here

    // get the stop time and store in num2
    nLow = t.tickLo();
    nHigh = t.tickHi();
    num2.set(nHigh, nLow);

    System.out.print("123,000 * 5,000 = ");
    System.out.println(num3.toString());

    // display the start and stop time in ticks
    System.out.print("Start time   (ticks)  = ");
    System.out.println(num1.toString());
    System.out.print("Stop time    (ticks)  = ");
    System.out.println(num2.toString());

    // elapsed time = stop time - start time
    // num2 = num2 - num1
    num2.subtract(num1);
    System.out.print("Elapsed Time (ticks)  = ");
    System.out.println(num2.toString());

    // convert to microseconds
    // num2 = ((num2 * 868) + 50) / 100;
    num2.multiply(868);
    num2.add(50);
    num2.udivide(100);

    // display elapsed time
    System.out.print("Elapsed time (usecs)  = ");
    System.out.println(num2.toString());

    System.out.println("\nDone.");

  } // end main

} // end class
```

Int32 method Demonstration

Int32Demo.java is a detailed, step-by-step, fully comprehensive demonstration of the Int32 class that is included with this application note.

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.