

The magazine of the ACCU

[www.accu.org](http://www.accu.org)

# {cvu}

Volume 20 Issue 4 August 2008 £3

## Features

Restaurant C++ and Pidgin Python  
**Pete Goodliffe**

Java Web Start  
**Paul Grenyer**

World View of a Java Champion  
**Peter Pilgrim**

## Regulars

Desert Island Books  
Code Critique  
Book Reviews

**Alison Lloyd**

An Introduction To The Java Native Interface





**Editor**

Tim Penhey  
cvu@accu.org

**Guest Editor**

Guy Bolton King  
gbk@birchcrown.demon.co.uk

**Contributors**

Pete Goodliffe, Paul Grenyer,  
Allan Kelly, Alison Lloyd,  
Roger Orr, Peter Pilgrim.

**ACCU Chair**

Jez Higgins  
chair@accu.org

**ACCU Secretary**

Alan Bellingham  
secretary@accu.org

**ACCU Membership**

Mick Brooks  
accumembership@accu.org

**ACCU Treasurer**

Stewart Brodie  
treasurer@accu.org

**Advertising**

Seb Rose  
ads@accu.org

**Cover Art**

Pete Goodliffe

**Repro/Print**

Parchment (Oxford) Ltd

**Distribution**

Able Types (Oxford) Ltd

**Design**

Pete Goodliffe

# A sideways glance at Java

**W**elcome to the August issue of CVu! If you subscribe to Overload as well as this journal, then this is the last time both will be arriving on your doormat together: as of issue 20.5, the two journals will be published in alternate months. In other words, the journal latency is increasing, or if you consider that a wanton abuse of a well-defined technical term, we're spreading the journal marmalade more evenly over the year's toast. The next issue of CVu will arrive in November, which means that potential authors have a whole more month than usual to ponder, propose and polish their first submission...

Something else you may have noticed is that I'm not Tim Penhey. For those of you who missed the Publications Officer's report in the previous issue, I'm the second of a short succession of guest editors while we find a new editor for CVu: if you think you'd like to have a go at editing an issue, or taking on the job for longer, please email [publications@accu.org](mailto:publications@accu.org).

This month we have a slight Java flavour to CVu. If you haven't taken a look at Java recently (and who could blame you: you may have been getting excited by all the goodies in C++0x), then perhaps it's time you did. The language leapt forward with Java 5 in 2004, and closures and type inference may yet make it into Java 7, making it more interesting and hopefully less verbose. However, in recent years Java the language has started to share Java the platform with a number of others, in a decidedly similar way to .NET's CLR: a variety of useful and interesting languages now compile to Java bytecode, statically and/or dynamically, to run on the JVM. You can do functional programming and strong typing with Scala, extension scripting and rapid prototyping with Groovy, Jython, JRuby and JavaScript (via Rhino), and soon you'll be able to write rich internet applications with the declarative JavaFX. All of these can use and extend the huge number of libraries that already run on the JVM, JNI is there for you if you need access to the bare metal, and you can deploy them all with WebStart. Several of them have quick-start web-frameworks: JRuby on Rails, Groovy on Grails and Scala on Lift (not to mention the as-yet-unreleased-and-may-never-be Rhino on Rails).

So if you can chisel a gap in your Copious Free Time, and you have a yen to learn something new (and if you're in ACCU, you almost certainly do), I hope the contents of this issue give you something to fill it with.

**Don't forget:** CVu 20.5 is out in November; Gail Ollis will be your editor for that issue.



GUY BOLTON KING,  
GUEST EDITOR

## The official magazine of ACCU

ACCU is an organisation of programmers who care about professionalism in programming. That is, we care about writing good code, and about writing it in a good way. We are dedicated to raising the standard of programming.

ACCU exists for programmers at all levels of experience, from students and trainees to experienced developers. As well as publishing magazines, we run a respected annual developers' conference, and provide targeted mentored developer projects.

The articles in this magazine have all been written by ACCU members – by programmers, for programmers – and have been contributed free of charge.

To find out more about ACCU's activities, or to join the organisation and subscribe to this magazine, go to [www.accu.org](http://www.accu.org).

Membership costs are very low as this is a non-profit organisation.

---

## DIALOGUE

- 16 Desert Island Books**  
Paul Grenyer introduces Allan Kelly and his selection of books.
- 18 Regional Meetings**  
Local ACCU gatherings.
- 19 Code Critique Competition**  
This issue's competition and the results from last time.
- 22 Bookcase**  
The latest roundup of ACCU book reviews.
- 24 ACCU Members Zone**  
Reports and membership news.

---

## FEATURES

- 3 An Introduction to the Java Native Interface**  
Alison Lloyd shows us round the JNI, avoiding some traps on the way.
- 6 Restaurant C++ and Pigdin Python**  
Pete Goodliffe asks us to understand the idiosyncracies of the languages we're working in.
- 8 JavaOne: JavaFX and the Future of Java**  
Peter Pilgrim shares his perspective of the JavaOne conference.
- 11 Java Web Start**  
Paul Grenyer demonstrates an easy way to distribute Java apps.

---

## COPY DATES

- C Vu 20.5:** 1<sup>st</sup> October 2008  
**C Vu 21.1:** 1<sup>st</sup> December 2008

---

## IN OVERLOAD

Alex Fabijanic describes a way of dynamic typing in C++, Klaus Marquardt discusses possible treatments for 'Performatitis', Stuart Golodetz examines partition trees, Bill Clare looks at different approaches to parameterization, Anthony Williams explores error handling and Allan Kelly begins a new series: 'On Management'.

---

## ADVERTISE WITH US

The ACCU magazines represent an effective, targeted advertising channel. 80% of our readers make purchasing decisions or recommend products for their organisations.

To advertise in the pages of C Vu or Overload, contact the advertising officer at [ads@accu.org](mailto:ads@accu.org).

Our advertising rates are very reasonable, and we offer advertising discounts for corporate members.

---

## COPYRIGHTS AND TRADE MARKS

Some articles and other contributions use terms that are either registered trade marks or claimed as such. The use of such terms is not intended to support nor disparage any trade mark claim. On request we will withdraw all references to a specific trade mark and its owner.

By default, the copyright of all material published by ACCU is the exclusive property of the author. By submitting material to ACCU for publication, an author is, by default, assumed to have granted ACCU

the right to publish and republish that material in any medium as they see fit. An author of an article or column (not a letter or a review of software or a book) may explicitly offer single (first serial) publication rights and thereby retain all other rights.

Except for licences granted to 1) Corporate Members to copy solely for internal distribution 2) members to copy source code for use on their own computers, no material can be copied from C Vu without written permission from the copyright holder.

# An introduction to the Java Native Interface

Alison Lloyd shows us round the JNI, avoiding some traps on the way.

**T**he Java Native Interface often seems to be regarded in the same light as advanced string theory: everyone's heard of it, and knows a little bit about it, but it's also seen as complicated, requiring specialised knowledge to use. A number of problems can be solved (or at least reduced) via its application; and talking about JNI at parties is a sure-fire way of not getting invited again.

While it's true that some frighteningly complicated JNI-based systems exist, the fundamental building blocks are simple, and once you see the trick, creating JNI-based applications is (mostly) trivial. In this article, based on my talk at the 2008 ACCU conference, I'll introduce the fundamentals of using JNI, and demonstrate a simple application. I'll also touch on some of the useful features provided by the system, and finally, I'll discuss the most irritating pitfalls. If you take nothing else from this article, read the pitfalls section; if you ever need to use JNI in the future, it'll save you time, grey hair, and at least a week in rehab.

So what exactly is JNI? In essence, it is a way for Java code, running in its virtual machine, to interact directly with other, native code, possibly written in C, C++, or whatever. As a standard part of the Java system, you can use the JNI to embed native code (perhaps written in C) into your Java system. Alternately, you could embed a Java virtual machine into a system written in C++, if you wished. In short, JNI provides all the necessary glue and translation for native code to live and work happily with Java.

Using JNI, you can do any of the following:

- Call native code from Java code
- Call Java methods from native code
- Create and use Java objects
- Throw and catch Java exceptions from native code
- Embed a JVM (complete with Java code) into a native application.

The Java native interface may be used to mitigate one of Java's problems, which is a lack of hardware support. No Java-related text would be complete without some mention of platform independence, and this one is no exception. JNI allows systems developers to talk directly to hardware, which is particularly useful when you're working with hardware completely unsupported in Java. This is done in such a way as to retain as much platform independence in the Java code as possible, which is nice.

In particular, the JNI is good for:

- Reusing existing libraries
- Direct hardware access, adding support for hardware
- Time-critical code / operations
- Making use of another language's better support for something.

Which leads to the other side of something very versatile and powerful: when shouldn't it be used? As already mentioned, integrating native code into a Java application makes it somewhat more platform dependant, in the sense that it only runs on one platform: if you'd like to run elsewhere, you need to port the native code. In addition, the application becomes more complicated, with all the attendant opportunities for bugs and memory leaks; at the very least, you add an extra compilation stage. If you find yourself having a conversation that goes something like, 'We absolutely have to use Java for this project!' then JNI wizardry probably isn't going to save you.

## A basic JNI application

The JNI exists towards the bottom of the Java ecosystem, around where the JVM interfaces with the underlying system, providing the link between the Java stack and an external (native) block of code. There are thus 3 parts to this model:

1. The Java application, comprising the various Java objects and the JVM;
2. Some native code, helpfully wrapped up into a dynamic library;
3. The JNI, sitting somewhere between 1 and 2.

For the purposes of this article, I will be discussing Windows programming, using DLLs, but everything applies equally to other systems that support Java. The native code is usually wrapped in whatever the platform uses for dynamically loaded code (e.g. \*NIX shared objects), while the Java code stays the same.

In time-honoured fashion, I will be writing a 'Hello World' program, with the twist that the actual printing-to-screen will be done by some native code; you may imagine that I'm using a really esoteric monitor that isn't supported by Java, if that helps.

## Step 1: Design the system

System design is a fairly subjective topic, about which much has been written. On the basis that if one can't be a shining example, one should at least be a dire warning, the general rules of thumb I follow are:

- Try to stick to the 'one piece of functionality equals one class' idea;
- Have a clear idea of what each piece of native code is going to do, and split it up into simple functions;
- It may make sense to split the required native code into several libraries for neatness;
- Keep as much of the functionality in the Java application as possible – keep the native code as simple as possible;
- Wrap the native code in wrapper objects, which allow it to be easily isolated from the rest of the system.

Bearing in mind that every time you need to change the native code, you add a compilation stage to building your application, I would recommend trying to keep the native code to stuff that doesn't change very often. Keeping as much of the functionality in the Java code helps with this.

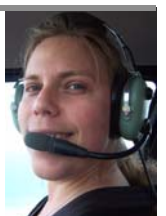
Our example program, being pretty simple, only requires a single piece of native code, which prints a string to the screen. This will be implemented as one native function.

## Step 2: Write the Java code

In the Java application, the printing functionality will be provided by a native library. It is necessary to tell Java at compile time what methods are

### ALISON LLOYD

Alison used to work in education but is now recovering. When not fiddling with her motorbike or flying helicopters, she has been known to do computer-related things. She writes embedded software, mainly to fund her flying habit. She can be contacted at [alison@zinc.org.uk](mailto:alison@zinc.org.uk).



implemented elsewhere, and what those methods look like, which is done by marking a method as native. For example:

```
public native void printString (String strOut);
```

Note that the method must exist inside a class, like all other methods. It may be public, private or protected, and it may take and return any types. You may overload native methods, too. Rather like an abstract method, there is no implementation, just a prototype; if it helps, think of a native method as like an abstract method, where rather than being implemented in a descendant class, the implementation comes in a (separate) native library. You may have as many native methods in a class as you like.

In addition to some native methods, you need to tell the JVM where to find the native implementations. As a dynamic library should only be loaded once, this is done in a static, or class-level block, for example:

```
static {
    System.loadLibrary("myLib");
}
```

This is tied to a class, and makes sure the library is loaded when the JVM loads the class. Of particular importance is that the name passed to **System.loadLibrary** will be converted to a platform-specific scheme, so if this code was running on Windows, the JVM would look for something called `myLib.dll`. Conversely, if it were running on Linux, toys would be thrown if `libmyLib.so` were nowhere to be found.

In a larger, more complex system, this would form a wrapper class, which provides an interface between the rest of the (Java) system and the native code. If the native methods are public, other objects may call them directly. Alternately, you may wish to keep your native methods private, and have other objects call wrapper classes; this is useful when you need to build on the native functionality with some additional logic within your Java application.

It isn't necessary to load the dynamic library in the class whose native methods it implements; it just has to be loaded somewhere in the system. That said, loading libraries outside the classes where they're used doesn't really add much beyond needless complexity. If you find you're defining native methods in several different (Java) classes, you may want to consider splitting your native code into several libraries.

Putting all this together, you should end up with something like Listing 1. Having written your wrapper class, you should compile it and sort out any syntax errors.

### Step 3: Generate the native header

The Java SDK comes with a handy tool for translating Java native methods into C / C++ header files, called `javah`. For our example, having compiled the `HelloWorld` class, you would do:

```
javah -jni HelloWorld
```

This would produce a header file rather like Listing 2. Of note is that each native method has been rendered into a C prototype, with the name being munged into something that includes class, method, and parameters (more on this below). In addition to whatever parameters were specified, each method also gets a pair of standard parameters, which are used to access JNI functionality. The header file pulls in the JNI functionality via the `jni.h` header.

### Step 4: Implement the native code

The native code in this example will be implemented in C. I generally use Visual Studio to roll up a DLL when I need one, but you can of course use whatever method you prefer. Having created the DLL, add the generated header from step 3, and then copy the prototype to a source file. Fill in useful names for the parameters and implement whatever functionality you need.

```
public class HelloWorld
{
    public native void printString (String strOut);

    public static void main (String args[]) {
        HelloWorld app = new HelloWorld();

        app.printString("Hello World!");
    }

    static {
        System.loadLibrary("myLib");
    }
}
```

Listing 1

```
/* DO NOT EDIT THIS FILE - it is machine
generated */
#include <jni.h>
/* Header for class HelloWorld */

#ifndef _Included_HelloWorld
#define _Included_HelloWorld

#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:      HelloWorld
 * Method:     printString
 * Signature:  (Ljava/lang/String;)V
 */
JNIEXPORT void JNICALL
Java_HelloWorld_printString
    (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

Listing 2

```
#include "HelloWorld.h"
#include <stdio.h>

JNIEXPORT void JNICALL
Java_HelloWorld_printString
    (JNIEnv *jenv, jobject jobj, jstring str) {
    const char *s = (*jenv)->GetStringUTFChars
        (jenv, str, 0);

    printf("%s\n", s);

    /* Free up memory to prevent memory leaks */
    (*jenv)->ReleaseStringUTFChars(jenv, str, s);
}
```

Listing 3

The native code for our example application is given in listing 3.

The `JNIEnv` and `jobject` parameters are used to access various JNI helper functionality. In this case, we need to convert the Unicode Java string into a C char array, which is done using the `GetStringUTFChars` function. This allocates some memory, so when you're done with the string, you should call the `ReleaseStringUTFChars` function. There are a number of other string-related functions available - see 'String Operations' in section 4 of the JNI specification.

Before compiling the native code, you need to...

## Step 4a - Tell Visual Studio where to find the JNI stuff

You need to add the JNI location to your include path. This will be the Java SDK installation, as follows:

```
<java install>\include AND
<java install>\include\<platform>
```

So something like:

```
C:\jdk1.5.0_15\include
C:\jdk1.5.0_15\include\win32
```

Having done this, you should be able to compile the DLL. In order to run the application, the DLL needs to be somewhere the JVM can find it. This may be either the execution directory (i.e. wherever the Java application is being run from) or in the search path (which may be altered by fiddling with the JVM initialisation files – here be dragons); the path of least resistance is to place it in the same directory as your application.

Finally, the example ‘Hello World’ application should run, predictably printing "Hello World!" to the screen.

## Useful information

Some background is in order at this stage. In particular, how exactly are the Java native method names converted into the C function names, how does the JNIEnv pointer work (and what else can it do), and how does one call Java methods from within the native code?

Native function names are constructed from:

- The prefix **Java\_**
- Mangled fully-qualified class name, followed by a separator ("\_")
- Mangled method name
- For overloaded native methods, two underscores followed by the mangled argument signature.

Note that the fully-qualified class name **includes the package name, if any**. Read that again! When generating the C header with **javah**, if you don’t specify the full class name, including the package, **the generated C prototypes will be wrong**, which means that the JVM won’t be able to find the matching implementation for the native methods. If you get an **UnsatisfiedLinkError**, check your native function names.

Method signatures, which are necessary for calling Java methods or working with Java objects from native code, are a shorthand for uniquely identifying a given method. Primitive types are represented as single letters, while object types are fully qualified (with an ‘L’ prefix); arrays are designated with an opening square bracket ([), followed by the array type.

The general method signature form is ( **arg-types** ) **return-type**.

For example, the method:

```
long foo (int a, float b[], String c)
```

would give:

```
(I[FLjava/lang/String;)J
```

The full list of type representations can be found in section 3 of the JNI specification.

The JNI environment (JNIEnv) pointer passed into all native functions provides a handle to the JNI functionality. In particular, this gives access to the various translation mechanisms for converting Java types and objects into a form that may be used in C. It also gives access to the invocation API, which allows calling of Java methods (and object creation / manipulation) from within native code. In the same way that the Java library APIs contain just about everything you can imagine (and many things you probably shouldn’t), the JNI environment provides (almost) everything you need to translate Java gubbins into C.

An example of this is calling a Java method from native code. In order to do this, you need to know the class name, method name, and method

signature (see above for signature construction) for the method you want to call. You then use **JNIEnv->GetMethodId** to get an ID for the method, and call it using **JNIEnv->CallXXXMethod**, where **XXX** is the return type (there’s a version of **CallXXXMethod** for each possible return type). There are equivalent functions for calling static methods. Accessing object fields is done in pretty much the same way (get a field ID, **GetXXXField**).

Note that some data-related functions cause memory to be assigned, such as the string translation functions. The (primitive) array handling functions are another example. When working with these data types, you need to remember to free the memory when you’re done, generally using the relevant **JNIEnv->ReleaseXXX** method:

- **ReleaseStringUTFChars**
- **ReleaseIntArrayElements**
- etc.

## Pitfalls

There are a few pitfalls to be aware of when working with the JNI. In typical fashion, this sort of thing will cause endless frustration when the system mostly works, yet fails in odd, irritating ways. I have already touched on several, but I’ve gathered them into a single section at the end for easy finger pointing.

### Specify the package name when generating the C header

If you don’t specify the fully-qualified class name, including the package name, when generating the C header with **javah**, the native function names won’t match what the JVM thinks they should be, and it won’t be able to reconcile the native methods with their implementations. Extraneous **UnsatisfiedLinkErrors** are a good indicator of this.

### Memory leaks when working with strings and arrays

When converting Java Strings and arrays into something the native C code can handle, remember that you need to free up memory when you’re done. Failure to do this will result in slow (or not so slow) memory leaks, especially if you reassign the pointers before releasing the memory!

### Multiple instances of the same library

The JVM will only load a given dynamic library once, even if you load it in several places in the Java code. This could cause a problem if you’re storing state information in your native code, or have initialisation code designed to be called once, and are expecting several Java wrapper classes to each get separate copies of the DLL. The DLL (or SO) will get loaded with whichever class gets loaded first, and any subsequent classes will get a reference to it. Keeping your native code as simple as possible will generally mitigate this problem, to a point.

## Finally

This article provides an introduction to the basics of using the Java Native Interface. By demonstrating a simple ‘Hello World’ program, it shows the steps required to create a JNI-based application. I’ve also touched on various other areas of the JNI system, and pointed out a few pitfalls.

I hope I’ve demystified JNI, and demonstrated that it can be a powerful tool. Like all versatile ways to shoot oneself in the foot, good planning is key to producing a useful system, but if done right, a JNI-based application can be simple and easy to work with. ■

## References

- [1] Sun Microsystems, Java Native Interface Specification, <http://java.sun.com/j2se/1.4.2/docs/guide/jni/spec/jniTOC.html>
- [2] Sun Microsystems JNI tutorial: <http://java.sun.com/docs/books/tutorial/information/download.html>
- [3] Sun Microsystems, *Java Native Interface: Programmer’s Guide and Specification*: <http://java.sun.com/docs/books/jni/index.html>

# Restaurant C++ and Pidgin Python

Pete Goodlife asks us to understand the idiosyncrasies of the languages we're working in.

I guess that I'm a typical Briton. I am a tea-drinking, fish-and-chip junkie who goes red thirty-five seconds after exposure to sunlight. I don't own any bulldogs, although I've been known to spout plenty of bull. And my grip of foreign languages is typically poor. I have what you'd politely call restaurant-French. Any slightly taxing conversation involves me gesticulating wildly, whilst repeating the same sentence louder and more

slowly until the conversant finally understands what I'm saying. Or politely pretends that they do. It works. At least, until they give up and talk back to me in English.

So, not a natural French-speaker, then.

Just how good is your mastery of your programming languages? Is it like my French – do you have restaurant Java, or tourist Python? Or do you really know the languages you use? Do you need phrase books and cheat sheets, or are you fluent, knowing the natural language idioms? Do you have to screw your eyes up and think hard when crafting code to make sure that it makes sense? Do other readers understand

what you write? Or do you write pidgin-code, translating idioms from a different language into the one you're writing?

A fluent French speaker doesn't think in English and convert their thoughts to French before speaking them. They think in French, and what they speak aloud comes naturally. There is no mismatch of idioms. There is no need for internal translation of the English idiom to the French idiom. To be truly effective in a programming language, to be able to craft Really Good Code, you have to operate in the same way.

There is a very real difference between fluent, idiomatic code, and pidgin-code. Stop for a moment and consider the number of ways that Listing 1a offends you. Count them all. How is Listing 1b better? Is it any better? And what language is each one written in, anyway?

When we're staring at code, we like to see clear structure and code patterns that we're familiar with – the natural idioms of that language. Certain code patterns are offensive – they naturally cause us to sit up, take note, and apply extreme caution. The mere sight of a `goto` invokes the gag reflex, when we see messy and inconsistent layout we feel bile rising, global variables cause an allergic reaction, and in the face of illogical and unmalleable structure we're gripped with the urge to run away very quickly. This kind of judgement is a part of what sets great programmers apart from the merely adequate ones. And we all want to be great programmers, don't we? How advanced do you think your internal quality meter is?

## Beauty is in the idiom of the beholder

It's interesting to note that our sense of 'beauty' is shaped by familiarity – by the prevalent idioms of the implementation domain. What constitutes natural and beautiful code differs from language to language. Idiomatic C code is quite a different beast from idiomatic Python. Listing 2 (2a, 2b, and

### PETE GOODLIFE

Pete Goodlife is a programmer who never stays at the same place in the software food chain. He has a passion for curry and doesn't wear shoes. Pete can be contacted at [pete@cthree.org](mailto:pete@cthree.org)



// Listing 1a - Ouch!!

```
bool ouch()
{
    if (do_something() == FAILED)
        goto fail_1;
    if (do_something_else() != OK)
        goto fail_2;

    return true;
    fail_2:
    tidy_up_second_thing();
fail_1:
    tidy_up_first_thing();
    return false;
}
```

// Listing 1b - Better?

```
bool better()
try
{
    do_something();
    do_something_else();
}
catch (...) { return false; }
```

2c) illustrate this. The Python code in Listing 2b is idiomatic, but it could have been written like Listing 2c – that listing is a more direct translation of the original C code. But it's not idiomatic Python, and it doesn't look or feel 'right' as a consequence. Listing 2c is more verbose, and consequently harder to comprehend and more likely to harbour bugs.

And that's just a really small example. (After all, small examples are idiomatic for magazine columns.)

// Listing 2a: Idiomatic C code

```
int list[] = { 1, 2, 3, 5, 8 };
for (int n = 2; n < 4; n++)
{
    do_something(list[n]);
}
```

// Listing 2b: Equivalent idiomatic Python code

```
list = [1, 2, 3, 5, 8]
for element in list[2:4]:
    do_something(element)
```

// Listing 2c: Equivalent non-idiomatic  
// Python code

```
n = 2
while i < 4:
    do_something(list[n])
    n += 1
```

Listing 1

Listing 2

## Listing 3

```
class ExampleMemberNames
{
    // Many programmers prefix member variables
    // with "_"
    int _common;

    // Also common is an "m_" prefix
    int m_member;
    int m_another;

    // Herb Sutter advocates a trailing underscore
    int sutter_;

    // Scott Meyers is the sensible advocate of
    // minimalism
    int meyers;
};
```

## Listing 4

```
class Foo
{
private:
    int thing;

public:
    // How would you name the ctor parameter?
    Foo(int thing_in) : thing(thing_in) {} // (1)
    Foo(int t) : thing(t) {} // (2)
    Foo(int thing) : thing(thing) {} // (3)
};
```

We become accustomed code that fits the natural idioms of the language we're using. Non-idiomatic code is most often what sets our internal alarm bells ringing. And rightly so. Idioms don't just look nice, they help us to write safe, correct code, avoiding the subtle pitfalls in the language. Like old wives' tales, there is a body of collected wisdom in our programming language idioms that is perilous to ignore.

Learning the specific idioms of a language are a rite of passage, and mark your mastery over the language, like a journeyman programmer becoming getting aquatinted with his tools.

## No idiom is an island

Important as they are, idioms are not sacred. Nor are idioms fixed and immutable. Fashion doesn't stand still; over time tastes change. Some classic programming idioms have dated: Hungarian Notation used to be a conventional, safe, and well-regarded practice. These days it is not merely passé, but socially unacceptable: the modern equivalent of software leprosy.

Idioms don't have all the answers, either. Multiple idioms compete over the same coding practice, and no one is necessarily right. Some aspects of code beauty are not clear cut and are continually the root of religious debates. For example, what is your opinion on the following; do you even care about them?

- Do you indent with spaces or tabs?
- If you use C-like languages, where do you put your braces?
- Do you advocate single-entry-single-exit functions?
- Do you prefer the functional coding style, where functions have no side effects?

## functional programming enjoys a renaissance

The functional coding idiom, in particular, is gaining a lot of mindshare recently as functional programming enjoys a renaissance and the industry

begins to learn how powerful and tractable some of the functional idioms are. Indeed, imperative languages like D are gaining functional language facilities and ways to express side-effect-free functions.

## Idiom idiocy

But sometimes the desire for elegant, beautiful, idiomatic code can trip us up. Well intentioned use of idioms can bite you. Here's a cautionary tale involving C++. Now, C++ idioms are particularly amusing. The oft-cited Perl mantra is There's more than one way to do it. C++ is like that for idioms – there's always more than one idiom for anything in C++, and you can bet that each has zealots who fervently believe that theirs is the Only Right Way To Do It.

One of the most basic, contentious, C++ idioms is the naming of member variables. Many of these idioms involve the subtle incursion of Hungarian Notation. See Listing 3 for examples.

Many C++ programmers prefix member variables by an underscore. However, this is dubious practice as the language standard reserves many identifier names beginning with an underscore. Class member variables are not actually one of the reserved cases, but this convention sails dangerously close to the wind. Also common is the `m_` prefix (where `m` stands for member). I've even seen the disgustingly cute `my_` prefix, e.g. `my_member`. Euch!

So what do the C++ experts do? Herb Sutter advocates a trailing underscore on member variable names. Andrei Alexandrescu sadly seems to follow his lead here. I have to admit that I have a personal dislike for this approach as the variable names read very strangely.

Scott Meyers is the sensible advocate of minimalism – he writes the variable name, the whole name, and nothing but the name (see the end of Listing 3). To my mind, this approach makes sense. If you need any extra indication of membership then you probably have code that is too hard to read – your function has too many parameters, or your class is too large. Fix that problem, don't mask it with silly variable names.

Why does this simple naming issue matter? Well, it shouldn't, until we combine the Meyers Minimal Member Moniker Mechanism with another idiom. When constructing a C++ class, members are given their initial values in the member initialisation list. There's another naming minefield here: three of the options are enumerated in Listing 4.

## the sensible advocate of minimalism – he writes the variable name, the whole name, and nothing but the name

They are only subtly different, are functionally equivalent, and each seems perfectly adequate. They are all common in modern C++ code. My workplace tends to adhere idiom 3, it's nicely symmetric and doesn't introduce another unnecessary name into the code.

Great.

Well, not quite. Idiom 3 has a hidden sting in its tail. Sure enough, in Listing 4 it works just as advertised. But consider what happens when you need some slightly more complex constructor logic:

```
Foo::Foo(int thing) : thing(thing)
{
    if (thing == 1)
        thing = 2;
}
```

What is the value of the `thing` member when a `Foo` is constructed with the value 1? It's 2, right? Well, no it isn't. It's 1. But how can that be, I hear you ask? The name `thing` inside the constructor is bound to the constructor parameter, not to the class' member variable. If you wanted to assign the member, you must write:

```
this->thing = 2
```



# JavaOne: JavaFX and the Future of Java

Peter Pilgrim shares his perspective of the JavaOne conference.

In May 2008, I got my fifth chance to visit California and attend the annual JavaOne Conference. Whenever I go to conferences like this I typically spend the majority of the time enjoying the technical sessions. However, it is an enjoyable experience meeting other delegates and other engineers from around world. The conference is truly international. Java and the number of people involved in its ecosystem are vast. There are people who specialise in everything from mobile device development to SOA. It is impossible to keep up with the momentum of 15000 delegates, all interested in different areas of computing.

At this year's JavaOne, the conference was split into several tracks: Cool Stuff, Desktop, Consumer Technologies, Java SE, Rich Media and Content, Open Source, Java EE, Tools and Script Languages, SOA and Enterprise Integration, Java ME and Next-Generation Web. The technical sessions and birds-of-a-feather talks matched at least one of these tracks.

## JavaFX: Declarative rich internet client programming

The most important sessions, in my view, involved improvements to Java's graphics, desktop and client side capabilities. Hence, I spent a lot of my time in sessions on the Rich Media and Content track, because I am particularly excited by the JavaFX family of technologies. I shared the belief with several other User Interface engineers that JavaFX stack was this year's great innovation. JavaFX is a statically typed declarative compiled language for building RIAs (Rich Internet Applications). JavaFX is the marketing term for a language created by a certain Mr. Christopher Oliver, when he was employed by SeeBeyond. SeeBeyond was later acquired by Sun in September 2005. Oliver called his creation F3 (Form Follows Function). A lot of developers, at the conference, commented that when they saw the syntax of JavaFX, it felt like a non-Sun technology. In fact, Charles Lowell of the Drunk and Retired Podcast, declared his surprise in his podcast. He thought that Sun must have hired an old Lisp programmer or a Haskell engineer to create the language. Lowell was not far from being correct. Oliver had a background in Lisp programming and his inspiration for F3 came partially from linguistics and literate programming.

What is the difference between JavaFX and normal Java? Other than favouring a declarative syntax for programming, JavaFX also compiles to

byte-codes that are interpreted and run on the JVM (Java Virtual Machine). The language supports four data types: Boolean, Integer, Number and

**In comparison JavaFX is avant-garde, but you still need to know what you are doing**

Object, whereas Java has primitives like char, short, int, long, float, double and, of course, Object. JavaFX makes event notification very easy through a built-in language concept of binding. Binding is a feature which allows an attribute of a target object to be notified when an attribute of a source object is changed. Building an equivalent Swing user interface using Java takes a lot of effort, code and know-how. In Java you would need to create your event class, listener and source. After that you would need to add logic to publish and subscribe to your custom event. In comparison JavaFX is avant-garde, but you still need to know what you are doing. The declarative syntax of JavaFX is a boon for understanding the structure of the user interface. The binding feature permits the user interface elements to be updated whenever the data model changes and vice versa. JavaFX can integrate with other APIs written in Java. So the world of open source and commercial frameworks and libraries is fully available. It is a little harder to call a JavaFX object from Java, but entirely possible.

JavaFX, then, is Sun Microsystem's spearhead. It is their attempt to gain more market share in the RIA sector and compete with the other two major players: namely, Microsoft's SilverLight and Adobe's Flex. Sun has wisely invested attention in the Java Runtime Environment over the past year. In the past, the big problem with Java has always been the size of the

## PETER PILGRIM

Peter is a Java EE software developer, architect, and Sun Java Champion from London. By days he works as an independent contractor in the investment banking sector. Peter can be contacted at [peter.pilgrim@gmail.com](mailto:peter.pilgrim@gmail.com)



## Restaurant C++ and Pidgin Python (continued)

otherwise you're just writing to a temporary variable that will shortly be thrown away. This is a subtle but nasty way to introduce obscure bugs into your codebase. So there you have a collision of idioms. Some idioms are bad for you! Hurrah for C++, and hurrah for idioms.

How could you alleviate this problem? There are many ways. For example, you could make the thing parameter const in the constructor implementation. But for built-in types passed by value, this isn't idiomatic! How else could you avoid it? (You could always choose to follow a different set of idioms. Or use a different language.)

## The moral of the story

It's important to consider the idioms of the language you're working in – and to gauge the beauty and quality of code against the familiar idioms it

should adhere to. Common language idioms have several important uses: they help to show the elegance, beauty, and artistry of a piece of code. They help you to write code that seems familiar and easy to work with, and they (usually) help you to avoid simple bugs. You can gauge your mastery of a language by how well you know its idioms.

It's particularly important to understand why these idioms exist. Learn to think in the programming language you're using, to think in terms of it idioms.

But don't blindly trust idioms. Idioms can be flawed. Always use your brain. Of course, if this seems like too much work, perhaps you should give up and produce boring, ugly code. Or learn to speak French properly instead. ■

download over the network. The runtime was too big to be transported across the wire to clients in comparison with Adobe's smaller Flash solution. Flash is available in 99.9% of desktop computers with a web browser according to some industry figures. The trouble was also that the Java Runtime Environment was very monolithic and not modular. Consequently, the cost of administration across client PCs was high, and obviously Sun and Java have some catching up to do. At JavaOne 2007, Sun announced the Consumer JRE, which would make significant in-roads in these deployment, administration and installation problems. The Consumer JRE includes many initiatives such as QuickStarter, a Java Kernel, a Deployment Kit and hardware accelerated graphics improvements. At this year's conference, Sun announced the early beta of JDK 6 Update N (the official name of Consumer JRE). It is available now for developers to test and experiment with. The new JDK also has a bonus, a great new look and feel cross-platform UI, called Nimbus as the default.

There were several conference special key notes to illustrate the new features of JavaFX. In one such presentation Sun demonstrated something truly amazing, the 'Video Ball'. This was a live demo of JavaFX playing scores of video players. Each video was mapped to the surface of a 3D sphere. Every video frame was also rotated in real-time in three dimensions around x,y and z axes. No longer could you say that Java was slow, apparently. In years gone by, Java has had a poor story when working with audio and video. Now with this Video Ball demonstration, all of our tails were up with excitement.

Adobe Flash is currently still the leader in web based video and audio and this technology is well proven with excellent sites like Youtube.com and Vimeo.com. These Web 2.0 applications would not be possible without the Flash Video encoding and decoding technologies, but now JavaFX (and Java) have at least a playback ability. It is called the Java Media Component API. Sun announced the public release would expected to be July 2008. Through this library Java will be able play audio and video content through codecs installed on the users machine. It is a clever way to get around the litigation and confusion surrounding codec licenses and lawyers, methinks. Sun announced at the conference that it had signed a deal with leading Video Codec provider, On2 to provide a cross platform media platform codec (On2 is also the company behind the Flash Video codec by the way). So this is wonderful news, for developers who hope to take advantage of Java's famed portability (write once run [almost] everywhere) with a cross-platform codec.

On the grapevine, I talked to many engineers, who expressed some of disappointment with Java at the moment. Some are interested in other compatible languages like Scala, created by Martin Odersky, and dynamic languages like JRuby and Groovy. For the JavaFX Freaks like myself,

## When are we going to see any tech conference when the product is released rather than hear about a product that will be released soon?

there were complaints that yet again this conference had produced not one single deliverable product in time. It was again all promises about product releases well into the future, by three, six or even twelve months down the line. In particular, the JavaFX SDK Preview 1.0 release was delayed until the summer of 2008 (so by the time you read this in ACCU, it will, fingers crossed, have already been released). This was frustrating to developers, I could share their pain. We all asked a simple question. When are we going to see any tech conference when the product is released rather than hear about a product that will be released soon? For those you are interested in mobile phone development then JavaFX Script for Mobiles will be

released in Spring 2009. On the positive side, clearly the makers of Java are definitely attempting pushing the boat out to reinvigorate Java on the Desktop again.

## What's Coming in Java 7?

Of course, there is more to Java than rich media or graphics and there was debate about the modularisation of the Java platform. It would appear that OSGi and the JSR 277 modularisation standard have made a peace of sorts, albeit unofficially. Java Specifications Requests are the names of proposing standards that are govern by the Java Community Process (I talked about the JCP in the previous Champions View series article #3). JSR 277 is a specification about introducing a standard module system into the Java platform. The related JSR 294 specifies superpackages and the concept of importing and exporting Java packages. Peter Kriens and the OSGi body were very much against the original specifications, because of

**there is more to Java than rich media or graphics and there was debate about the modularisation of the Java platform**

their non-involvement in the standard. OSGi is an existing standard for Java modularisation going back almost a decade. The OSGi side were pushing their de-facto initiative as being more mature and already proven. It would appear that the Glassfish application server project has signalled the peace between these two opposing forces, because it has decided to support both in version 3 of their development source code. Glassfish is the open source project of Sun's application server product.

And then, there was the unresolved closures in Java debate. At this point in time, it appears disappointingly that closures will not be making their entrance into the Java language any time soon. Closures are functional blocks that appear in other languages like Ruby, Scala and Groovy. Getting resolutions on closures is hard, because there are at least three competing papers floating around on the web: BGGA (Gilad Bracha, Neal Gafter, James Gosling and Peter von der Ahé), CICE (Concise Instance Creation Expressions by Joshua Bloch, Bob Lee and Doug Lea) and FCM (First Class Methods by Stephen Colebourne and Stefan Schulz). Each camp has its supporters and there is also another camp that believes adding closures would be detrimental to the language. Some folks even believe that developers should move to other languages with closures, Scala, and leave Java well alone. Personally, I would love to see closures make it into Java, if only to make writing control abstractions easier, because I have seen lots of commercial APIs that throw checked exceptions all over the place. Closures would enable me to write a simple library API to handle these cases. James Gosling had this to say in a recent blog about the history of the Java language:

Closures were left out of Java initially more because of time pressures than anything else. Closures, as a concept, are tried and true - well past the days of being PhD topics. The arguments are in the details, not the broad concepts. In the early days of Java the lack of closures was pretty painful, and so inner classes were born: an uncomfortable compromise that attempted to avoid a number of hard issues.

In relation to the possible (non-)inclusion of closures in the next version of Java, version 7, there was interest in other improvements for Java SE 7. I think everyone felt that there was much uncertainty about the features to include. Java is not the only language that runs on the JVM. Since Sun has invested in Ruby and very recently Python running on the JVM, respectively JRuby and Jython. They hired the key personnel to work full time on these projects. I think it is a very good guess that the dynamic invocation instruction will make an appearance inside the JVM 7. The invokedynamic byte code will support dynamic languages and allow the possibility of writing efficient and economic compilation techniques at run-time. Almost certainly a standard module system will appear in Java 7 with superpackages and a default repository. Whether if it will be OSGi compatible remains to be seen.

The concurrency library extensions, like the brand new TransferQueue collection (created by Doug Lea and Brian Goetz) and some innovations

from Cliff Click will make it into Java 7. The ability of Java to run on multi-core systems is very important and Sun will surely make any concurrency improvements available inside the Java SE core. In relational to concurrency, the improvements to I/O, in NIO2, will also make it into the Java 7. Lastly, I would expect the Java Media Components and the brand new Swing Application Framework to also make it in.

So when will we see Java 7? This is the bone of contention. Apparently, and very surprisingly, there is no official Java SE 7 JSR in existence at this time of writing, although Danny Coward, Sun's Java SE chief engineer and the specification lead is said to be working on it. Well, he has been working for 18 months already or so. So your guess is as good as mine, but I guesstimate late 2009.

## Enterprise Java

At this year's conference, there was less excitement on the enterprise development side. JPA 2.0 (Java Persistence API 2), EJB 3.1 (Entity Java Beans) were discussed, presented and, naturally, the Java Enterprise Edition 6. These standards are in public draft review or will be released for future debate. However, there was one talk that did interest me. It was JAX-RS, Java API for XML RESTful Services. This specification allows RESTful services to be created and declared on POJOs using annotations. In much the same way you can already define a SOAP/WSDL style web service on a methods belonging to Session Beans in EJB 3, the JAX-RS annotations will allow you to declare RESTful service endpoints. So this is exciting for enterprise development, because annotations make it easier to write these services. I suspect RESTful services will become popular inside companies that want to expose functionalities between divisions, departments and 'silos'. A RESTful service is great way to expose data to another department using standard Internet protocols and techniques.

## In summary...

Java has a great future on the desktop, providing that the rich media applications and libraries live up to the expectations. Adobe still have the lead on audio and visual content. On the other hand, Java has a ubiquitous virtual machine and it is proven. I think the Consumer JRE will be a big hit. If they can get the size of the install down and the launch time for a cold start Java[FX] applet to be short as it is for a Flash application then Sun is on to a winner. The JVM is available on mobile phones, desktops and inside servers. I think we are heading to a marvellous new age of development, where the engineer can dictate the technology for once. I think that Sun should also think about the 'Prosumer' market as well. These are the smarter engineers and media people who are starting to think about computing on the move. Asus did think about it years ago with its cheaper ultra mobile laptops and obviously Apple has its Air product on the high end. There is definitely a brand new market niche out there now and Java should be part of it as well. ■

## References

- [1] JavaFX <http://www.sun.com/software/javafx/>
- [2] JavaFX Script <http://www.sun.com/software/javafx/script/>
- [3] JavaFX Script Documentation <http://openjfx.java.sun.com/current-build/doc/index.html>
- [4] JavaFX Home Page to sign up for the private preview of the JavaFX SDK and to view demos <http://www.javafx.com/>
- [5] Open JavaFX <https://openjfx.dev.java.net/>
- [6] Latest JavaFX Script Compiler Release Milestone 3 <http://openjfx.java.sun.com/>
- [7] James Weaver's JavaFX Blog <http://learnjavafx.typepad.com/>
- [8] Create rich applications with JavaFX Script <http://www.ibm.com/developerworks/java/library/j-javafx/>
- [9] Planet JFX [http://jfx.wikia.com/wiki/Main\\_Page](http://jfx.wikia.com/wiki/Main_Page)

- [10] Drunk and Retired Podcast by Charles Lowel and Michael Coté <http://www.drunkandretired.com/podcast/>
- [11] David Herron on Scene graph API available under open source [http://weblogs.java.net/blog/robogeek/archive/2007/12/scene\\_graph\\_api.html](http://weblogs.java.net/blog/robogeek/archive/2007/12/scene_graph_api.html)
- [12] Alex Miller's frequently updated list of features proposed for Java 7 includes 150 links related to closures at the time of writing (Alex Miller, Pure Danger Tech) <http://tech.puredanger.com/java7>
- [13] James Gosling addressed the history of closures in Java in a recent blog post (On the Java Road, January 2008) <http://blogs.sun.com/jag/entry/closures>
- [14] Closures for the Java Language <http://www.javac.info/BGGA>  
Closures specification <http://www.javac.info/closures-v05.html>
- [15] FCM Closures specification [http://docs.google.com/View?docid=ddhp95vd\\_0f7mcns](http://docs.google.com/View?docid=ddhp95vd_0f7mcns) and see here FCM for a prototype [http://www.jroller.com/scolebourne/entry/fcm\\_prototype\\_available](http://www.jroller.com/scolebourne/entry/fcm_prototype_available)
- [16] CICE Closures specification [http://docs.google.com/View?docid=k73\\_1ggr36h](http://docs.google.com/View?docid=k73_1ggr36h) also see the CICE+ARM prototype <http://slm888.com/javac/>
- [17] Java 7 Development Home <https://jdk7.dev.java.net/>
- [18] Dr Cliff Click, Azul Systems, JavaONE Slides on Towards Scalable Non-Blocking Coding Style in Java [http://www.azulsystems.com/events/javaone\\_2008/2008\\_CodingNonBlock.pdf](http://www.azulsystems.com/events/javaone_2008/2008_CodingNonBlock.pdf)
- [19] Consumer JRE: <http://java.sun.com/developer/technicalArticles/javase/consumerjre>



## C++ Libraries and Tools to Simplify Your Life

- > **POCO C++ Libraries:** free open source libraries for internet-age cross-platform C++ applications
- > **POCO Remoting:** the easiest way to distributed objects and SOAP/WSDL Web Services in C++
- > **POCO Open Service Platform:** create high performance component-based, manageable and dynamically extensible applications in C++
- > and much more: Fast Infoset, WebWidgets, ...
- > available on many platforms—highly portable code
- > scalable from embedded to enterprise applications

  
applied informatics

Free Download and Evaluation: [appinf.com/simplify](http://appinf.com/simplify)

# Java Web Start

Paul Grenyer demonstrates an easy way to distribute Java apps.

Java. I spent years avoiding it. I felt it was inferior to the power of C++. I thought it was slow, clunky, the GUI was rubbish and that garbage collection was for wimps who did not know how to clean up after themselves or use smart pointers. Ok, so we all know I was wrong. And life being the way it is, being so outspoken about Java was sure to come and bite me and it did.

Since December I have been writing Java as part of my day job and I found I liked it so much I've even started using it for some of my own projects. I do not even miss Microsoft's Visual Studio. I have become very attached to Eclipse [1] and having code checked in real time, therefore negating a build stage, is very useful.

I have been so busy writing Java (and editing my new CVu column, 'Desert Island Books') that I have not written an article on anything else for quite some time. The editor of Overload has been nagging me for material, as has the new publications officer. I have also seen a few comments here and there about how poorly Java is served by the ACCU at present, but then with a strong history in C and C++ this is to be expected. However, my plan here is to redress the balance a little.

I'm spending most of my free time (not that I have a lot these days) working on a file viewer application that allows fixed length record files in excess of 4GB to be viewed without loading the entire file into memory. I wrote one of these in C++ (MFC) for a company I worked for a number of years ago. It worked well, but was a bit clunky and the user interface looked rubbish. I think they are still using it, but I'm not sure. I have had a few failed attempts to write it in C# recently, but it was not until I had a go in Java with its **JTable** and **TableModel** classes that I really made some progress.

The file viewer is a little way off being finished, but I am starting to think about package and deployment options. I want it to be easy and one of the applications I use in my day job uses Java Web Start [2] and it works really nicely. Sun describe Java Web Start as:

Using Java Web Start technology, standalone Java software applications can be deployed with a single click over the network. Java Web Start ensures the most current version of the application will be deployed, as well as the correct version of the Java Runtime Environment (JRE).

It sounds ideal for a constantly developing application that may be used by people all over the world on different operating systems.

As I sit down to write this article I have done no more than briefly read the Java Web Start documentation (so much for writing about what I know about – again!). I am intending to write an article about how to create applications and deploy them using web start by investigating it myself and writing down the steps as I go. I'll assume a reasonable familiarity with Java and Swing [4].

From reading the documentation it looks like I need to package my Java application in a JAR file so I'll look at how to do that and make the process easily repeatable using ANT.

## Java Web Start application

In order to test Java Web Start I need a simple Java application. Before getting stuck into writing such an application it is worth consulting the *Java Web Start Guide* [5] section on Application Development Considerations, which states:

Developing applications for deployment with Java Web Start is generally the same as developing stand-alone applications for the Java(TM) Platform Standard Edition. For instance, the entry point for the application is the standard:

```
public static void main(String[] argv)
```

However, in order to support Web deployment – automatic download and launching of an application – and to ensure that an application can run in a secure sandbox, there are some additional considerations:

- An application must be delivered as a set of JAR files. All application resources, such as files and images must be stored in JAR files; and they must be referenced using the **getResource** mechanism in the Java(TM) Platform Standard Edition.
- If an application is written to run in a secure sandbox, it must follow these restrictions:
  - No access to local disk.
  - All JAR files must be downloaded from the same host.
  - Network connections are enabled only to the host from which the JAR files are downloaded.
  - No security manager can be installed.
  - No native libraries may be used.
  - Limited access to system properties. The application has read/write access to all system properties defined in the JNLP File, as well as read-only access to the same set of properties that an Applet has access to.
- An application is allowed to use the **System.exit** call.
- An application that needs unrestricted access to the system will need to be delivered in a set of signed JAR files. All entries in each JAR file must be signed.

As expected, the application needs to be JARed. My file viewer application uses both resources and requires access to the local system so the JARs

```
import javax.swing.JFrame;
import javax.swing.JLabel;
public class HelloJavaWebStart
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Java Web Start!");
        JLabel label = new JLabel(
            "Hello, Java Web Start!", JLabel.CENTER);
        frame.add(label);
        frame.setSize(250,100);
        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Listing 1

## PAUL GRENYER

An active ACCU member since 2000, Paul is the founder of the Mentored Developers. Having worked in industries as diverse as direct mail, mobile phones and finance, Paul now works for a small company in Norwich writing Java. He can be contacted at paul.grenyer@gmail.com







will have to contain resources and be signed, but I want to start with something simpler first. All of this is covered in the Application Development Considerations.

I will start with a simple Java Swing application (Listing 1).

The above application creates a simple window with the title 'Java Web Start!' and a label 'Hello, Java Web Start'. The window is 250 by 100 pixels and visible. The easiest way to test the application is at the command line (unless of course you are using Eclipse):

```
javac HelloJavaWebStart.java
java HelloJavaWebStart
```

The window looks like Figure 1.

## JARing

The Sun website describes JAR files as follows:

The Java™ Archive (JAR) file format enables you to bundle multiple files into a single archive file. Typically a JAR file contains the class files and auxiliary resources associated with applets and applications.

The format of the command line parameters for the jar command is:

```
jar cf jar-file input-file(s)
```

The options and arguments used in this command are:

- The **c** option indicates that you want to create a JAR file.
- The **f** option indicates that you want the output to go to a file rather than to stdout.
- **jar-file** is the name that you want the resulting JAR file to have. You can use any filename for a JAR file. By convention, JAR filenames are given a **.jar** extension, though this is not required.
- The **input-file(s)** argument is a space-separated list of one or more files that you want to include in your JAR file. The **input-file(s)** argument can contain the wildcard **\*** symbol. If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

Putting the application into a JAR is therefore straight forward:

```
jar cf HelloJavaWebStart.jar
HelloJavaWebStart.class
```

However, to be able to just run the application from the JAR file, I need a manifest file to tell Java which **main** should be run by default. Obviously, the **HelloJavaWebStart** application only has a single **main**, but any number of classes can be included in a JAR and all of them could have their own **main** methods.

A manifest file is just a text file that, in this case, indicates which **main** to run:

```
Main-Class: MyPackage.MyClass
```

The text file must end with a new line or carriage return, otherwise the last line will not be parsed properly. Also, there must only be a single space between the colon and the start of the package or class name.

The **HelloJavaWebStart** application uses the default package, so only the name of the class is required:

```
Main-Class: HelloJavaWebStart
```

To incorporate the manifest file into the JAR add the **m** command line parameter and the name of the manifest file to the invocation of the JAR tool:

```
jar cfm HelloJavaWebStart.jar Manifest.txt
HelloJavaWebStart.class
```

On Windows and Linux (and MacOS X – Ed.) the application can now be run by simply double clicking the JAR file. The alternative is to run it from the command line:

```
java -jar HelloJavaWebStart.jar
```

## JARing with ANT

Having to repeatedly type the **javac** command followed by the **jar** command is both time consuming and error prone (not to mention irritating). Ant [3] is a build tool that can automate both. I won't go into the details of installing Ant, but the basic steps are:

1. Download and unpack Ant.
2. Make sure the Ant **bin** directory is in your platforms **PATH** environment variable.
3. Add **JAVA\_HOME** to your platform's environment variables and make sure it points to the location of your Java SDK installation (e.g. on Windows: **C:\Program Files\Java\jdk1.6.0\_06**).

Ant uses XML to describe builds. Every Ant build file must contain a project:

```
<project name="HelloJavaWebStart" basedir=".">
...
</project>
```

As shown above, every Ant project should specify its name and the base directory. The name is specified by the **name** attribute. The base directory is the directory where Ant will go looking for the files to be processed and is specified by the **basedir** attribute. Specifying a full stop tells Ant to look in the current directory. When an Ant build file is run, the tasks inside the project are executed.

The **javac** task is used to compile java files:

```
<javac srcdir = "${basedir}"/>
```

In its simplest form the **javac** task only needs to know where to look for the **.java** files to compile. This is specified by the **srcdir** attribute. When **\${basedir}** is read by Ant it is replaced by the value of **basedir** specified by the project. The **javac** task will look in the current directory for **.java** files and, by default, write the **.class** files to the same directory. In most cases you would want to specify a separate source and destination directory. Ant does of course allow this, but it is not necessary for this example.

The **jar** task is used to create JAR files:

```
<jar jarfile="HelloJavaWebStart.jar"
basedir="${basedir}"
manifest="manifest.txt"
includes="*.class"/>
```

The **jar** task needs to know:

- The name of the JAR file to create, specified by the **jarfile** attribute.
- The directory to look in to find the files to jar, specified by the **basedir** attribute.
- The location of the manifest file, specified by the **manifest** attribute.
- The types of files to include in the jar, specified by the **include** attribute. In the example above, specifying **\*.class** tells the jar task to include all **.class** files and nothing else.

```
<project name="HelloJavaWebStart" basedir=".">
  <javac srcdir="${basedir}" />
  <jar jarfile="HelloJavaWebStart.jar"
    basedir="${basedir}"
    manifest="manifest.txt"
    includes="*.class"/>
</project>
```

By default Ant build files are called `build.xml`. The complete Ant file for the `HelloJavaWebStart` looks like Listing 2 and should be saved to the same directory as `HelloJavaWebStart.java`.

To invoke Ant and build the application type the following at the command line:

```
ant
```

This will give something resembling the following output:

```
Buildfile: build.xml
[javac] Compiling 1 source file
[jar] Building jar: HelloJavaWebStart.jar
```

```
BUILD SUCCESSFUL
```

Ant is a very powerful build tool and does far more than I have described here. See the Ant documentation for details.

## Configuring a web server

Java Web Start applications can be hosted on almost any web server, but the server must be configured to support the JWS mime type.

### Apache

Apache [6] web server could not be easier to configure for Java Web Start:

1. Open the `mime.types` file from the Apache `conf` directory.
2. Add the line:

```
application/x-java-jnlp-file JNLP
```

to the end of the file and save.
3. Restart apache.

### Microsoft Internet Information Server (IIS)

The standard IIS 5.0 that come with Windows XP appears to support JWS by default, although it is not listed on the 'Mime Types in IIS' page [8]. If you do find you need to add the JNLP mime type, follow these steps:

1. In the IIS snap-in select the website to add the mime type to and bring up the Properties dialog box.
2. Select the HTTP Headers tab.
3. Under MIME Map, click the File Types tab and select New Type.
4. Type `.jnlp` in the Extension field and `application/x-java-jnlp-file` in the Content Type field, and then click OK.

## Publishing an application

Once a web server has been configured, all that is left is to publish the JAR file to the server, along with a Java Network Launch Protocol (JNLP) file that describes how to launch the application and a hypertext link to that same JNLP file. JNLP files can be very simple or very involved. Listing 3 is the simplest possible JNLP file that will get the `HelloJavaWebStart` application to launch:

- The **codebase** attribute specifies the base location for all relative URLs specified in **href** attributes in the JNLP file.
- The **information** element contains other elements that describe the application and its source. The **title** and **vendor** elements are the bare minimum.
- The **resources** element describes all the resources that are needed for an application.

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="http://myserver/apps">
  <information>
    <title>JavaWebStart</title>
    <vendor>Paul Grenyer</vendor>
  </information>
  <resources>
    <j2se version="1.2+"/>
    <jar href="HelloJavaWebStart.jar"/>
  </resources>
  <application-desc/>
</jnlp>
```

- The **j2se** element specifies what version of Java to run the application with.
- The **jar** element specifies the JAR file to launch.
- The **application-desc** element denotes this is the JNLP file for an application.

A simple hypertext link is needed to launch the application. Assuming that the JAR file and a JNLP file called `HelloJavaWebStart.jnlp` have been copied to `http://myserver/apps` the following link will allow the application to be launched:

```
<a href="http://myserver/apps/
  HelloJavaWebStart.jnlp">Hello Java Web Start</a>
```

That completes a simple example of creating and deploying a Java Web Start application. Yes, it really is that easy. However, if you need to access resources such as images it does get a little more complicated, but not much.

## Retrieving resources from JAR files

As mentioned previously, any resources used by a Java Web Start application must be included in the JAR file. The Java Web Start Application Development Considerations states the following:

Java Web Start only transfers JAR files from the Web server to the client machine. It determines where to store the JAR files on the local machine. Thus, an application cannot use disk-relative references to resources such as images and configuration files.

All application resources must be retrieved from the JAR files specified in the resources section of the JNLP file, or retrieved explicitly using an HTTP request to the Web server. Storing resources in JAR files is recommended, since they will be cached on the local machine by Java Web Start.

The following code example shows how to retrieve images from a JAR file:

```
// Get current classloader
ClassLoader cl =
    this.getClass().getClassLoader();

// Create icons
Icon saveIcon = new ImageIcon(
    cl.getResource("images/save.gif"));
Icon cutIcon = new ImageIcon(
    cl.getResource("images/cut.gif"));
...
```

The example assumes that the following entries exist in one of the JAR files for the application:

```
images/save.gif
images/cut.gif
```

To retrofit this on the `HelloJavaWebStart` application I need an image (the ACCU [7] logo will do nicely) in a subdirectory to the directory where

```
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class HelloJavaWebStart extends JFrame
{
    private HelloJavaWebStart()
    {
        this.setTitle("Java Web Start!");

        ClassLoader cl =
            this.getClass().getClassLoader();
        ImageIcon image =
            new ImageIcon(cl.getResource(
                "images/accu_logo.gif"));
        JLabel label =
            new JLabel(image, JLabel.CENTER);

        this.add(label);
        this.setSize(250,100);
        this.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE );
    }

    public static void main(String[] args)
    {
        new HelloJavaWebStart().setVisible(true);
    }
}
```

HelloJavaWebStart.java is, called images. The application also needs slightly refactoring so that an instance is available to call `getClass` on and the label text needs to be replaced with the image (Listing 4).

It is important that paths to images use forward slashes, otherwise they will not be found. The path is the relative path in the JAR file. If you use Ant to build the new **HelloJavaWebStart** application, but run the `.class` file or run it from Eclipse:

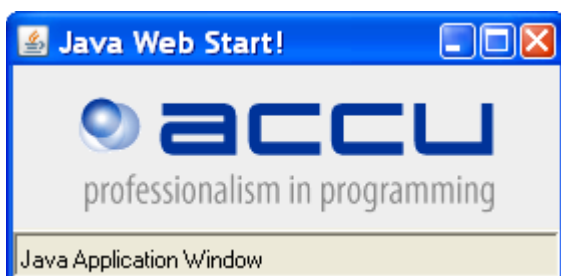
```
java HelloJavaWebStart
```

all is well. However, if you try and run the JAR file an exception is thrown as the image cannot be found. This is because the image has not been included in the JAR file. To do this the `jar` task must be modified to include the image:

```
<jar jarfile="HelloJavaWebStart.jar"
    basedir="${basedir}"
    manifest="manifest.txt"
    includes="*.class, images/*.*/>
```

As you can see above, `images/*.*` has been added to the `include` attribute. This tells Ant that as well as all the `.class` files, it should include all the files in the `images` subdirectory. Run Ant again and launch the application via the JAR file locally and then copy it to the web server and try it there (Figure 2).

So adding a static image to the JAR was not that bad. What is more interesting is allowing the user to select the image at runtime.



```
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class HelloJavaWebStart extends JFrame
{
    private HelloJavaWebStart()
    {
        this.setTitle("Java Web Start!");

        final JFileChooser chooser =
            new JFileChooser();
        if (chooser.showOpenDialog(this) !=
            JFileChooser.CANCEL_OPTION)
        {
            ImageIcon image = new ImageIcon(
                chooser.getSelectedFile()
                    .getAbsolutePath());
            JLabel label = new JLabel(
                image, JLabel.CENTER);
            this.add(label);
        }

        this.setSize(250,150);
        this.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE );
    }

    public static void main(String[] args)
    {
        new HelloJavaWebStart().setVisible(true);
    }
}
```

## Allowing local access

As mentioned previously, if a Java Web Start application wants to access local resources, such as the file system, it must be digitally signed and the JNLP file configured to allow it. Let's start by modifying the **HelloJavaWebStart** application to open a file chooser dialog and allow the user to select the image that is loaded (Listing 5).

When this application is built, with the Ant build file (removing the JARing of the images directory as it is not needed for this example), and run locally it allows the user to select and display an image from the hard disk. If the JAR file is deployed to the web server and the same test run, the error "access denied" is given.

This error is caused by the lack of security permissions in the JNLP file. These can be added by modifying the JNLP file (Listing 6).

Running the application from the web server now gives the error "Unsigned application requesting unrestricted access to system."

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="http://myserver/apps">
    <information>
        <title>JavaWebStart</title>
        <vendor>Paul Grenyer</vendor>
    </information>
    <resources>
        <j2se version="1.2+/">
        <jar href="HelloJavaWebStart.jar"/>
    </resources>
    <application-desc/>
    <security>
        <all-permissions/>
    </security>
</jnlp>
```

For this to work the JAR file must be signed using the Java SDK keytool and jarsigner tools.

Open a command prompt and move to the directory holding the source files for **HelloJavaWebStart**. Then follow these steps:

1. Create a key store, called **myKeystore** with the alias "myself" by entering the following command and entering the required information at the prompt:

```
keytool -genkey -keystore myKeystore
        -alias myself
```

This should give the following output and create a file called **myKeystore**:

```
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Paul Grenyer
What is the name of your organizational unit?
[Unknown]: Marauder
What is the name of your organization?
[Unknown]: Marauder
What is the name of your City or Locality?
[Unknown]: Norwich
What is the name of your State or Province?
[Unknown]: Norfolk
What is the two-letter country code for this unit?
[Unknown]: UK
Is CN=Paul Grenyer, OU=Marauder, O=Marauder,
L=Norwich, ST=Norfolk, C=UK correct?
[no]: yes
```

```
Enter key password for <myself>
(RETURN if same as keystore password):
```

2. Create a certificate by entering the following command and the password entered in the previous step:

```
keytool -selfcert -alias myself
        -keystore myKeystore
```

3. To check that the key store and certificate have been created, enter the following command using the same password as before:

```
keytool -list -keystore myKeystore
```

This should give output similar to the following:

```
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

myself, 18-May-2008, PrivateKeyEntry,
Certificate fingerprint (MD5): 77:72:06:EC:
18:2F:00:85:8E:E8:A8:EE:74:69:F9:EF
```

4. Finally, to sign the JAR file enter the following, and the same password:

```
jarsigner -keystore myKeystore
HelloJavaWebStart.jar myself
```

which should give the following output:

```
Enter Passphrase for keystore:
Warning:
The signer certificate will expire within six months.
```



Figure 3

Now copy the JAR file over and try the application via the web server again. You should be presented with a dialog similar to Figure 3.

Click Run to launch the application with full access to local resources.

This is not quite the end of the story though. Although the key store and certificate only need to be created once, signing the JAR file needs to be done every time the JAR file is deployed, so should be part of the Ant build file. Ant has a **signjar** task that does this (Listing 7).

```
<project name="HelloJavaWebStart" basedir=".">
  <javac srcdir="${basedir}" />
  <jar jarfile="HelloJavaWebStart.jar"
      basedir="${basedir}"
      manifest="manifest.txt"
      includes="*.class" />
  <signjar jar="HelloJavaWebStart.jar"
          keystore="myKeystore"
          alias="myself"
          storepass="secret" />
</project>
```

Listing 7

- The **jar** attribute specifies the JAR file to sign.
- The **keystore** attribute specifies the key store to use.
- The **alias** attribute specifies the alias to use.
- The **storepass** attribute specifies the password for the key store.

The output from the new task looks like this:

```
[signjar] Signing JAR: HelloJavaWebStart.jar to
HelloJavaWebStart.jar as myself
[signjar]
[signjar] Warning:
[signjar] The signer certificate will expire
within six months.
[signjar] Enter Passphrase for keystore:
```

Although it appears from the output that the password is requested, it is in fact automatically entered by the **signjar** Ant task.

The *Java Web Start Guide* section on Application Development Considerations reminds us that:

Note that a self-signed test certificate should only be used for internal testing, since it does not guarantee the identity of the user and therefore cannot be trusted. A trustworthy certificate can be obtained from a certificate authority, such as VeriSign or Thawte, and should be used when the application is put into production.

## Conclusion

So here we are at the end. As a learning experience, getting to grips with Java Web Start is actually quite straightforward. Of course there is plenty more that can be configured in the JNLP files, especially in terms of



# Desert Island Books

Paul Grenyer introduces Allan Kelly's essential reading.

Allan Kelly is another ACCU stalwart. He has an opinion on everything and, often irritatingly, he is usually right. The moment I met Allan Kelly will be imprinted on my mind forever. It was at the 2002 ACCU conference at the Motor Museum. I had been an ACCU member for about a year and generally made a lot of unhelpful noise. As I shook his hand, Allan looked me in the eye, grinned and said 'So you're Paul Grenyer.' That rather set the tone for the conference.

In recent years I have spent many an evening drinking alcohol with Allan and we even worked together for a brief period in 2007. I don't want to turn this column into a which-Pink-Floyd-album-is-the-best debate, but I have to say (sorry Allan), it isn't *Dark Side of the Moon* either.

## Allan Kelly

I guess I am not the only person who has listened to Desert Island Discs and wondered what eight records I would take. However I like to think I have come a lot closer to the situation than many might.

A few years ago I sailed from Montreal to London; well, sailed might give you the wrong idea. The ship was 250m long and carried 2,500 containers – TEU in the trade – and I arrived not so much in London as at an ugly container port, Thamesport, east of the city.

Why did I do this? Not to save airfare (it was more expensive) but for fun, or rather to say 'I have sailed the Atlantic' – the way people would in olden days. As there were the 22 or so crew, the Captain and I had plenty of reading time on the seven-day journey.

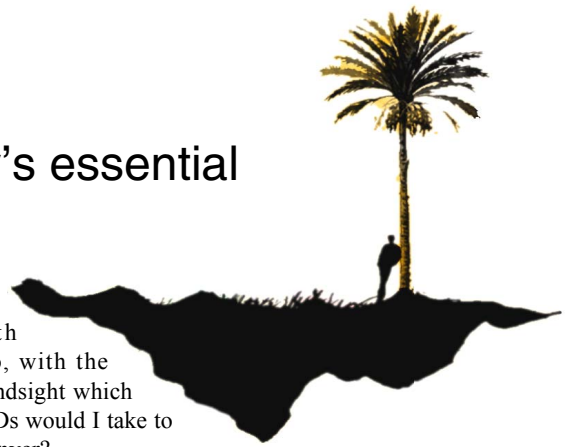
True, in the middle of the journey, staring out at the grey Atlantic without another ship in sight I did wonder: what am I doing here? – but by then it was too late. In retrospect it was worth it for the experience. If nothing else sailing up the frozen St Lawrence River, breaking ice as we went made it all worth while.

I can't remember exactly the books and CDs I took but I do remember taking about 12 CDs. While the MV Canmar Pride was a comfortable ship it was no desert island. Neither was there much prospect of finding one in

the North Atlantic. So, with the benefit of hindsight which books and CDs would I take to the Caye Grenyer?

As most readers probably know I spend little time programming these days and lots of time managing programmers. So no apologies for choosing more managerial books. The challenge with both books and CDs is to choose ones you know you will like (tending towards ones read before) and those where you will find something new (tending towards ones never read before).

First has to be *The Fifth Discipline* by Peter Senge. To many this may be an abstract book which discusses ideal work environments, but I loved this book and couldn't put it down when I read it. It really inspired me to see the world differently. The book described what learning organizations are,



## What's it all about?

Desert Island Disks is one of BBC Radio 4's most popular and enduring programmes:

<http://www.bbc.co.uk/radio4/factual/desertislanddiscs.shtml>

The format is simple: each week a guest is invited to choose the eight records they would take with them to a desert island.

I've been thinking for a while that it would be entertaining to get ACCU members to choose their Desert Island Books. The format will be slightly different from the Radio 4 show. Members will choose about 5 books, one of which must be a novel, and up to two albums. The programming books must have made a big impact on their programming life or be ones that they would take to a desert island. The inclusion of a novel and a couple of albums will also help us to learn a little more about the person. The ACCU has some amazing personalities and I'm sure we only scratch the surface most of the time.

Each issue of CVu will have someone different. If you would like to share your Desert Island Books please email me: [paul.grenyer@gmail.com](mailto:paul.grenyer@gmail.com).

## Java Web Start (continued)

security and versioning, and there's plenty more that Ant can do, but that's outside the scope of this article.

If you are still not sure of the advantage of Java Web Start over the usual installer method of deploying applications, let me describe a situation I have been in on many occasions:

It is 7am in the morning and I'm in a deserted office moving from PC to PC installing the latest release of the software. It gets to 9am and everyone starts arriving. But it is ok, all the machines have been updated.

So I go back to the development office to find the phone ringing. As I listen to the user explaining the problem with the new release I realise it's an easy fix, but will require another install to every machine. The whole business grinds to a halt while the bug is fixed and then gets delayed a further two hours while everyone's machine is updated.

If I had used Java Web Start I would not have had to get into the office at 7am to do the release and any post release problems could be deployed more quickly and easily once fixed. In fact releasing involves building some signed JARs, copying them to the webserver and asking everyone to restart.

Easy! ■

## References

- [1] Eclipse: <http://www.eclipse.org/>
- [2] Java Web Start: <http://java.sun.com/products/javawebstart/>
- [3] Ant: <http://ant.apache.org/>
- [4] Java Swing: [http://en.wikipedia.org/wiki/Swing\\_\(Java\)](http://en.wikipedia.org/wiki/Swing_(Java))
- [5] Java Web Start Guide: <http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html>
- [6] Apache: <http://httpd.apache.org/>
- [7] ACCU: <http://www.accu.org>
- [8] Mime Types in IIS: <http://technet.microsoft.com/en-us/library/bb742440.aspx>

how they operate and how you can start to build one. As many readers will know, I see the whole of software development as learning, so it follows naturally that the best teams and organizations are ones that learn. If you ever wonder what is beyond coding, read this book: it deserves reading and re-reading.

Next is Jim McCarthy's *The Dynamics of Software Development*. For me this book deserves far more attention than it ever got: it came out about the same time as books from Steve Maguire and Steve McConnell, and was overshadowed by them. In retrospect we would see it as an 'Agile' book today, but in 1995 the term had yet to be invented. I would love to have the time to re-read it, see how well it has stood the test of time and discover new lessons.

When I first read the book I was working on a Railtrack privatisation project conforming (supposedly) to ISO 9000 standards. The project can best be described in two words: Death March. The contrast with McCarthy's advice could not have been greater. Advice like Cycle Rapidly, Establish a Shared Vision, Nobody Reaches the Zero Defect Milestone Until Everybody Does might seem like convention wisdom today – at least in Agile circles – but in the 1995/6 ISO 9000 world people thought these ideas were madness. Nobody was ever doing to tell John Major 'You can't privatise the railways because the software isn't ready?', so the Government would spend what ever they needed to and the subcontractor pointed to their ISO 9000 badge to show they were following 'best practice'.

I learned two invaluable lessons on that project: some projects are just better off not done, and inside every big project is a small one trying to get out.

Back in my undergraduate days I had a lecturer who relentlessly sung the praises of Gödel, Escher, Bach. I was (obviously) very impressionable because a few years later I got the book and attempted to read it. Technically I succeeded, I read it from cover to cover, but, it took me over four years with lots of time-outs and other books in between. Hopefully, my time on the island will give me time to read it properly this time.

That leaves me one serious book still to pick. I'm ready to be adventurous here and grab something I haven't read before. Since Jez managed to rescue a PC with a development environment I'm going to assume I can too. As with so many people at the ACCU conference I got the functional programming buzz so a book on Haskell is very tempting but I don't know which one to pick. So, since I've been meaning to learn Python and web development forever I'm going to grab a Python web development book. I hear good things about Django so I'm going to take a risk, grab *Pro Django: Web Development Done Right* and hope that between this book

and the online docs I can master both Python and Django. With a bit of luck I'll develop some really useful application which, after my rescue, I'll be able to sell for a couple of million.

Now a really difficult choice: my novel. Between GEB, learning Python and Django and not forgetting surviving and escaping I'm going to be busy on this island. I'm going to need to be able to relax and mentally escape so I need a good novel.

My novel has to be good, deep, fresh and lasting. Two books spring to mind immediately and, as fate would have it they sit next to each other on my bookshelf: *Catch-22* and *The Ragged Trousers Philanthropists* (actually originally published in 1914). Another odd coincidence is that of the few hundred books on my shelf these are among the very few which I have read more than once. Given my heavyweight selection elsewhere I'm going to have to plump for *Catch-22*: I'm going to need something 'humorous' on the island. After I've read it once I intend to try and read it in the correct chronological order.

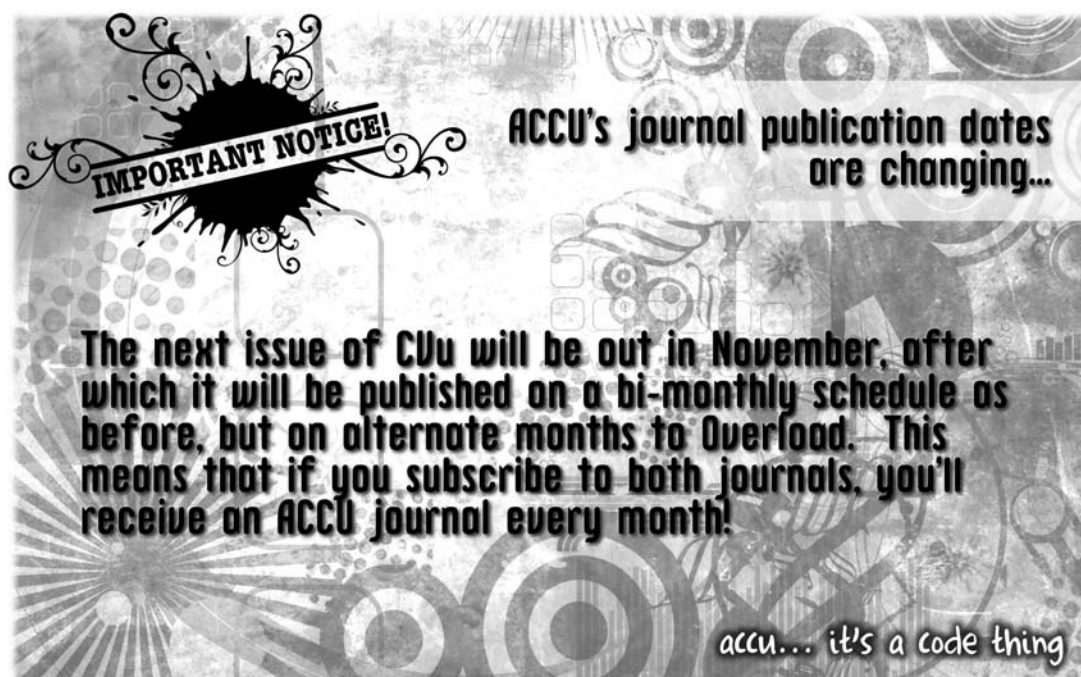
Now things get hard, two 'albums' – in quotes because I'm not sure Herr Mahler ever recorded an 'album', some symphonies, yes, but an album... This is really hard because in my world there are three types of music: classical (including opera), jazz (including blues) and everything else falling under the broad category of rock/pop. This last category includes everything from early Clash, through Elvis (Costello) to modern Ministry of Sound Annuals.

But I can only take two of three. Although I listen to more jazz than anything else these days I simply can't choose which jazz recording I would take: Miles Davis *Kind of Blue*, Charles Mingus' *Criss-Cross*, or Guy Barker's *Soundtrack* to remind me of London? But what of Shorter, Monk, Tracey, Hancock, Methany, and-and-and...no, I can't decide so none of them will come.

From the sinking ship I will save Stravinsky's *The Rite of Spring*, which fortunately in my collection comes with Petrouchka—so I won't forget all my Russian. (London Symphony Orchestra conducted by Rafael Frubeck de Burgos, 1989, Collins Classics).

The final choice is also easy, but it means leaving behind New Order's *Substance 1987*, a great album and the theme track to my late teens and early twenties. Instead I grab the album that really opened my eyes to rock music, an album that is only a few years younger than myself but does not age, one that has always been there for me, one that sounds as new today as it did when I first heard it: *Dark Side of the Moon* by Pink Floyd.

Next issue: Steve Love picks his desert island books.



**IMPORTANT NOTICE!**

**ACCU's journal publication dates are changing...**

**The next issue of CVu will be out in November, after which it will be published on a bi-monthly schedule as before, but on alternate months to Overload. This means that if you subscribe to both journals, you'll receive an ACCU journal every month!**

*accu... it's a code thing*

# Regional Meetings

A round-up of the latest ACCU regional events.

## ACCU London

Report from Steve Love (steve@essennell.co.uk)

Thursday 19th June – ‘Visual Studio 2008 and .Net 3.5’, a talk by Michael Taulty.

In spite of a smaller turn-out than was expected for this talk, Michael Taulty from Microsoft gave an interesting and detailed talk about some of the features in the latest edition of Microsoft Visual Studio and the new .Net Framework. It seems there is quite a lot of new stuff in VS 2008, so Michael chose – perhaps a little predictably – to talk mainly about the LINQ (Language Integrated Query, for those who’ve not yet encountered it) technology, a flagship feature of .Net 3.5, C# v3 and VB.Net.

In essence, LINQ is a data querying language, similar in syntax and approach to SQL, which provides a unified syntax for filtering data from a number of different sources, for example an SQL database, an XML document, or in-memory objects and collections. Michael gave examples of each of these three approaches, highlighting the fact that the syntax was almost identical between them. In fact, any data can be made a source for LINQ if the necessary adapter is provided. Michael also demonstrated how the actual mechanics of getting the data differs between the in-memory provider – which merely iterates over a collection (actually an IEnumerable for those .Net’ers reading) – and the SQL provider which uses a new IQueryable interface.

Most of the presentation wasn’t about the syntax of LINQ itself, however; Michael first took us through the new language features which enable LINQ to operate. These features include anonymous types, automatic type inference, lambda expressions and partial- and extension-methods. Without these features, LINQ wouldn’t be possible, but they have all been exposed separately as language features so all programmers can take advantage of them.

Michael was animated and knowledgeable, and used a mixture of slides and code (actually writing code as he talked, which was quite a feat!) to demonstrate the features. It also allowed him to show off some of the GUI effects of – I presume – Windows Vista by flicking back and forth between Visual Studio itself and the presentation. The spinning cube of different desktops was particularly exciting.

In usual ACCU London tradition, we repaired to the pub afterward to discuss, among other things, the usefulness of LINQ, and the language features which enable it, and why ironing is such an important – if controversial – activity.

Thanks go to Michael for giving such an interesting and enlightening talk which certainly provoked much discussion, to 7city for hosting the talk and providing a room, and to Allan Kelly and James Slaughter for organising another successful evening.

## References

LINQ: <http://msdn.microsoft.com/en-us/netframework/aa904594.asp>

Michael’s Slides are available from his blog at <http://miketaulty.com>

## ACCU Oxford

Report by Jim Hague (jim.hague@acm.org)

The OX postcode contains more ACCU members than any other single UK postcode, so it seemed a pity to me that there weren’t any Oxford regional events.

After being idiot enough to muse about this to a few people at the 2008 Conference, I was gently encouraged to adopt the Open Source approach to the problem – if you don’t like something, fix it yourself.

So, the inaugural Oxford group meeting was held at the West Oxford Community Centre on June 25th. To allow time for a little discussion on the direction the group should take, we had two short talks timetabled. I resurrected a talk on organising cross-platform source code bases previously given at Conference in 2003, and Phil Armstrong embarked on a session on type inferencing with pick-it-up-as-we-go Haskell. Unfortunately technical problems at the laptop/projector interface meant the session was unfinished by the time we had to leave the meeting room and adjourn to the Watermans on Osney Island.

A second meeting on July 30th has been announced on accu-general, where we will retry Phil’s session and hear from Chris Jefferson, BSI C++ committee member, on the latest on what’s likely to be in C++0x.

After that we’ll be taking a break for August, but look to return at the end of September with a full length talk. At the time of writing I’m hoping this will be a look at the IBM zSeries mainframe world, and Cobol’s niche.

It’s early days for the Oxford group, but we’re under way. Keep your eyes on ACCU-general for news of future events.

## ACCU North East

Report by Ian Bruntlett (ianbruntlett@hotmail.com)

The meeting started off quietly, with cheese rolls and drinks readied in time. There were about 7 of us with differing levels of programming confidence. ACCU NE members tend to have dual nationality, visiting ACCU NE on the third Saturday of the month, in Contact’s building (see [http://accu.org/index.php/accu\\_branches/accu\\_ne](http://accu.org/index.php/accu_branches/accu_ne) for more details), and visiting the Tyneside LUG (Linux User Group, <http://www.tyneside.lug.org.uk>) on the first Saturday of the month in the Discovery Museum, just off the Westgate Road in Newcastle.

We installed ghc (the Glasgow Haskell Compiler) on a Kubuntu box (upgraded from Ubuntu earlier in the meeting) and an OpenSuSE box, and experimented with doing the usual strange functional language things – very concise quick sorts and recursive mathematical functions.

Later, Alex Kavanagh showed some production Perl code and explained some of the issues in writing Perl for production.



## Write for us!

C Vu and Overload rely on article contributions from members. That’s you! Without articles there are no magazines. We need articles at all levels of software development experience; you don’t have to write about rocket science or brain surgery.

What do you have to contribute?

- What are you doing right now?
- What technology are you using?
- What did you just explain to someone?
- What techniques and idioms are you using?

If seeing your name in print isn’t enough, every year we award prizes for the best published article in C Vu, in Overload, and by a newcomer. For further information, contact the editors: [cvu@accu.org](mailto:cvu@accu.org) or [overload@accu.org](mailto:overload@accu.org)

# Code Critique Competition 53

Set and collated by Roger Orr.



Please note that participation in this competition is open to all members, whether novice or expert. A book prize is awarded for the best entry.

Readers are also encouraged to comment on published entries, and to supply their own possible code samples for the competition (in any common programming language) to [scc@accu.org](mailto:scc@accu.org).

## Last issue's code

I am having problems getting a template to work. The code below is supposed to print out the range of the data points, but the range function isn't producing the right answer for the `WDatum` class. Someone told me it was because I needed a virtual destructor – but that made it worse. I've even got rid of all the compiler warnings I had. Can you help me understand what's gone wrong?

Can you help answer the question? The code is shown in Listing 1.

## Critique

**Robert Jones <[robertgbjones@gmail.com](mailto:robertgbjones@gmail.com)>**

The easy thing about this code critique is the fix. Simply change the type of the iterator variable in the for-loop of the templated range function from `Datum` to `T`.

What's more interesting is what's going on. In the case of instantiating the range function for type `WDatum`, the iterator of type `Datum` is iterating over an array of objects of type `WDatum`, so incrementing the iterator (pointer) will leave you pointing at a place other than the next array element.

It just so happens that:

- `WDatum` is exactly one float larger than `Datum`, so the first increment probably leaves you pointing at part of `WDatum` just after the `Datum` part of `WDatum`, but only probably!
- `WDatum` is twice the size of `Datum`, so the termination condition is met for arrays with an even number of elements.
- The range function is called with an even number of elements.

This is a nice example of how 'undefined behaviour' can mean 'very nearly works'.

Adding the virtual destructor changes the object layout again, so all bets are off. At some point as we increment through the array we are likely to interpret vtable data as a `float`, which is likely to produce gibberish. In addition, the loop termination condition may never be satisfied, unless the `sizeof(WDatum)` just happens to be a multiple of the `sizeof(Datum)`.

**Charles Bailey <[charles.bailey@igence.com](mailto:charles.bailey@igence.com)>**

Fixing code so that it compiles without warnings is often a good compass but as this code shows, it shouldn't be the ultimate destination.

`main` and the `Datum` and `WDatum` classes are simple enough, so the template function is the natural place to start looking for the bug.

Unfortunately, the expression statement `++it` is where the bug is being triggered. I was going to say that that is where the bug is, but we've all seen so many thousands of loops with just such a simple increment in the last position that it seemed a little unfair to single it out as a buggy statement.

What actually sticks out like a sore thumb in the template function is the use of the `Datum` identifier in the template class. When writing templates

it is always a good idea to minimise the requirements that you put on any template parameters. This makes the template as generic as possible and improves the chances that it can be

```
// Simple datum
class Datum {
    float payload;
public:
    Datum( float value = 0 )
        : payload( value ) {}
    float getValue() const
    { return payload; }
};

// Weighted datum, simple by default
class WDatum : public Datum {
    float weight;
public:
    WDatum( float value = 0, float weight = 1 )
        : Datum( value ), weight( weight ) {}
    float getWeight() const
    { return weight; }
};

// Return range (max - min) of data
template <typename T>
float range( T * begin, T * end )
{
    float top = 0, bottom = 0;
    for ( Datum *it = begin; it != end; ++it )
    {
        float v = it->getValue();
        if ( !top && !bottom )
            top = bottom = v;
        else if ( top < v )
            top = v;
        else if ( bottom > v )
            bottom = v;
    }
    return top - bottom;
}

#include <iostream>
int main()
{
    static const int count = 4;
    Datum data[count] =
        { 1.3f, 1.2f, 1.4f, 1.7f };
    WDatum wdata[count] =
        { 1.3f, 1.2f, 1.4f, 1.7f };
    float drange = range( data, data+count );
    float wrange = range( wdata, wdata+count );
    std::cout << "range (expect 0.5)\n"
        << "Datum " << drange << "\n"
        << "WDatum " << wrange << std::endl;
}
```

Listing 1

## ROGER ORR

Roger has been programming for over 20 years, most recently in C++ and Java for various investment banks in Canary Wharf and the City. He joined ACCU in 1999 and the BSI C++ panel in 2002.

He may be contacted at [rogero@howzatt.demon.co.uk](mailto:rogero@howzatt.demon.co.uk)





usefully reused, which is, after all, the whole point of writing templates in the first place.

With the power of psychic debugging, we can see that `range` used to be a non-template function operating on `Datum` pointers and was converted to a template so that it could be used with `WDatum` as well. During the conversion the use of the `Datum` type in the `for` loop was missed and it is this that causes the problem.

Because `WDatum` is derived from `Datum`, a pointer to `WDatum` can and will be converted to a pointer to `Datum` without warning. This conversion is correct and works fine. The gotcha here is that `WDatum` and `Datum` have different sizes. This means that in an array of `WDatum`, each element is further away from the previous element than in an array of `Datum`. Pointer arithmetic just uses the static type of the given pointer. When you do `++it` where `it` is a pointer to `Datum`, it will be incremented to point to where the next `Datum` would be in an array of `Datum`. If it is actually pointing into an array of `WDatum` then the pointer will not have been moved far enough to point to the next `WDatum` array element, but will instead be pointing to some offset inside the original `WDatum` element.

Ah! So now we know what the issue is, the fix is easy, then?

```
template <typename T>
float range( T * begin, T * end )
{
    float top = 0, bottom = 0;
    - for ( Datum *it = begin; it != end; ++it )
    + for ( T *it = begin; it != end; ++it )
    {
        float v = it->getValue();
        if ( !top && !bottom )
```

Just use the templated type `T` instead of `Datum` and the template now works for both `Datum` and `WDatum`.

All done? Not yet.

Looking at the logic, there's a test on `( !top && !bottom )`. If both `top` and `bottom` are zero then we reset both `top` and `bottom` to be the value of the current item. Why is this? Well, `top` and `bottom` are initialised to zero before the loop iterates through all the values. This initial zero shouldn't form part of the range so the first time through we need to reset both `top` and `bottom` to be the value of the current item. This isn't exactly what the code says, though, and here is a test case that proves it.

```
Datum data2[count] =
{ 0.0f, 1.2f, 1.4f, 1.7f };
Datum data3[count] =
{ 1.7f, 1.4f, 1.2f, 0.0f };

float drange2 = range( data2, data2+count );
float drange3 = range( data3, data3+count );
std::cout << "range (expect 1.7)\n"
<< "Datum 2: " << drange2 << "\n"
<< "Datum 3: " << drange3 << std::endl;
```

Here, `drange2` and `drange3` contain exactly the same values but in a different order. `drange2` starts with 0.0, but `drange3` ends with 0.0. For this pair of ranges I get:

```
range (expect 1.7)
Datum 2: 0.5
Datum 3: 1.7
```

The zero at the start of the `drange2` array of `Datum` is being ignored because although we've had a legitimate zero value, the code assumes that both `top` and `bottom` being zero means that this is the first time around the loop. While the latter implies the former, the converse does not always hold. The solution is to say what we actually mean. If we are on the first iteration then it will be the same as `begin`.

```
float range( T * begin, T * end )
    for ( T *it = begin; it != end; ++it )
    {
        float v = it->getValue();
        - if ( !top && !bottom )
```

```
+ if ( it == begin )
    top = bottom = v;
else if ( top < v )
    top = v;
```

That's better.

As another aside, `double` should normally be preferred over `float`. In most current architectures, `double` is faster than `float` and `floats` are often converted to `doubles` for calculations before being converted back for storage in a `float` result variable. Usually the only reason for choosing `float` over `double` is where there is a proven need for a more compact storage format.

Now back to the template function. Looking at the template parameter `T`, notice that it is only ever used in the function as `T *`. This is a good indication that we can make the template more general by replacing `T *` by a more general template parameter. Even the name of the iterating variable, `it`, suggests that this could be part of the original intention. I choose the template parameter name `Iter`.

```
-template <typename T>
-float range( T * begin, T * end )
+template <typename Iter>
+float range( Iter begin, Iter end )
{
    float top = 0, bottom = 0;
    - for ( T *it = begin; it != end; ++it )
    + for ( Iter it = begin; it != end; ++it )
    {
        float v = it->getValue();
        if ( it == begin )
```

Excellent, in one easy step we have now enabled our template to work not only with pointers, but with all kinds of iterators including most of those provided by the standard library's containers.

There is still one thing that sticks out as being non-generic and that is the use of the `getValue` function. It means that any type that we want to iterate through in the range template must provide a `getValue` function which returns something that can be converted to a `float`. It would be nice if the template could be used with raw `floats`, for example.

Well, it is possible, but whether it is worth the effort is open to debate. Here is one approach.

First we create a template class with a static function that can extract a `float` value from a constant reference to something. We give it an implementation that copes with anything that can be implicitly converted direct to a `float`.

```
template <typename T>
struct FloatExtractor
{
    static float Extract(const T& t)
    { return t; }
};
```

Now we add specializations for `Datum` and `WDatum` which call `getValue`.

```
template <>
struct FloatExtractor<Datum>
{
    static float Extract(const Datum& t)
    { return t.getValue(); }
};

template <>
struct FloatExtractor<WDatum>
{
    static float Extract(const WDatum& t)
    { return t.getValue(); }
};
```

We now replace the `it->getValue()` call with something that uses our potentially specialized `FloatExtractor`.

In order to get the type to use as a template parameter we use a useful template from the standard library: `std::iterator_traits`. It is specialized for pointer types to 'do the right thing', all the standard containers' iterators have appropriate specializations. If your favourite iterator doesn't have a specialization of it, you can always write one.

OK, take a deep breath, here is the new call:

```
float v =
    FloatExtractor<typename std::iterator_traits
        <Iter>::value_type>::Extract( *it );
```

OK, not very beautiful, but look, I can now use the template with a list of floats:

```
std::list<float> flist;
flist.push_back( 1.2f );
flist.push_back( 0.0f );
flist.push_back( 1.7f );
flist.push_back( 1.4f );
std::cout << "list (expect 1.7)\n"
    << "flist: " << range( flist.begin(),
        flist.end() ) << std::endl;
```

`range` is now a more powerful template, was it worth the extra work? That depends.

And now a final controversial assertion. Take this with a large pinch of salt! The original code was actually correct and the bad behaviour is actually a compiler bug. For a pointer type `p` the expression `++p` is defined as being equivalent to `p += 1` which is in turn equivalent to `p = p + 1`. The value of `p + 1` is only defined if `p` points to an element of an array object and the result is defined only in terms of the elements of that array object. For the behaviour of `p + 1` to be defined the type of `p` must be a pointer to a completely defined object type (i.e. not a pointer to `void` or a forward declared class), but there is no requirement that the completely declared object type be the object type of the array elements in question.

Interpretation of the standard is, however, notoriously difficult and this reading implies something like run time checking for all pointer arithmetic operations. This expense is almost certainly not intended, so I wouldn't expect your compiler vendor to be too sympathetic to any bug report submitted on this issue.

## Commentary

This code demonstrates a problem caused by the similarity in C and C++ between pointer arithmetic and array indexing.

If you have a single object of type `WDatum` then it 'is-a' `Datum` and a pointer to this object can be implicitly converted from a `WDatum*` to `Datum*`.

So this code is completely valid:

```
WDatum myPoint;
Datum * pPoint = &myPoint;
```

When you have an `array` of objects the same behaviour is true of each object in the array individually:

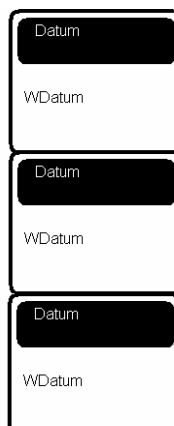
```
WDatum myPoints[2];
Datum * pPoint = &myPoint[1];
```

However, doing pointer **arithmetic** is not valid as this changes the address by the size of the actual object type. A picture may make this clearer:

Each `WDatum` object contains (by inheritance) a `Datum` object and individually can be treated as one. However, the `Datum` objects are not contiguous – they are separated from each other by the intervening extra part of each `WDatum` object.

## The Winner of CC 53

Both entries picked up the main bug – the use of `Datum` inside the template function. However Charles also noticed the bad termination condition so I've given him the prize for this critique.



Thanks to all who contribute to this column – it has been good to see a few new people contributing their critiques but there's still room for you if you've not yet entered the competition!

## Code Critique 53

(Submissions to [scc@accu.org](mailto:scc@accu.org) by 1st October

I'm trying to write a program to see if a triangle is right-angled. I've got part way there, but the program seems a bit unreliable – sometimes it crashes and sometimes it says triangles are right-angled that aren't. Can you help?

You can also get the current problem from the accu-general mail list (next entry is posted around the last issue's deadline) or from the ACCU website (<http://www.accu.org/journals/>). This particularly helps overseas members who typically get the magazine much later than members in the UK and Europe. ■



```
#include <iostream>
#include <map>
#include <string>
#include <sstream>
using std::cin;
using std::cout;
using std::endl;
typedef std::map<int,int> triangle;
// will have 3 points!
// Read integer: DD (base10) or 0xDD (base16)
int readInt( std::istream & is )
{
    std::string s;
    is>>s;
    int result(0);
    char x;
    std::istringstream iss(s);
    if ( ! iss>>result )
        cout << "Not a number!" << endl;
    else if ( result==0 && iss>>x && x=='x' )
        iss>>std::hex>>result;
    if ( ! iss.eof() )
        result = 0;
    return result;
}
triangle readTriangle( std::istream &is )
{
    triangle vals;
    cout << "Enter triangle coordinates: ";
    for ( int idx = 0; idx != 3; ++idx )
    {
        int key = readInt( cin );
        vals[key] = readInt( cin );
    }
    return vals;
}
int main()
{
    triangle vals = readTriangle( cin );
    triangle::iterator it = vals.begin();

    int ax = it++->first;
    int bx = it++->first;
    int cx = it++->first;
    double slope1 = ( vals[bx] - vals[ax] ) /
        double( bx - ax );
    double slope2 = ( vals[cx] - vals[bx] ) /
        double( cx - bx );
    if ( slope1 * slope2 == -1 )
        cout << "Right angled" << endl;
}
```

# Bookcase

The latest roundup of book reviews.



If you want to review a book, your first port of call should be the members section of the ACCU website, which contains a list of all of the books currently available. If there is something that you want to review, but can't find on there, just ask. It is possible that we can get hold of it.

After you've made your choice, email me and if the book checks out on my database, you can have it. I will instruct you from there. Remember though, if the book review is such a stinker as to be awarded the most un-glamorous "not recommended" rating, you are entitled to another book completely free. I must thank Blackwells and Computer Bookshop for their continued support in providing us with books.

## Systems Construction and Analysis: A Mathematical and Logical Framework

by Norman Fenton and Gillian Hill, published by McGraw-Hill (1993), ISBN 0077074319

Reviewed by Colin Paul Gloster

This is a very large theoretical undergraduate textbook. It is too large to read in under one month, therefore no course could be reasonably based on the book in its entirety. The coverage in Chapters 14 to 17 is too shallow as these deserve five courses (two courses for Chapter 16) and dedicated textbooks, but I do not criticize them for managing to fit so much into one book.

The book's strengths lie in formal languages and logic. However, the importance of formality is overstated in this book and inconsistencies (page 14 is contradicted by page 15, and page 180 is contradicted by page 181); mistakes (e.g. Figure 2.27 supposedly contains four different drawings of the same tree but they are not all the same tree: one has more vertices than the others); and bad names (e.g. on page 53  $\text{id}(\vee)$  was defined to be the indegree of  $\vee$  instead of the identity of  $\vee$ ) might leave students unconvinced.

It was emphasized on pages 111 and 161 that 'logical implication' (denoted by the meta-symbol  $\Rightarrow$ ) is for an assertion whereas the term 'assertion' is avoided on those pages when talking about the connective for implication within the calculus ( $\rightarrow$  which they confusingly called 'The logical connective for implication' which does not help to distinguish it from 'logical implication'). This taboo on assertion for implication within the calculus ( $\rightarrow$ ) is not maintained on pages 106 and 149.

I have found only one other undergraduate textbook which mentions fixed points (in the sense of lambda calculus), though I suppose that others have been published. It is nice to see in print their admission that functional programming is not ideal for everything.

If I ever write a book then I would be afraid that I do not know how to avoid having my next criticism rebounding on me. The index is commendably big but not good enough. For example, bijection was briefly defined in the second chapter and does not appear again until page 202, with no reminder as to what bijection

is and no entry in the index. Another example is the  $/$  character which is not listed in the index, and which is sometimes used (according to page 207) when showing equivalence classes of a set with an equivalence relation (as opposed to alternative meanings such as division). This could make the book too difficult to use for studying.

I am impressed by how many obscure languages are mentioned in the book. Object orientation was treated as a special case of procedural programming. Do you agree? One thing I can make no sense of is the claim 'specifications are loose and permissive because functions are partially defined rather than partial' on page 264. Is it a mistake or is it one of the book's subtle points which only the most intelligent readers deserve to understand?

Chapter 13 is controversial. The use of `goto` was defended if the language being used does not support a better control structure. I believe that a better approach would be to use a better language. Less than three pages were given over to Petri nets and only one sentence to Harel's statecharts. Chapter 13 is not all bad though. Its discussion of 'restructuring' shows that OO refactoring was already old when it was new.

## xUnit Test Patterns – Refactoring Test Code

by Gerard Meszaros, published by Addison-Wesley (2007), ISBN 0131495054

Reviewed by John Penney



An academic tome that needs some extensive refactoring before its handful of interesting ideas can be of use.

Unit testing is an increasingly mainstream development practice, whether it's done to support TDD or as part of a more traditional test-after methodology. Our own personal experience is that in the areas where we've practised unit testing our unit test codebase comfortably exceeds our deliverable codebase. As such, it seems that there's an urgent need for a book that focuses on the problems and patterns that are peculiar to unit test code. We were therefore keen to embrace this book but were ultimately disappointed.

What we have is a hefty book with a quality feel: for a book of this size the lack of searchable CD/DVD is disappointing. The tone is rather academic and we wondered if it was targeted at undergraduates rather than practising engineers. The book comes in three parts, each exploring unit testing from a different angle. This structure means that the reader comes across the same idea being presented several times. It makes sense in each context, but left us rather impatient. The scope of the book extends beyond refactoring too, with discussions of test tool architecture and test organisation.

The first part presents a narrative taking the reader through the goals, philosophy and principles of unit testing before presenting the different ways in which tests can be constructed and organised. It makes for a well-structured and interesting read, though the pace is a little pedestrian at times. Like the rest of the book, this section is liberally sprinkled with code samples, diagrams and tables which break up the text and are mostly informative, though we found the faddish mix of programming languages used in the examples to be rather irritating.

The subtitle of Meszaros' book is rather misleading: refactoring is really only the focus

## Bookshops

The following bookshops actively support ACCU (offering a post free service to UK members – if you ever have a problem with this, please let me know – I can only act on problems that you tell me about). We hope that you will give preference to them. If a bookshop in your area is willing to display ACCU publicity material or otherwise support ACCU, please let us know so they can be added to the list

- **Holborn Books Ltd** (020 7831 0022)  
www.holbornbooks.co.uk
- **Blackwell's Bookshop**, Oxford (01865 792792)  
blackwells.extra@blackwell.co.uk

of the second part, where the author catalogues a number of 'test smells'. These are analogous to the 'code smells' first identified by Fowler in his seminal book *Refactoring* (a 'smell' is a symptom of a problem: the analogy with a real-world smell is apt – often your subconscious knows something is wrong before you do!). Meszaros' descriptions of test smells are typical of the rest of the book, being exhaustive and well-researched. However, it's hard to get excited about some of them: the symptoms and cause of an Erratic Test are much like those of erratic production code. And an Obscure Test is as frustrating and dangerous as that obscure module of legacy code that everyone knows and dreads, and has much the same causes and cures. Still, many of the smells are interesting points of conversation and we can imagine them forming the starting point for reflection workshops where your own legacy test code is examined and its flaws exposed and discussed.

The final part of this book (excluding the rather pointless appendices) presents a catalogue of patterns. Here we felt particularly frustrated. There are lots of interesting ideas here: we have a bestiary of 'test doubles' for example where different kinds of mock/stub/test object are explored, and an odd but strangely satisfying little collection of 'value patterns' in which Meszaros presents three different patterns answering the problem 'How do we specify the values to be used in unit tests?'. But the sheer number of patterns and detail afforded to each means that it's impossible to make use of them.

A lot of the OO principles and patterns that apply to production code should apply to unit test code, but this book demonstrates that there are some interesting smells and patterns that are more prevalent in the world of unit testing. But this interesting material is swamped in detail and duplication. There are a number of (decidedly smaller) books on unit testing and related topics that can and do provide more benefit. Prefer a different unit testing book such as *JUnit Recipes* (Rainsberger) and a copy of *Working Effectively with Legacy Code* (Feathers).

## Getting Noticed on Google

by Ben Norman, published by Computer Step, ISBN 1-84078-332-X

Reviewed by Giuseppe Vacanti

On the High Street the location of your business is very important when it comes to attracting passing shoppers. On the electronic high street, this translates to being listed within the first few links returned by a web search engine.

OK, I wrote 'a search engine', but most likely you read Google. In December 2007 Google topped the US search engine rankings, with almost 60% of the searches. Yahoo came in a distant second, with 22%, followed by Microsoft with 10%. We commonly take these numbers to apply to the rest of the world, and so

we come to the conclusion that topping the Google ranking is the most effective way to be found by the Internet shopper.

*Getting Noticed On Google* is a short book containing a number of procedures and bits of advice that web site managers should follow in order to raise their sites' Google ranking. Most of the advice contained in the book is eminently sound, although, given that the Google algorithms are not known, it is difficult to judge how effective any of it can be. On the other hand, the book's advice follows the gist of what can be found on Google's own web master pages, so by that measure it cannot be too much off.

If you are ready to read all of the material that Google itself makes available to web masters, you probably do not need this book. The book is however quite accessible, mostly avoiding technical jargon and marketing mumbo jumbo (with the exception of a few sentences here and there, like on page 8 where we encounter 'Google is an advertising vertical in its own right that, if used properly, can provide an abundance of highly converting traffic.' Uh?).

As I said, most of the advice in the book makes sense, and it would apply to a web site even if it were not trying to enter the Google top ten list: choose your keywords well, have meaningful text for your links, stay on message, and in general – although this is finally mentioned only one third down the book – write good content for your site.

The book covers a number of free tools available in order to study how Google has ranked your site: unsurprisingly these are all provided by Google, who has all the knowledge on this matter. Another tool mentioned is Web CEO, that comes in a free and a business version. Web CEO makes an analysis of your web site, and suggests way to improve its web search ranking. Finally, the book covers also Google Analytics and Google AdWords: the former as another analysis tool, the latter as a way to bring traffic to your site.

All in all, the book is a worthwhile read. Although much of the same information could be gathered by reading a few of web sites, including Google's own pages for web masters, the book brings it together in a concise manner, and it offers many ideas for further investigation.

## Genesis Machines – The new science of biocomputing

by Martyn Amos, published by Atlantic Books, ISBN-13: 978-1843542254

Reviewed by Ian Bruntlett

My apologies for this review – I'm only competent to review the conventional aspects of this book and am not qualified to review the biological aspects of this book.

This book looks decades into the future, considering the fundamentals of computation

(Von Neumann Machines, Turing Machines) giving a birds-eye view with copious references to other sources and books.

Chapter 2, 'Birth of the machines', covers things like the tally stick, a computing device dating back to 20,000 BC, and the rise of mathematics (in the 5th century BC Indian mathematicians discovered the number zero and the idea of a positional number system). It (incorrectly) states that you can count up to 1024 on 'your fingers using binary': this is incorrect, you can only count up to 1023! However it does do a good job of introducing binary and boolean algebra (AND, OR, NOT but not XOR).

It describes the steps taken to produce the first 'Feynmann machine': the TT-100, a computing device that used the manipulation of molecules at the heart of its operation.

The rest of the book covers the development of biological / DNA computers, something that is beyond my realm of experience.

VERDICT: An interesting read, if a little technical.

## Implementation Patterns

by Kent Beck, published by Addison-Wesley, ISBN 978-0-321-41309-3

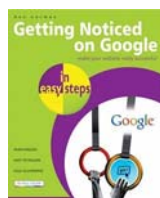
Reviewed by Seb Rose

It is difficult to decide who this book is aimed at. Given the title of the book (and the subtitle on the back cover that reads '... a Catalog of Patterns Infinitely Useful for Everyday Programming'), you might expect a reference tome that sits beside the ubiquitous GoF volume. I doubt that that is where it will end up.

For starters the book is unashamedly slanted towards Java, with references to Java visibility, Java collection classes and (in the context of framework design) the Java packaging scheme. This is not necessarily a weakness in itself – the GoF book presented the material in C++ and Smalltalk – but the patterns in this book are much lower level, and in some cases, much more specific to the implementation language. I would have expected some indication of this bias in the title or on the cover somewhere.

I would also question whether the book describes patterns at all. Certainly, they are not presented formally (with context, problem, forces, solution and consequences), but instead are small sections of prose grouped into chapters with titles like Class, State, Behavior, Methods.

Some of these sections describe the Java language itself (e.g. 'Behavior: Control Flow') while others are stylistic in content (e.g. 'State: Role-Suggesting Name'). I found myself agreeing with almost everything that was written, but very little reached levels of concreteness that I actually thought might be useful. And when I read that 'the whole question of equality seemed more important 20 years ago' ('Methods: Equality') I found myself





### View From The Chair

**Jez Higgins**  
**chair@accu.org**



In my last 'View From The Chair', I sang the praises of several retiring office holders, thanking them for their various contributions to ACCU over the past several years. As I've said before, we don't always express our appreciation as often as perhaps we should. While it is important to acknowledge the work people do, I don't want to appear as if I'm begging for pats on the back for the committee or for myself. Nor do I want to suggest the benefits only flow from the individual to the membership. The reverse is also true.

ACCU is a participatory organisation. All of us participate in some way, even if we don't realise it. That little spark of recognition, excitement even, when you see a new CVu sitting on your doormat? That's it, that's the start. If your software practice has been changed even slightly by your membership of ACCU, then consider yourself engaged. You have, as they say, got the point. I believe, though, that the more we, as individuals, take part in ACCU activities then the more we, both as individuals and collectively as an organisation, benefit. These benefits are not always direct, measurable, or obvious. They might be unexpected or surprising. They might take years to blossom, but they are there and they are real.

I don't want to turn this View into a plea for more articles (although I believe every member does have something interesting to write about) or to beg you to join the committee (but I'd be happy to see you). I would simply ask you to consider doing something. It need only be small – a book review, an email to accu-general, a discussion with a colleague. It could be large – a series of articles, a conference presentation, becoming an officer. You might find it very easy, or it may present a challenge to overcome. Whatever it is, you'll have fun and ACCU will appreciate it.

### Membership Report

**Mick Brooks**  
**accumembership@accu.org**



This month marks the association's traditional membership renewal period. For new members who don't know what I'm talking about, we used to have a fixed membership year, running from August to August. Although membership is now on a more flexible rolling scheme, the vast majority of subscriptions still become due at the end of August.

The preferred way to renew is by logging in to the website, then following the links named 'Account', then 'ACCU Subscriptions', and then 'Renew', where you can pay by credit or debit card. I'll happily accept a cheque if you'd

prefer (email me for details). Some of you will have arranged to pay by standing order, and won't have to do anything to ensure your membership continues. If you're not currently paying by standing order, then you're too late to arrange it for this year, but drop me a line if you want to set it up for 2009. Less hassle for you, and we even give you a discount.

However you choose to pay, now is a good time to log in to the website and review your mailing address details and contact preferences. If you have problems or questions about renewals, or anything else, then email me at [accumembership@accu.org](mailto:accumembership@accu.org).

### Advertising Officer Report

**Seb Rose**  
**ads@accu.org**

It has been an eventful few months in advertising. We reworked our advertising packages (to make them really good value) and have now sold all of our cover spaces until well into next year. There were a few teething problems with our use of OpenX to deliver ads on the web site, but we got over them and we now have only one slot left to fill.

So if you think that your organisation could benefit from exposure to the thousands of visitors to the ACCU web site, you'd AUG 2008d best get in touch as soon as possible!

## Book reviews (continued)

wondering if I'd been transported to a twilight zone somewhere between software engineering and gender politics.

The prose style is clear and easy to read, which lends credibility to the main thrust of the book (outlined in a 2-page chapter 'Motivation'), which is that coding style is about communicating intent to people. However, I think that less experienced programmers will find it all a bit woolly, while those with more experience will wonder if there's anything good on telly.

### Beyond the C++ Standard Library – An Introduction to Boost

**by Bjorn Karlsson,**  
**published by Addison**  
**Wesley,**  
**ISBN 0-321-13354-4**

**Reviewed by Rob Jones**

Highly Recommended



My technical books broadly fall into two categories. On the one hand there are references, like Stroustrup's C++ Programming Language or Josuttis' works. On the other hand there are books with more of a narrative flow like Meyers' or Sutter's books.

With *Beyond the C++ Standard Library* Karlsson manages to fall into both categories.

After a brief introductory chapter the remainder of the book is simply a catalogue of popular picks from the Boost libraries, one chapter per library. In some ways this is one of the book's few downfalls, in that the coverage is necessarily patchy. The book in no way attempts to offer full coverage of the Boost libraries, and would be a pretty hefty tome if it did, but having had Karlsson show you around some libraries he is missed for others.

In keeping with the Boost libraries themselves, this is not a guide for the C++ novice. The base level of knowledge presumed by the author is significant, and you need to be familiar and comfortable with all the advanced features of C++, like templates and functors, and similarly with the STL (containers, algorithms, binders etc). In a book titled, *Beyond the C++ Standard Library* this does perhaps go without saying.

The text is graced with many excellent examples, which conspire to cover a broad range of different use cases, and together form a useful body of knowledge of the details of the syntax. In same way that K&R shows the idioms of C, and Meyers and Sutter many of the idioms of C++, this book shows us how to get things done

with the Boost Libraries. It has quickly become the volume I reach for when I need to remind myself what a certain construct looks like.

The text accompanying the examples is comfortably fast paced, and with a pleasant narrative style. Karlsson even manages to inject a little humour from time to time, which is welcome in book dealing with such a intricate topic. As well as being a comprehensive account of how it works, this book also manages to convey and share the author's obvious pleasure at the beauty and elegance of the Boost libraries, and gives good insight into the architectural forms that these facilities encourage. In short, with this book under your belt, the solution space for everyday programming requirements is so very much richer.

Your mileage may vary depending on your environment, but for me the highlights were the chapters on Boost.Bind, Boost.Lambda & Boost.Function. The ability to define anonymous functions at the call site scratched an itch for me in the same way that the introduction of classes into C did – an elegant solution to an obvious problem.

At 380 pages, this is already a substantial book, but I can only hope Karlsson will write further volumes!