Rev – 6
2/Oct/07
Farokh Morshed

Learning Curl By Way of Java

This is a short article on learning Curl by way of Java. I hope that Java programmers that want to learn Curl can use this note as a high level primer before delving into the Curl Developer's Guide (the official definition of Curl).

By way of introduction, I am an architect working on the Curl IDE team as a developer/manager. My background is 30 years of writing compilers (mostly Digital's compiler front-ends and middle-ends), developer productivity tools (NuMega's performance, coverage, program analyzers, and model driven program generators), and production environment application server diagnostics and monitoring (Mercury Interactive Diagnostics products). I joined Curl about a year and a half ago. My main interest these days is to improve how we programmers can program through better languages, tools, environments, model driven architecture, and software factories.

Among my first tasks here was to learn Curl. Having programmed in Java for years, I learned Curl first by contrasting it with Java. This note is essentially a brain dump from that exercise. Even though I will try to touch on each major area, it will not be in much detail.

Note that I am not going to compare Curl with Java. To compare, it would mean to talk about subjects such as downloading bytecode versus text source code, efficiency of generic instantiations in the final code, optimizations, garbage collectors, the value of having global procedures, etc. Although these are all good subjects to discuss, they do not fit the purpose of this note.

Editorial note: Once the Curl Developer's Guide is available in HTML, I will revise this note with links to sections in those pages.

Let's jump right to it.

The Curl language was designed to write programs that range from documents (HTML like) to full applications. Java, to me, is a much better C++. Curl and Java have no overlap in providing document support, but lots of similarities in providing general application writing.

Essentially, Curl and Java both have all the basic language constructs for writing today's desktop applications, with plenty of UI classes, various collection classes, good OO methodology, and static type checking. Java can subjectively be considered as more of a general programming language, but Curl is definitely customized for creating rich Web content. There is a large common subset of language features and platform capability,

but then Java provides some general features (such as threads), while Curl provides some Web content specific features (such as text formatting), and then Curl adds some nifty language features (such as multiple value returns and macros) that Java does not have.

Whether Java is more general or Curl is actually subjective. If you define general as more "widely accepted and used", then obviously Java is more general. If you define general as being able to write code for "different system tiers", then Java is more general because you can write for desktops, application servers, embedded devices, etc. But if you define general as being prolific, having "lots of language features", then Curl is more general. So it all depends on what you mean by general.

You can easily write GUI centric and system code in Curl, as it has been done in the Curl IDE and RTE (They are both written in Curl). Being general is an important goal of the Curl platform. If you are writing your application in Curl, you should not have to mish-mash it with other technologies. Combining technologies gets you into unnecessary complexities, such as readability and debugging mess across technologies. Case in point is Ajax based solutions that combine HTML and Javascript (never mind the server technology that generates them) in asynchronous fashion. Not for faint-of-heart! Other up and coming proper RIA solutions, like JavaFX, are also recognizing this important requirement.

There is a lot to cover, so let's start with a focal point. The table below summarizes the major contrasts between Java and Curl. Use the links to get more detail on a subject.

Concept or Construct	**In Java**	**In Curl**
Procedural vs. Declarative programming	Procedural only	Declarative and procedural…
Top-level	Package, import, and types	Top-level expressions…
Classes	Yes	Yes, also has value classes…
Inheritance	Yes	Yes, also has multiple inheritance
Package	Yes	Yes, but dynamic packages…
Compilation unit	Class, interface	File, but not really a compilation unit, just source text…
Class members	Yes	Yes, but no nested classes…
Interface	Yes	No, but has abstract classes…
Threads	Yes	No, but has applets and events…

Primitive Types	Yes	Yes, also has quantities…
Reference types	Yes	Yes
Generics	Yes	Yes, but only parameterized classes
Basic language constructs (if, conditional, loops, etc.)	Yes	Yes, and other features not in Java…
Interfacing to other languages	Yes (JNI)	Yes, DLLs and JavaScript…
Text formatting	No	Yes, lots of text formatting…
Closures	Kind of	Yes, anonymous procedures…
Running Programs	Applications, Applets, Servlets, Beans, etc.	Applets, detached applets…
Extensibility	No	Yes, macros…

Declarative and Procedural Styles

Programming declaratively, as opposed to procedurally, is better suited for creating UI. The meaning of the program is easier to grasp. So, you need a language syntax that let's you write declaratively. As it turns out, you need both. You still need to program the algorithms behind the UI, and the business logic, in OO procedural style. The need for the declarative style is not unique to Curl and it is becoming a commodity for languages targeted to RIA programming. The JavaFX folks have a good example here https://openjfx.dev.java.net/Learning_More_About_JavaFX.html that shows the advantages of being able to program in JavaFX script declaratively. Below is an example that compares Java and Curl. Both programs essentially do the same thing: They create two buttons that change the background color of their pane. Putting aside the verboseness of Java compared to Curl, note how easier it is to read the Curl program. You are also seeing how Curl is customized to create web content in that it assumes its top window must be a child of the browser window and it provides an automatic (hidden) UI event loop.

Java program:

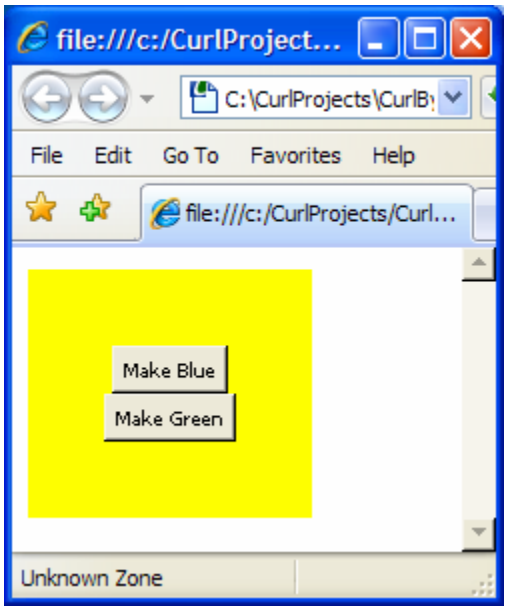
```
package curlbyjava;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
```

```
import javax.swing.border.EmptyBorder;

// Two buttons that change their container's background color
public class ButtonDemo extends JPanel implements ActionListener {
    protected JButton b1, b2;

    public ButtonDemo() {
        setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        setBackground(Color.yellow);
        this.setBorder(new EmptyBorder(10, 10, 10, 10));

        b1 = new JButton("Blue");
        b1.setActionCommand("b1");
        b2 = new JButton("Green");
        b2.setActionCommand("b2");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b1.setToolTipText("Click to make blue.");
        b2.setToolTipText("Click to make green.");
        add(b1);
        add(b2);
    }

    public void actionPerformed(ActionEvent e) {
        if ("b1".equals(e.getActionCommand())) {
            b1.getParent().setBackground(Color.blue);
        } else {
            b2.getParent().setBackground(Color.green);
        }
    }

    private static void createAndShowGUI() {

        JFrame frame = new JFrame("ButtonDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new ButtonDemo());
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Same program written in Curl:

```
{curl 6.0 applet}

{VBox
   halign = "center",
```

```
    margin = 1.0cm,
    background = "yellow",
    {CommandButton
        label = "Make Blue",
        tooltip = {Tooltip "Click to make blue."},
        {on Action at b:CommandButton do
            set b.parent.background = "blue"
        }
    },
    {CommandButton
        label = "Make Green",
        tooltip = {Tooltip "Click to make green."},
        {on Action at b:CommandButton do
            {set b.parent.background = "green"}
        }
    }
}
```



Top-level expressions

In Java, the goal symbol of syntactic grammar consists of package, import, and type declarations. Type declarations consist of class and interface types, and so on.

In Curl, a program consists of a series of expressions (usually surrounded by curl braces). Most expressions produce a value, and the value of an applet's last top-level expression is displayed in the browser.

Consider this Hello World applet in Curl.

```
|| Define a top-level procedure
```

```
{define-text-proc {message ...}:any
   {return
      {TextFlowBox
         font-weight = "bold",
         {value ...}
      }
   }
}

|| Now, call the procedure and let the applet display the returned object
{message Hello World!}
```

This is quite different than a Java applet with a class that has a main method, etc.

Source Files

Most implementations of Java (all I know of) rely on hierarchical file systems to manage source and class files. You put the source code of exactly one class (actually a type that can be a class, interface, enumeration, or annotation type) in a source file, arrange your source files in directories with names that reflect the containing package, and compile the sources into class files in the same hierarchical arrangement as sources but with a different root. So the file for class "graphics.Rectangle" is in "class-root/graphics/Rectangle.class" with its source in "source-root/graphics/Rectangle.java". Then, you give the compiler or the JVM a classpath containing the root to find your sources or classes. But, you already know how to do all that.

In Curl, what source files can contain, the compilation step, their location (classpath), and how the RTE (including the compiler) finds them are all different!

In a nut-shell:

- Source file content – Can be any set of Curl syntax. Usually, they contain a set of imports, global procedures, related classes, etc. In this case, Curl is more like C++ or C# than Java.
- Compilation step – There is none! You provide source code text to Curl and it compiles it incrementally as it needs to. Curl compiles the Curl text syntax into native code incrementally with no persisted intermediate state.
- Classpath – Is specified using the location parameter in an import statement or in a manifest (see packages). Typically, the import statements in source files do not have a location parameter. Instead, they rely on the manifest to locate the files.
- Location – Location for source files in a package is a URL and has nothing to do with the name of its directory. So, classes for package A.B.C can be located in directory "some-root/xyz". See "Packages" for more details. This level of indirection lets a class to be located somewhere other than the local file system.

Packages

In Java, placing a class in a package does two things: First, it creates a separate namespace so importers can import symbols by prefixing the class name with the package name, and second the class's physical location relative to a classpath is determined (whether in a directory on disk, or in an archive). In Curl, the first is the same, but the second is different. The physical location of Curl's imported symbols is determined by the location parameter to the import statement or better yet by a manifest file. This separation gives it a level of indirection.

Here is an example.

Let's create a math package that I can import from another package, or applet, etc.:

```
Directory: math-package-dir
        File: load.scurl
                {curl 6.0 package}
                {package COM.EXAMPLE.MATH.TRIG}
                {define-class public MathTrigs
                        {method public {cos angle:Angle}:float
                                …
                        }
                }
```

Above, I have a file called load.scurl in directory math-package-dir. This file represents a package by the name of COM.EXAMPLE.MATH.TRIG. This package defines one public method, cos. Typically, you would define the types inside a package (the class, procedure, etc.) in a separate file and include that file in the package. But for our example we skip that.

Let's use the above math package:

```
Directory:  math-stuff
        File: math_wizard.curl
                {curl 6.0 applet}
                {applet manifest = "math-wizard-manifest.mcurl"}
                {import * from COM.EXAMPLE.MATH.TRIG}
                {let do-trig:MathTrigs = {MathTrigs}}
                {VBox
                        {bold cosine of an angle of 35 degrees is:
                                {value
                                        {do-trig.cos 35}
                                }
                        }
                }
        File: math-wizard-manifest.mcurl
                {curl 6.0 manifest}
```

```
{manifest APPLET-PROJECT}
{component package COM.EXAMPLE.MATH.TRIG,
   location = "/math-package/load.scurl"
}
```

The applet above uses the manifest math-wizard-manifest.mcurl to locate the COM.EXAMPLE.MATH.TRIG package. Now, if you load the math-wizard.curl file into a browser, it should just work. If you deploy the applet into a server, you should have to only change the location of the package in the manifest file.

This is all similar to using classpath, or the class-path header field in a Jar's manifest, with one important difference. The location parameter is a URL. Therefore, the file can be anywhere on the internet. In Java, the classpath can only consist of the locations in the local network. To get a file from the internet you have to write some Java code.

A Curl manifest can also delegate to another manifest by simply adding a component attribute and giving the location of the other manifest. Makes picking up a third party library easy.

Also, Curl has a concept of super packages: A package that contains the symbols of other packages. Useful when you want to combine a set of packages into one entity that can be imported all at once.

No Nested Classes

Curl does not have nested classes. I sometimes miss not having them, but not as much as I would have if I could not put more than one class in a Curl source file and if I did not have anonymous procedures.

I miss nested classes because of the "increased encapsulation" feature. Let me explain. As you know, there are different kinds of inner classes in Java with different benefits:

1. Static member classes – Benefits: Logical grouping of classes, increased encapsulation, more readable and maintainable.
2. Non-static member classes (inner classes) – Benefits: All benefits of static member classes plus access to instance members of enclosing class. There are two kinds of inner classes:
    a. Anonymous inner classes – Additional benefits: When you want to define and use all at the same place and don't really need a name.
    b. Local inner classes – Additional benefits: None

The "more readability" benefit is already in Curl because you can put more than one type in the outer scope in a source file. The "logical grouping" benefit is not that big a deal because you can still hide the type in its package so it does not become part of the package API (even though it does become visible to rest of the types in the package). The "increased encapsulation" where the nested class can access private members of the enclosing class (another way of saying that you don't have to make the private members

of enclosing class package-private just so the nested class can access it) is what I miss. I also miss the anonymous inner class benefits where I can define and instantiate an entire class that implements an interface or extends a class. But, Curl does have anonymous procedures which provides some of the Java anonymous inner class benefits, and also can do things that Java cannot. This subject deserves its own section, anonymous procedures.

Anonymous Procedures

While Java has anonymous classes (a kind of inner class), they do not quite live up to the general idea of closures. Also, while Curl has anonymous procedures, a more orthodox implementation of closures, Curl does not have the ability to anonymously implement an entire "contract".

To best of my knowledge closures originated in Lisp, and have since appeared in many languages. The idea is to write a block of code to close over its surrounding scope.

You already know how to use anonymous classes in Java. You define and instantiate a class at the same time, that implements an interface or extends a class, and this new class (with no name) can read the variables in its surrounding scope. Where it falls short of the complete idea of closure is that it cannot write to the variables outside its scope. That is why you must declare as final any accessed variables in the surrounding scope. That is why we put our variable in a reference type, make the reference type final, and only then we can write to the variable.

In Curl, an anonymous procedure can read and write to its surrounding scope variables. You don't need to do the trick above. This is not an easy thing to do for compilers, but sure makes it easier for the programmers.

In Curl, anonymous procedures, as well as global and class procedures, are first-class objects with their own data types. Hence, you can pass them as arguments, return them, store them in a data structure, etc. In Java, you would have to first enclose the method in a class.

Abstract Classes but No Interfaces

Curl does not have the type interface. But, it does have the abstract class type, and it allows multiple inheritance, so you can accomplish the same using abstract classes. As you know, in Java you can use an abstract class in place of interface in many cases. See http://mindprod.com/jgloss/interfacevsabstract.html for a good breakdown on when to use the Java abstract class versus the interface type. The same applies to Curl. Essentially because Curl has multiple inheritance, then not having the interface type becomes a matter of taste rather than hindrance.

Applets and Events

In Java, managing concurrency is solved by the threads model: There are one or more threads (in a thread group) that share memory. Threads synchronize among each other using locks. Typically each Java thread maps to an OS thread.

In Curl, managing concurrency is solved by the events model: Each applet maps to an OS thread. An applet has an event queue. Each node on the event queue consists of an event type and an event target to which the event is to be fired. Events are fired at event targets (such as keyboard action events). An event target (subclass of EventTarget and the parent class of UI classes) provides an event handler which is an event type and a procedure. In this model, the events are fired by your code or RTE code at targets, events are queued to the applet, applet pulls events off of the event queue, finds the corresponding event target, and if the event type in the event node matches the event type of one of the handlers in the event target then it invokes its corresponding event procedure.

These are two text book ways of managing concurrency, and the debate rages as to which is better! There are those that say that the thread model is too hard to program and too dangerous. Never need it! There are those that argue that the event model is too simple and prevents you from doing many things. There are also those that stay objective in this debate. For a good read see http://home.pacbell.net/ouster/threads.pdf.

My own conclusion is that for scalable server programs that can utilize multiple CPUs threads are better. I also sympathize with the argument that split-phase handlers (such as disk I/O handlers) are easier to write and more efficient using threads. But for writing client side applications the events model is easier to write and debug. You also don't have to worry about deadlocks that threads are so famous for!

Of course, you can certainly have multiple applets in the same OS process in a Curl program. Also, each applet can have sub-applets. But, these applets do not share memory. Be aware that sub-applets are not well documented (although used often in the RTE code).

Procedures

In Curl like C++ and C#, you can define a global procedure. A procedure can also be defined inside a class, which in this case it behaves the same as a Java static method.

Macros

This subject deserves its own full article. To quickly summarize, you write macros to extend the syntax of the language. In fact many of the language constructs such as "on", "after", "for", "define-class", are themselves macros. Curl macros are similar to function-like C macros with an important difference. C macro processing is a preprocessor concept that is done before parsing begins. Curl macro expansion is done during parsing and execution. What you are substituting is already a parse tree, and the parse tree is constructed by actually executing some of your Curl code inside the macro

definition! This difference really matters as you can imagine. For example (although this example does not do justice to the magnitude of the difference), unless you properly parenthesize a C function-like macro parameter you can get wrong expansions when the parameter is an expression. But in Curl you don't need parenthesis, because the substitution is already a parse tree. See the section on syntax-switch in the Curl Developer's Guide for an example of a macro that actually executes code to transform a set of given identifiers into a class with those identifiers as fields. Pretty wild!

Running Programs

Generally Java programs run in the following ways:

1. Desktop applications
2. Applets
3. MIDlets
4. Inside a J2EE container (Servlets, beans, etc.)
5. Called from an application written in another language.

Curl programs run in the following ways:

1. Desktop applications – Curl detached applets run as a standalone application
2. Applets – Applets run hosted in a browser and served up from an http server (or local file system). They can also be imbedded in HTML like Java applets, but I really don't know the use case for this since you can do it all in Curl.
3. Called from an application written in another language – You can call Curl from JavaScript and vise versa.

Part of distributing and running programs is to deal with versioning issues. Curl applets declare what version of the RTE they are supposed to run under. Note the {curl 6.0 applet} herald in the example program above. When this applet is run the RTE loads its 6.0 engine (called helper engines) to execute the applet. In fact, multiple versions of the RTE can run at the same time on a machine. This feature solves the "Dll hell" problems.

Other Features Not In Java

In addition to the above features touched on above, here are some other features in Curl but not in Java:

1. **Multiple return values** – Method can return more than one value, make it easier than creating a class to contain the values.
2. **Quantities** - A quantity is a numerical value together with an associated unit of measurement. Hence, quantities have their own type and compiler will do type checking for you. For example {value 1meter + 5centimeters} is 1.05m. On the other hand, {value 6m + 2s} results in a syntax error.
3. **Strict and loose type system** – Because Curl can be used for quick prototyping it allows you to play loose with the type system, similar to JavaScript. For example, you can declare the type of a variable to be "any" which stops the compiler from type checking but leaves the burden of type conversion to the run-time. Bad idea for robust code but handy for prototyping and some esoteric code.
4. **Text formatting** – Curl has extensive support for HTML-style text formatting that replaces the need to use HTML.

5. **Value classes** – Curl's value class is the same idea as the C++ struct: They are not an object, cannot inherit, and they are passed by value.
6. **Flexible layouts (stretchy)** – The GUI classes use easy to program layouts that adapt and resize themselves to the area of the screen.
7. **Keyword and rest arguments** – Provides a mechanism for writing concise code for methods that take a large or unknown number of arguments.

Conclusion

I have to say, it was so easy to get started writing in Curl. But as I wrote more complicated code I had to read the Developer's Guide more carefully. This is a large language, but luckily it has a fairly gentle slope. Curl is a natural platform to use to write Web content. That is what it is designed to do.

Many thanks to David Kranz and Phil Shapiro for providing valuable review comments on the early revisions of this note.