The magazine of the ACCU

www.accu.org

# {cvu}

Volume 19 Issue 4 August 2007 £3

## Features



## Regulars







{cvu}

Volume 19 Issue 4
August 2007
ISSN 1354-3164
www.accu.org



# Why write for ACCU?

As most of you are aware, the contents of both C Vu and Overload are written by ACCU members and contributed for free, for the benefit of the ACCU organisation as a whole.

Why write an article? There are several responses to this. A great one is for the author to learn. You do learn a huge amount while writing an article. Ideas that were only half formed get clarified and a deeper understanding is gained through the need to explain the idea to others. The author of the article also gains experience in writing. Having the ability to clearly communicate ideas in writing is a great skill for a software developer.

A number of years ago I bought a copy of the book *The Cathedral & The Bazaar* by Eric Raymond. The book is a collection of essays on Linux and Open Source. I found the book extremely enjoyable, and it is one of the reasons that I ended up learning Python, but that is for another time. The essay that I want to draw a parallel to is called 'Homesteading the Noosphere'. In the essay Raymond talks about gift cultures and how they relate to open source software developers. I think that writing for ACCU fits into this idea of a gift culture, the authors of the articles that you read in C Vu and Overload are gifting their experience, knowledge and time.

Becoming an author within ACCU, a highly knowledgeable organisation with many members who are top of their fields, will contribute to your own professional development, and help you forge a reputation of your own. This reputation is not necessarily direct or immediate, but grows over time. There is always space at the top for those willing to work the hard yards to get there. Now I'm not saying that writing for ACCU will make everyone think that the sun shines out of your…, however if you are aiming for the top, becoming an author is a step in the right direction.

TIM PENHEY,
**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.

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

# DIALOGUE



# FEATURES



# REGULARS



# COPY DATES

**C Vu 19.5:** 1st September 2007
**C Vu 19.6:** 1st November 2007

# IN OVERLOAD

Roger Orr debugs release builds, Thomas Guest uses shell scripts, Richard Harris tries to reduce unnecessary copies and Kevlin Henney revisits Parameterize from Above.

# 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.

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

# COPYRIGHTS AND TRADE MARKS

# Please release me

## Pete Goodliffe helps us to "let go".

A few months ago I was brought in to help a team make a software release. This wasn't the first release that the group had made, but it was a pretty important one, and they wanted me to make sure that it went smoothly. In this kind of situation you need someone with nerves of steel, an unerring eye for detail, and an encyclopaedic knowledge of the intricacy of every build system and release procedure. Or failing that, you ask me to help.

Now, I'm no "release engineer" (yes, there are such curious beasts out there). But I've been around long enough, and made enough software releases to know what passes muster as Good Practice and what's Not Good Practice.

What was going on in this shop was Not Good Practice. Their release 'procedure' almost caused me to involuntary vomit.

They didn't really know what was going on, and they certainly weren't in control. The release seemed to have come as a surprise to everyone – if it was on a plan anywhere then no one had looked at that plan until the last minute. There was no documentation about how to make a release; it was knowledge tucked away in a single person's head. And sadly it hadn't been remembered very well, so we had to work most of it out from first principles again. Even the software's build process wasn't understood properly. And to add insult to injury, the release could only be made on this one guy's development workstation because:

- A lot of the code was not under source control; it lived locally on the hard disk in his machine (which – as I'm sure you can guess – was not being backed up). The code was based upon an original software release by another organisation. We had no record of where this original code came from, which version of the code it was, or how it had been subsequently modified.
- The exact build environment (including the required version of the compiler, a set of libraries, and other environmental setup – environment variables and config files – had been set up magically on this computer, and had not documented or put in a central configuration file anywhere.
- He couldn't remember exactly how it had been set up like this in the first place, and couldn't produce another identical build machine. If that machine ever died we'd be in a world of painful pain.

**Different types of software are constructed and deployed in different ways, and consequently have different release procedures**

That was not good. It was not justifiable. It was just plain wrong. And I made sure everyone knew about it.

Sometimes I really do moan.

But sometimes it's justified.

Software releases are the lifeblood of the software development industry. If all of your programs were mere toys for personal use then you wouldn't need to worry about having a good software release 'procedure'. But instead, you'd have to worry about how to make enough money to pay for all the electricity you were wasting. In the Real World, most every piece of software must be 'released' in one form or other: to test, to manufacture, to the customer, to the wild world beyond the programmer's workstation.

There is a real art to doing this well. It's not too hard, but it requires great care, a methodical approach, and attention to detail. Contrary to some opinions, getting it theologically right really does matter. Bad software releases can bite you on the backside weeks, months, or even years later.

So in this article I'm starting a mini series on the joys of releasing software. We'll see what software releases are, what 'good' software releases are, how to make them, and then how to make them over and over again.

Once I have finished this series I will stack all the articles in a pile, roll them up tightly, and start beating people round the head with them!

### The types of software release

This would be an easy article to write if there was only one way to release software, a single universal mechanism. But there isn't. Different types of software are constructed and deployed in different ways, and consequently have different release procedures. Consider, for example:

- Shrink-wrap software (the kind of PC applications that are commonly sold to users in shrink-wrapped boxes, but these days are equally likely to be downloaded over the internet) must be built, the distribution files must be incorporated into an installer that the end-user can run, and then either wrapped into an ISO image to be put on a CD or DVD, or perhaps zipped up into a archive that can be downloaded over the internet.
- Web-based software and web services are not distributed to end users in the same way. Instead this kind of software release is deployed on a web server. If the release is an update to an existing web application, the users will immediately (and hopefully seamlessly) start using the new version without any knowledge of an upgrade taking place.
- Embedded devices commonly have their software (otherwise known in this domain as firmware) compiled and burnt into memory chips on the device. Depending on the richness of the embedded device, the software may be upgradable in place, or the install process may be a once-only manufacturing operation. A single embedded device may consist of several chips (a general purpose CPU, PICs, micro controllers, etc) which each need their own software installed and loaded in various different ways.
- Mobile application software is a specific, but increasingly common, form of embedded software release. Most mobile devices theses days are very rich-featured embedded devices, and software is often installed to them live, over a network connection.
- Software libraries and software frameworks may be developed as shrink-wrap components for other developers to use. They might be distributed as pre-built object files or shipped in source form. C-

**PETE GOODLIFFE**

Pete Goodliffe 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

style languages also require a set of header files to be shipped with the software.

So clearly there are many different forms of software and, consequently, of software release procedure. However, they have many common considerations, and I'll do my best to cover them here.

## What is a software release?

OK, let's get the tedious "obvious" stuff out of the way first. What actually is a software release? My personal definition of a software release is:

> The process of obtaining, building, packaging, and deploying/distributing a versioned software product, for an agreed purpose.

Let's break this down one step at a time and see what it means. We'll look at it all in more detail later on in the series.

- **Obtaining the code**

  The first step in a release procedure is identifying and obtaining the code that constitutes the release. We must start from a well-known code set, extracted from a well-known place. Usually that means it is held under source control and you know what version/branch the code is to be fetched from.

  We actually care about more than just source code at this stage; we must collect all of the code files and all of the required supporting material that will contribute to creating the release (including graphics, data files, documentation, etc).

- **Building the code**

  In many programming languages, the source code files must be compiled into object libraries or executable programs, or byte-compiled into files capable of being interpreted by a language executor. The outcome of this step will be one or more program files, along with any runtime support required to make the program run (e.g. data files, control scripts).

- **Packaging the code**

  Once all of the executable files have been prepared, the results must be packaged. They may need to be collated into a central location from their positions around the source tree. A program installer might be generated, or the files simply archived together. As the files are packaged a README file may be incorporated as a convenience for the end user.

- **Deploying/distributing the packaged software**

  Often this step is forgotten in the release process, especially in shrink-wrap land. A DVD or CD image of the release set must be created if the software is shipping on a physical medium. If the software is a network service, then it must be copied over to and installed on the central servers. This is an important integral part of a release process. Releasing software doesn't stop when the code has been built. (There is potential to make an error right up to the point the user is given the software!)

- **Versioned software**

  There are several versions that must be tracked by a good software release. The first 'obtain the code' step must fetch a clearly defined revision of a code set. However, the constructed software release will have a version number (that may or may not be derived from the source code version).

  The format of the release version number isn't important, but end users must be able to differentiate different software releases of the same program. Clear version numbers help you tell "Version 1.02" of the software apart from "Version 1.08".

  Ideally, the software itself has some concept of its release version number (perhaps it displays it in an "About this program" screen). This adds an extra burden on the release mechanism: it must ensure that the software's internal idea of the version number matches that of the release process. It is very confusing to have a version "2.14" software installer that installs software claiming to be version "2.10a"

- **An agreed purpose**

  If you are making a release then there must be a need for it. Perhaps that goes without saying, but plenty of programmers do silly things for no good reason!

  You may be creating a brand new release of a new piece of software heading out on its maiden voyage. It might be an incremental upgrade release – either a 'patch' that installs over the top of a previous software install, or a new self-contained complete install. It might be a major backwards-incompatible version of the software that must first ensure any old code has been deleted as part of the upgrade process. Each of these motivations alters the requirements of the final software installer.

As a product nears its ultimate "release" to the big wide world, we will usually perform a series of mini-software releases:

- Often an *alpha* release of the software is made. This tends to appear quite early in the project, and is released either externally, or to a set of in-house testers. There will be no promises about the stability or quality of this alpha release – it just gives a good idea of what the software will be capable of.
- Later in the process we make *beta* releases. These are considered fairly stable and form the basis for serious bug testing.
- As we draw nearer the finish line a series of *release candidate* releases are made.
- When one of these is deemed acceptable, it becomes the "official" release and is deemed the *gold master* [1]. This is the ultimate release that is put onto CDs and sent to customers, deployed on the live web server, or that gets sent to far eastern factories to be put into the manufacture process.

there should be very little variation in the actual release procedure, just in the final destination of the software release

Each of these different releases should actually be made in a very similar way – there should be very little variation in the actual release procedure, just in the final destination of the software release. This is crucial – as we make the early testing releases we are therefore testing the release process as much as we are testing the software itself. We use the early releases to iron out bugs in the release procedure so that when we make the final release everything is guaranteed to work well.

## The prerequisites for a good release procedure

In the remainder of this first article we will consider the prerequisites for a good software release.

It's important to get our software release processes right. The most perfect software can easily be spoilt by a half-baked release procedure. What use is all your hard work if the software release incorporates an old, buggy set of source files instead of your latest-and-greatest, or if it fails to create a robust software installer image?

- Before you start to make a software release, you must plan it carefully. Do not rush out a release when it seems appropriate, think about it beforehand. You must define: its purpose (is it going to be a minor bug-fix/maintenance release of an existing software version, is it a new feature release, etc), when you intend to ship it (ah! we all love deadlines, don't we), whether there is any play in those deadlines, and when the alpha/beta/release candidate releases will be made.
- Software releases should be a high-level planning issue, tracked on a project plan and clearly visible to project managers.

# The Bazaar Thing

## Steve Love works the version control job.

I am a Subversion [1] user, and have been for quite some time. Before that, I'd used CVS [2] for quite a while, and been very happy with it until Subversion came along with its atomic commits and versioned file system. Some things just break your fidelity down too much, but Pete Goodliffe [3] has some very sound advice on this, which, paraphrased, is "pick a version control system you like and require a compelling reason to move away from it". Subversion had reasons good enough for me to switch from CVS. I wasn't expecting good enough reasons to move away from Subversion any time soon until our illustrious editor, Tim Penhey gave a 'Birds of a Feather' session at the ACCU 2007 conference on the highlights of Bazaar [4].

Now I have fairly straightforward requirements from my VCS; the only thing that really matters is that it works and I have enough trust in the software to be able to make any kind of mistake in my working copy safe in the knowledge I can roll it back to a previous version. If it's easy to use, cheap to install, doesn't have huge disk-space requirements, well all those things are nice, but they're just the dressing really. It's trust that matters. As developers we (should!) see our VCS as the Keeper Of The Keys, One Who Protects Our Valuables. It's an important job, and we want to make sure we hire the right software for it. So to speak.

### The cons of SVN

So back to Subversion. It's been a handy tool, but lacked just one thing. I work on a laptop quite a bit, and so want to be able to check in and out when I'm disconnected from real life and have no network connection. Subversion assumes a single repository, possibly on a network, perhaps even the Internet. Checking in a file sends changes to the one true repository. If my repository is on my home PC and my laptop is disconnected from my LAN, a check-in from the laptop must fail because svn can't connect to the repository.

One answer might be that my "main" repository is on the laptop, and I update from there to my home PC when it's connected; this obviously requires my laptop to be a server, which seems back-to-front. Another answer might be to have the main repository on a thumb-drive, and use

**STEVE LOVE**

Steve Love is an independent developer constantly searching for new ways to be more productive without endangering his inherent laziness. He can be contacted at steve@arventech.com

## Professionalism in Programming (continued)

- It must be clear who is going to make the release. In large teams there is often a designated buildmaster who maintains the build system and takes responsibility for managing all software releases [2]. In smaller teams the build/release role may move around.
- The release procedure must be clear and unambiguous. Even if you have a single buildmaster who takes overall responsibility, the whole process should be well defined and documented so that anyone can perform a release in the future.
- The release processes must be really very simple; this dramatically reduces the likelihood of error. We'll see the best way to document the process and make it simple later.
- It must be clear who makes the final decision about whether the software is acceptable and a release can be made. Naturally, this means that you should have a thorough QA process and good bug-tracking system.
- The code must be managed under source control. We need to be able to track in the future exactly what source set when into release 7 of our software. We must also have a mechanism to record which set of files formed the basis of which software release. (Source control system tags/labels and branches are useful for this.)
- Your software must have a robust and reliable build procedure. Building the same set of source files twice must always produce an equivalent set of output files. You can't have a reliable release procedure without this.
- A good release always starts from scratch. You should not build an important release over the remains of a previous release, no matter how tempting it is. Sure, you don't think that it matters, that any of those old files have changed, or that anything can go wrong. But you still shouldn't do it. Wipe the old build tree, and start again.
- You must leave sufficient time to make the release correctly.
- When crunch time comes it becomes far too appealing to make short-cuts (like building on top of a previous release to save having to compile all the source files). Ensure that releases are started early enough in the day to prevent a last minute rush and panic.
- There must be sufficient time to sanity check a release before it has to be handed on.
- The releases procedure should be audited and managed adequately.

Admittedly, that was rather hand-wavy list of ideals. But it's all important stuff and key to a reliable, robust software release procedure.

### Until next time...

In the next article we'll start to look at the mechanics of the software release process. This was the relaxing part – next time we'll pick up our shovels and dig a big hole for ourselves! ■

### Endnotes

[1] Terms here vary, but the intent is usually the same. Some teams call the gold master the production release, for example. The term "gold master" hails back to the production of vinyl music records, where the original moulds that records were pressed from were plated with gold to preserve them.

[2] I always worry that the kind of engineer who likes to be called a buildmaster is likely to be the kind of person who likes dressing up in a cloak and a pointy hat. But maybe that's just my weird preconceived notions.



that for everything. This works too, but thumb-drives are, if nothing else, prone to getting lost.

The alternative is to have a local repository on the laptop, which gets synchronised with my server repository when I have a network connection. For simple things, exporting from the laptop repository, and then checking that whole export into the main repository works just fine, but it loses all the change history associated with the laptop repository. Retaining that info can be done, but it's quite frankly more trouble than it's worth.

Newer versions of Subversion (1.4 and above) have svnsync, which allows mirroring of a repository to a slave, but the slave is read-only. This is fine for backups and mirrors, but not for remote check-ins.

### "I want" volume 1

So what I'm really saying here is that I want a VCS significantly like Subversion, but which allows me to merge changes from one repository to another. Specifically, which allows me to check-in locally on my laptop, and then merge those changes, with the version history, to another repository when I've got a network connection.

This functionality is actually planned (in some form) for future Subversion releases, but that particular future is a long way off. Also planned for Subversion in the medium term are improved rename support and smart merge tracking [5], two features described for Bazaar in more detail below. As with decentralised working, these are features that Bazaar gives me today.

**what I want from my VCS and what the next person wants may not be the same things**

## Back on topic

So I hear about the Bazaar VCS. Where Subversion is a centralised system, Bazaar is a distributed one. When people start talking about something being "distributed", I tend to start thinking about "massive complexity" and get just a bit cautious. To be honest, I was initially interested in hearing more because I know precious little of distributed version control; I didn't really register the fact it could help me with my disconnected laptop problem until after I'd heard more about it. There's a lesson in there, I think...

Bazaar is really more accurately described as de-centralised. The distributed nature of Bazaar repositories is a side effect of this. Instead of a single repository to which commits take place, a project versioned under Bazaar may have many repositories, each with many branches. It was this point – which I'll cover in a bit more detail – which first struck me about how Bazaar might help me overcome my particular problem.

The other thing that struck me about this "distributed" Version Control System was its simplicity. In terms of usage it is very Subversion-like, a deliberate design goal of the Bazaar development effort, and requires little in the way of set-up. Not things I expected from a "distributed" system!

## The inevitable comparison

So here we get right down to it: why do I think Bazaar is a suitable replacement for my beloved Subversion? The answer, of course, must be, because it does some things I want that Subversion doesn't do, and it does all the things I like about Subversion as well as Subversion does, or better.

Naturally, what I want from my VCS and what the next person wants may not be the same things, and therefore Bazaar (or Subversion, or any system for anything, for that matter) is not ideal for everyone. Mark Shuttleworth, the founder of Ubuntu and outspoken advocate of Bazaar, has interesting views on this, see his blog [6].

For me, the following aspects persuaded me that Bazaar was the way ahead.

### De-centralised repository handling

As I've already mentioned, this was the "one thing" I wanted that Subversion doesn't (yet) support – the ability to check in to a local repository and be able to merge those changes to another repository when it's convenient to me. Bazaar's de-centralised way of working supports this need directly, without resorting to scripts or hackery or third party tools. It is, in fact, the preferred approach to version control in Bazaar.

A branch can be made from any published repository. "Published" really means you can see it; it may be made available as a shared folder on a network, via an http server (no special tricks – it's just a file system), an ftp server or via Bazaar's built-in BZR protocol [7].

For my purposes, I use a shared drive between Windows-based computers. My home PC is named 'Churchill' on my home LAN, and it has the main (HEAD) repository. It shares the repository folder, `c:\dev`, on the network as `dev`. My laptop, called 'Attlee', connects to my LAN when I'm at home and also shares `c:\dev` as `dev`.

The basic work flow is that I would branch from the main repository to my laptop. On my laptop at a command prompt:

```
bzr branch \\churchill\dev c:\dev\root
```

The arguments are the **`[from]`** repository, then the **`[to]`** directory. I can now disconnect the laptop and take it on the train, checking in my changes as I go.

Note there is no need to check-out the newly created branch. In Subversion (for example), a branch is just a copy of some part of the development tree inside the repository. To work on the branch, you check out the branch, and check-in against it. In Bazaar a branch is the local repository, although it also supports what are called "bound branches", which behave more like a centralised system like Subversion. In the common case, the repository is a magic folder in your working copy area, called `.bzr`. Bazaar supports a separate location for the repository, too.

When I return home and reconnect my laptop to the network, in a command prompt on my home PC I can enter:

```
bzr merge \\attlee\dev\root -d c:\dev
```

The **`-d`** argument specifies the location to merge into. The default (no **`-d`** argument) is the current directory. This command brings all my changes, with history, back into the trunk. If I know (and I usually do, because I'm the only programmer in my house :-) there are no changes to the trunk on Churchill when I want to merge, I can instead perform the update to trunk from the laptop:

```
bzr push \\churchill\dev -d c:\dev
```

The **`-d`** argument in this case is the location to push from. As with merge, it defaults to the current directory. The push command will fail if the trunk has diverged – i.e. it contains changes which have not been merged to my laptop branch. Push differs from merge in that it makes the target directory a mirror of the source.

As with all good VCS systems, Bazaar has **`status`**, **`diff`** and **`log`** commands which do the expected things, including seeing the differences between two branches, and showing the log for a specific file on a branch. Of course, in the case of Bazaar, they are different repositories, but it helps to just think of a branch in a Bazaar-managed project as just a branch in say CVS or Subversion. The only difference is that accessing a particular branch might require network access.

In any case, for my common uses, that's it! How easy is that?

### Smart merging

I can already hear at least the Subversion users reading this asking "how do I perform subsequent merges when I make changes in the branch?" The answer is similarly easy: exactly the same thing as before:

```
bzr merge \\attlee\dev\root
```

Note the lack of revision numbers here. The default behaviour of Bazaar is to merge in all changes since the most recent merge – Bazaar remembers when that merge took place, so there is no need to specify the base revision. You can of course explicitly specify the range of a three-way merge:

```
bzr merge -r 10..51 \\attlee\dev\root
```

Conflicts occurring as a result of a merge are handled in the same way as with Subversion; conflicted files must be explicitly resolved, before they can be committed.

### First-class renames

A primary reason for my moving to Subversion from CVS was its support for moving directories and files around the repository. Even a mild refactoring of a development tree could cause copy and delete headaches for CVS, and Subversion provides that facility natively, so I can just rename stuff as I go.

But it's far from perfect in Subversion, either. The main problem is that, under the hood, Subversion just does a copy and delete. OK, so whereas in CVS such a (manual) operation would lose all the revision history, Subversion keeps that information, so I can look at a log of a file and see that it was moved from `x` to `d` at revision n. The main problem comes when merging between branches having different renames, and Subversion tries its best, but gets unstuck in some circumstances. Try this in some of your favourite VCSs:

- Create a folder `test`
- Create files `file1` and `file2` in `test`
- Check in to trunk
- Create a branch from trunk
- On the trunk, change `file1` then check in
- On the branch, rename `test` and change `file2`, then check in
- Merge back from the branch to trunk

With Subversion, my modifications on trunk to `file1` were lost, overwritten during the merge. Using Bazaar this worked just fine, merging changes into the renamed directory.

Bazaar makes item renaming a first class operation, and tracks those changes transparently so that merging between branches with different renames "just works". It may result in conflicts, but that's an inevitable result of rename support under these circumstances. I would much rather have to resolve conflicts than silently lose information!

### All the other stuff I like about Subversion

Bazaar has only a few commands, with simple to understand arguments, a policy it inherits (deliberately) from Subversion. There are commands like **`rename`**, **`copy`**, **`move`**, **`commit`**, **`update`** and **`blame`**, which do just what you'd expect.

## Can it be that perfect?

Of course not. There are trade-offs you need to decide upon when choosing any important tool, and certainly Bazaar has its faults. It is not always as quick as some other VCS tools, but only you can decide whether that matters. For my own small projects the difference isn't even noticeable.

The documentation is adequate but not fantastic; one of the greatest strengths of both Subversion and CVS is the quality of their documentation. I'm sure the Bazaar development team will get there in time. It's still young enough to be a moving target anyhow.

Related to its youth and Linux based heritage is the lack of UI tools for a Windows platform, such as Tortoise for CVS and Subversion. A TortoiseBZR project is underway, which I think will aid the take up of this fantastic tool no end.

## The prime directive of Bazaar

Low barriers to participation. That's it – in other words, keep it as simple as possible. Bazaar is pretty much just one command line client. Simplicity oozes from Bazaar, and drives everything it does. Rename support "just works" because that makes life easy for the developer. It's related to the Smart Merging provided by Bazaar, which in turn is related to the distributed nature of Bazaar. These things didn't get implemented by accident, they are all part of the philosophy of Bazaar which is about developer collaboration, and making that easy. Bazaar lives up to this philosophy admirably. Bazaar is free software, with a lively and vibrant developer- and user-community, and takes its name from the Bazaar Development Model, characterised in Eric S. Raymond's essay *The Cathedral and the Bazaar* [8].

For my needs as a sole-developer it's very nearly perfect. Bazaar would also be suitable for my work in a team of developers, because it supports several ways of working, and doesn't impose a single use-case on you. For example, it is possible to use both a central repository for Bazaar and lock-step-commit as if it were CVS or Subversion. In team where some or all the developers work at home or off-site, Bazaar wins again, for the same reason as it does for me and my laptop, especially if remote access inward through a corporate firewall is technically or politically difficult to implement.

Bazaar is so easy to get, so easy to install and so easy to use for simple things, I think I'd even recommend it for newcomers to revision control, because it has no need for a server or separate repository, and you can just work with it in-place with your development. All in all, this is my "developer's choice" tool for 2007 :-) ■

## References


1 Subversion: http://subversion.tigris.org
2 CVS: http://www.nongnu.org/cvs/
3 Pete Goodliffe, 'Effective Version Control' (Professionalism in Programming #40) *CVu* vol. 18.5, October 2006.
4 Bazaar: http://bazaar-vcs.org/ At the time of writing, version 0.17 has just been released.
5 http://subversion.tigris.org/roadmap.html
6 Mark Shuttleworth at http://www.markshuttleworth.com/
7 Bazaar doesn't really have the concept of user authentication: it assumes that can be done on the filesystem itself. Therefore, if you have permission to write to a directory, you are a committer.
8 http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

# A Brief Introduction to Cygwin

Silas Brown finds alternative tools.

GNU/Linux is a good operating system, especially for developers – it has so many useful tools immediately available, and it's relatively easy to hack and tweak how things work, besides other advantages. But there can be problems.

Linux is excellent at supporting old hardware; in fact it can extend the life of old hardware considerably, which is good for the environment and for your time – a good stable distribution will run for many years with minimal maintenance once it has been set up. But for new hardware things can be different, especially if it's non-standard and the specifications haven't been released to the public; in this case there can be a time lag before the Linux developers can catch up. For example when a company loaned me one of their brand-new laptops for a temporary work contract, and I tried putting a Knoppix bootable Linux CD into it, try as I might I could not get Linux to use its Ethernet card properly, and neither could anybody else who had that particular model of laptop according to my web searches.

The included copy of Windows XP was already working OK with the hardware, but I was very limited in the applications I could use with my partial sight. Thankfully with a modern TFT display I don't have to worry quite so much about always choosing colours that won't hurt, but if you need large fonts then Windows is not very good. The only way to reliably enlarge *all* the fonts is to change the display's resolution and/or DPI setting, but then a lot of dialogues won't fit on the screen and won't scroll. This means you can't get at the buttons, so you have to keep temporarily changing the resolution back and squinting, that is unless you have a spare thousand pounds to invest in a top-notch screen magnifier program but that will only last until the next version of Windows. If I were spending that amount, I'd rather just buy a Mac or a Linux box which can do it anyway. I managed to use the applications that the company wanted me to use to get onto their intranet, but beyond that I just couldn't get on very well without some decent software from the Unix world to help me out.

Enter Cygwin. Cygwin has advanced a lot since I last looked at it (which wasn't all that recently) and it now comes with an installer and lots of packages that are reminiscent of a Linux distribution. This makes for a very easy way of putting a Unix world onto Windows and running it safely and at full speed. Besides all the compilers, interpreted languages like Python, and command-line tools, there are things like TeX, X11 and associated programs, LyX, netpbm, XEmacs (although the Cygwin version doesn't include as many features as if you download it separately), and a host of other things that can be made ready to go by checking a box in the installer.

I found I could get going quite quickly once Cygwin was installed. It gives you a home directory in Windows' `Documents and Settings` folder, and makes it look like a Unix home directory. The full contents of the disk drives are available as `/cygdrive/c`, `/cygdrive/d` etc, and Cygwin also sets up a Unix-like file structure (`/usr/bin` etc.).

You can run either Cygwin or Windows programs from the shell, and there are only a few minor gotchas – for example, if you try 'gcc'-ing a C program and expect the binary to be in a `.out` then it's not, it's in a `.exe`. Also, if you run a Windows native program from within Cygwin then it can occasionally crash unless you first change to the directory of the Windows native program, and then you have to be careful about how you specify pathnames on its command line. In the case of The GIMP I found that the best thing to do is to move the files you want into GIMP's program directory and load them from there. But all these things can be worked around and I haven't run into anything major yet.

You cannot run Linux binaries directly; you have to recompile everything on Cygwin, which can be difficult if you need a large complex package. Thankfully, Cygwin already includes many such packages so you don't have to compile them (just check the check boxes in the installer). If the program you want is not included in Cygwin then it may have a Windows port anyway, which you can often install whether or not you have Cygwin. Most of the major free software projects do this, and they're often better than non-free Windows programs, so it's worth looking.

There are other useful downloadable Windows utilities, such as PuTTY for SSH and SCP (Cygwin includes SSH but it seems it doesn't yet include SCP for some reason), and CutePDF which lets you print to PDF from any Windows application. However I'm glad I have a Linux box as well so I don't want have to set up absolutely everything on that laptop, just the most useful things.

So it seems that Windows on fast hardware can be made to be nicer. And hopefully soon the driver developers will have caught up and it will be fairly easy to move back to Linux. ■

**SILAS BROWN**

Silas is partially sighted and is currently undertaking freelance work assisting the tuition of computer science at Cambridge University, where he enjoys the diverse international community and its cultural activities. Silas can be contacted at ssb22@cam.ac.uk

# Continuous Integration with CruiseControl.Net

## Paul Grenyer explains the CCNet Web Dashboard.

### CruiseControl.Net Web Dashboard

In part 1 of 'Continuous Integration with CruiseControl.Net' [1] I described creating a simple, but effective, continuous integration configuration for Aeryn [2] using CruiseControl.Net.

Another feature of CruiseControl.Net [3] is the Web Dashboard. The documentation describes the Dashboard as follows:

> The CCNet Web Dashboard Application is used for reporting a wide range of information. At one end of the scale it reports summary details of all projects in your organisation and at the other it can give specific metric output for any specific build.

Features of the Web Dashboard include:

- Multi-project, multi-server support – one Web Dashboard deployment can report over all the CCNet projects in your organisation so there is no need for multiple instances of a web application.
- A significant set of 'plugins' to support various reporting and configuration features.
- Plugin API supports custom features at various groupings of build, project, etc.
- Plugin API supports a complete code interface, enabling significantly richer plugins than were available with the old web application.

In this part, I am going to look at some of the Web Dashboard features and at getting the standard web interface installed.

### CruiseControl.Net Web Dashboard features

The Dashboard is accessed via a web browser by specifying the name of the CruiseControl.Net server and the `ccnet` virtual directory (for example, http://localhost/ccnet/). The Dashboard has four views: Farm, Server, Project and Build. The default view, Farm, is shown in Figure 1.

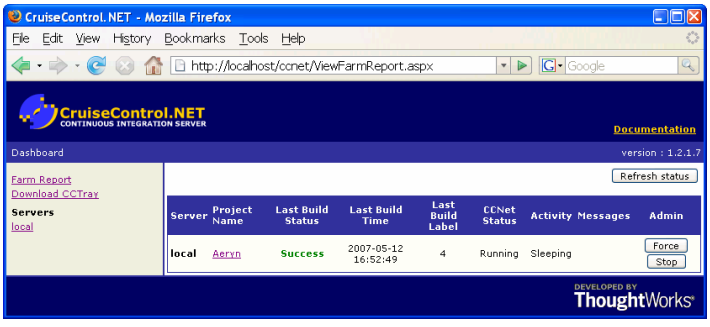
The CruiseControl.Net documentation can be accessed via the link from any view. The location bar near the top of the page shows the user where they are and allows navigation.

#### Farm view

The Farm view shows all the projects that are running on the server and their status and allows them to be stopped and started or a build forced.

There is a link to the CCTray installer. This is useful when CruiseControl.Net is first deployed to a team. Each team member should have CCTray installed on their local machine and this is an ideal way to distribute it.

Figure 1




There is also a link to the Server view and the Project view can be accessed by clicking on the project name.

#### Server view

The server view gives access to the server logs and server information. Being able to access the server logs via the Dashboard negates the need to go onto the server when configuration and build issues need to be investigated. The Dashboard could be improved by showing the server log in real time.

The server information page lists the servers in the system and the CruiseControl.Net software version.

#### Project view

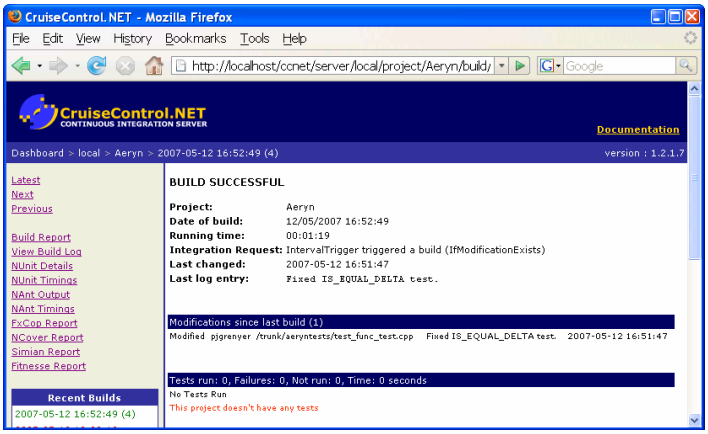
The project view is accessed by clicking on a project name from the Farm view. It gives a list of all the recent builds for the project. Successful builds are colour coded green and unsuccessful builds are colour coded red.

There are links to the most recent build report, the project statistics (if they have been implemented), the server log and the project configuration from ccnet.config.

#### Build view

The Build view is by far the most useful view. Primary it shows the results of latest builds. This is especially useful if for some reason emails are not being sent out or if emails have been deleted and previous build results need to be checked as the build report shows exactly the same information as generated by the email publisher.

Figure 2




Some of the other features of the build view are the build log, which shows the output from the build. The NUnit details and timings which show the results from running any NUnit tests and the output and timings for any NAnt builds. There is also support for third party analysis tools such as FXCop [4] and NCover [5] that I intend to cover in future articles.

**PAUL GRENYER**

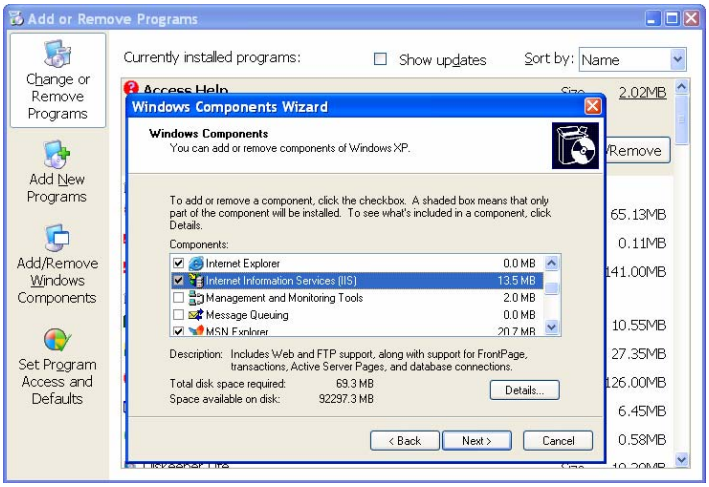
Paul has been a member of the ACCU since 2000. He founded the ACCU Mentored Developers and serves on the committee. Paul now contracts at an investment bank in Canary Wharf.

## Installing Internet Information Services

The Dashboard is implemented in ASP.Net and requires Microsoft's Internet Information Services [6] to run on the same machine as CruiseControl.Net. IIS is included as a package with Windows XP Professional, but is not installed by default. It can be installed via the Windows Component Wizard which is opened by pressing the Add/Remove Windows Applications button on the Add or Remove Programs window accessed from the Windows Control Panel. Put a tick in the Internet Information Service (ISS) box and click next (Figure 3). When the setup has finished click Finish.

To test that ISS is installed and working, open a web browser and enter the URL http://localhost/ (you can also use the host name or IP address). If everything has worked a page will be displayed with the message: Your Web service is now running. Windows XP Professional includes version 5.1 of IIS. Later versions have a different default page with a similar message.

Figure 3




Some people consider IIS a security risk so it is important to make sure you are also running the latest Microsoft Windows updates and check with the relevant system administrators before installing it.

## Configuring the Web Dashboard for IIS

If IIS was installed prior to CruiseControl.Net and the default install options were not changed, then the Web Dashboard will already be configured for IIS. If IIS was not already installed or the right install options were not selected, the easiest way to configure the Dashboard is to rerun the CruiseControl.Net installation. This will create the necessary IIS virtual directory.

Start by backing up `ccnet.config` and `ccnet.exe.config` from the CruiseControl.Net server directory (usually located in `C:\Program Files\CruiseControl.NET`). The Dashboard only installation should not overwrite the existing configuration, but it is better make a backup just in case.

1. Run the CruiseControl.Net MSI installer from the CruiseControl.Net website.
2. On the Choose Components screen uncheck CruiseControl.net Server and Examples (leaving only Web Dashboard checked).
3. Leave the Additional Components screen unchanged (ensuring that Create virtual directory for Web Dashboard is ticked).
4. Complete the installation.

To check that the virtual directory has been setup correctly, make sure CruiseControl.Net is running from the command line or as a service (the Dashboard works with either) and then go back to the web browser and enter the URL http://localhost/ccnet/. IIS must be configured for ASP.Net, so this will bring up the source for the default Dashboard ASP page which begins:

```
<!--
Note to people reading source code -
CruiseControl.NET includes an HttpHandler which
handles all .aspx requests. This file,
default.aspx, should never be processed by 'normal'
ASP.NET and is here just as a page to explain
configuration problems
-->
...
```

There are instructions on the CruiseControl.Net website for manually configuring the Dashboard for IIS, but they're not particularly comprehensive and using the installer is much easier.

## Configuring IIS for .Net

When multiple versions of .Net are installed, IIS needs to be configured to use a particular version. To do this open a command prompt and go the directory (e.g. v2.0.50727) of the .Net version to be used, which is usually in `C:\WINDOWS\Microsoft.NET\Framework` and type:

```
aspnet_regiis.exe -i
```

`aspnet_regiis.exe` is an ASP.Net registration tool. The `-i` parameter, according to the help, installs Asp.net and updates the scriptmaps at the IIS metabase root and for all scriptmaps below the root. Existing scriptmaps of lower versions are upgraded to the same version.

The Dashboard and ISS configuration is now complete. Going back to the web browser and refreshing to re-entering http://localhost/ccnet/ will bring up the Web Dashboard. ■

## References


1 Integration with CruiseControl.Net - Part 1, available from: http://accu.org/index.php/journals/1371
2 http://www.aeryn.co.uk/
3 http://ccnet.thoughtworks.com/
4 http://www.gotdotnet.com/Team/FxCop/
5 http://www.ncover.org
6 http://www.microsoft.com/windowsserver2003/iis/default.mspx

# Scripting C++ Objects

## Our Gang of Four bless COM with perl.

The first part of this article demonstrated how to provide a platform-independent and language-neutral calling convention based on metadata contained in C++ objects. In the second and concluding part we show how to provide a target language binding for Perl. In addition, we describe a novel technique for reducing verbosity in the target language that also facilitates the API's implementation through C++ mixin techniques.

## Perl to COM binding

### Extending Perl

We make the COM objects accessible to Perl by creating an extension library. This consists of a DLL exporting C++ functions which bind the Perl calls to the COM methods and some convenience functions in Perl. We derived this from a published extension which adds OLE support on Windows [1], removing excess functionality and modifying to call our COM libraries. This allowed us to rapidly prototype the scripting and we learned valuable lessons from this approach, but the refactoring work was extensive and we are likely in future to build extensions from scratch.

You can run the Perl utility **`h2xs`** [2] to generate the framework for a new extension. The generated files include a `.pm` package to contain the convenience functions and a `.xs` source file for the C++ binding code. The `.xs` file is preprocessed to add function interfaces and variable marshalling code ready for compilation with your usual compiler. In its simplest form, it contains little more than SWIG-style declarations of your library functions, but you can add complete function definitions in C++. Since there can be only one `.xs` file per extension, and the Perl header files define some macros which interfere with C++, it pays to factor out the pure C++ parts into other files and leave only the interface functions in the `.xs`; otherwise it quickly becomes unwieldy.

The XS preprocessor also generates Perl code which locates and loads the extension DLL when a script imports the class package. Once the DLL is loaded, it calls the **`BOOTSTRAP`** function exported from the `.xs` file. You can add your own initialization code to this function; for instance, it could make the COM system ready for the script to access objects.

### Creating Perl objects

The various classes exported by our COM libraries are represented by a single Perl class called **`COM`**, which appears polymorphic by virtue of the C++ forwarding mechanism.

Perl has three basic datatypes: scalars, arrays and hashes. A scalar variable can hold an integer, floating point or ASCII string value, or it can be a reference to another datatype or even a block of code. Typing in Perl is weak and it will silently convert a scalar value between its possible types according to context. Arrays are vectors of scalars and hashes are maps with string keys and scalar values.

An object variable is represented by a scalar which is 'blessed', that is, tagged with a package name; the package name is associated with a set of functions which form the class methods. The instance data is often stored directly in a hash referenced by the scalar, which makes it easy to implement the class purely in Perl code. However, we needed to define objects which bind to our C++ classes, so we implement some methods in the compiled part of the extension.

Class definition is not completely formalised in Perl, but it is conventional to provide a function called **`new`** in the package to act as the class constructor. In our case, **`new`** takes a single parameter, which is the **`ProgID`** (a string uniquely identifying the COM class to be created). A COM object of this type is created and the resulting smart pointer is retained. A new Perl hash variable (HV) is also created and blessed into the COM package. All Perl variables representing the C++ objects are of this class, although it would be possible to subclass the Perl objects according to the COM class if this proved useful. A new entry in a locally-defined lookup table associates the HV with the COM pointer so that the COM object can be retrieved when methods are called against the Perl object. Finally, we make a new Perl scalar variable (SV) which references the HV and return this to the script.

Perl variables are reference counted. Their scoping rules are similar to C++'s, so for instance a variable declared in a code block will drop out of scope at the end of the block. When this happens, the class's **`DESTROY`** function is called; our implementation of this reduces the reference count on the COM object to ensure that resources are released as soon as possible.

### Calling methods

Now that we have an object variable, we want to call methods against it. Any methods known at build time can be mentioned explicitly in the `.pm` or the `.xs`, but the COM interface methods are discovered at run time and we cannot list them in the extension. Instead, we can implement a general method called **`AUTOLOAD`**. An unrecognized method call such as

```
$atom->Translate($x, $y, $z);
```

will be passed to **`AUTOLOAD`**, along with the parameters **`$x`**, **`$y`**, **`$z`**. In our extension, **`AUTOLOAD`** is implemented in Perl for ease of writing and it passes the call onto a C++ function called **`Dispatch`** – Listing 1.


**KEVIN DIXON**

Kevin has been developing scientific applications in C and C++ since 1991, filling in the gaps with Perl. He is currently modelling polymers and other large molecular systems. He can be contacted at kdixon@accelrys.com

**JOHN LEAR**

John has been developing software in C++ on various platforms for the last 14 years. At Accelrys, he works on client development and infrastructure, hopefully making life easier for other developers. He can be contacted at jlear@accelrys.com

**SIMON GRAY**

Simon is an escaped experimental physicist who was spotted around financial programming for a few years before finding a more natural home writing scientific software. He can be contacted at Simon.Gray@Physics.org

**SIMON TREW**

Simon started his career as a software engineer in the defence industry. After graduating from UMIST, he produced object-oriented databases, before joining Accelrys. He can be contacted at strew@accelrys.com

Listing 1

```
package COM
{
  sub AUTOLOAD {
    my $self = shift;
    $AUTOLOAD =~ s/.*::/o;
    $self->Dispatch($AUTOLOAD, my $retval, @_);
    return $retval;
    }
}
```

The **$AUTOLOAD** variable contains the name of the method, prefixed with the class name (**COM::**); the class name is removed before **Dispatch** is called. **@_** is a list of the parameters passed to **AUTOLOAD**.

The **Dispatch** function in turn calls **GetIDsOfNames** to convert the method name to a **DISPID**. If the method is found, it is called through **Invoke**; otherwise, an error message of the form **"unrecognized function 'Translate' at line 31";** is sent back to the user.

The standard Win32 implementation of **Invoke** (and our variations) requires the parameters to be passed as **VARIANT**s. We create a vector of **VARIANT**s and populate them with mapped versions of the input Perl variables. Note that **Invoke** expects this array to be in the opposite order from how the variables are listed on the Perl stack, so that the first **VARIANT** represents the right-most argument in the Perl parameter list. The **VARIANT**s live until the **Invoke** call returns, after which their contents are released before control returns to the script.

The type mapping of the parameters is quite naive: a variable containing a floating point value is stored in the **VT_R8** (double) member of the **VARIANT**, an integer is placed in the **VT_I4** (long) and a string is converted to a **BSTR** and placed in the **VT_BSTR** member; an array reference is unpacked and inserted into a new **SAFEARRAY** as that is how our COM components pass arrays, though another implementation might easily pass a reference to a **std::vector**.

We can use this naive approach because the **Invoke** function will fine-tune the parameters to better match the COM method interface. From examining the Perl variable, we might know we have a floating-point number but we don't know enough about the COM method to determine whether this should be a float or a double; we certainly wouldn't want our script to have to specify this as we would lose the advantages of weak typing. Invoke has access to the COM method interface definition and can do some further type translation. For instance, it can convert an integer (or even a string containing a textual representation of a number) to a float without complaint if the C++ function requires it.

If **Invoke** succeeds, the retval **VARIANT** is unpacked and mapped to a new Perl variable for return to the script.

If one of the Perl parameters is a reference to a simple datatype (as can be specified in the script with a backslashed variable: **\$atom**), the typemapping code takes this to indicate that the COM interface is expecting an [out] parameter; we take care to distinguish such a reference from an object instance, which is a reference to a blessed hash variable. In this case, the Invoke **VARIANT** is expected to contain a reference to a second **VARIANT** which will itself be populated by the COM method; on return from **Invoke**, the value in the second **VARIANT** must be sent back into the corresponding Perl variable. Although we can therefore call through interfaces featuring [out] parameters, we have decided not to use such interfaces in our applications. Instead, we make use of Perl's ability to return multiple values from a subroutine as an array, to keep the scripts simple. A COM method returning multiple values can fill the [retval] **VARIANT** with a **SAFEARRAY** object, which is mapped into a Perl after the **Invoke** call.

### Flexible environments

Although the script can create COM objects of any type for which the **ProgID** is known, in practice we restrict the use of the new function to a single class, the **Application**. Other objects are created and accessed through methods on the **Application**. The **Application** is implemented with singleton data members so that scripts running in the same process can create several **Application**s and yet access the same contained objects.

The benefit of this is that we can control the Perl environment without affecting the script. The script is usually run from within our server application, which has a Perl interpreter embedded in it. The server creates an **Application** object and initializes its children, which can be adjusted to suit the server environment. When the script makes a new **Application**, it gains access to the **Application**'s children. However, it is equally possible to run the script outside of the server, using a standard Perl installation. This allows us to reuse the server components and scripts within another of our applications, a workflow manager which coordinates disparate programs by running scripts within its own environment. A second use of this flexibility is that the scripts can be run from a Perl debugger: users gain the ability to debug their server scripts without us having to build an expensive debugging application.

Conversely, we now have the option of embedding a Python interpreter in the server, which will run alongside the Perl interpreter. Once again, since the underlying COM objects are shared between the scripts, it will be possible to interleave Perl and Python scripts; a customer may prefer to write in Python, but they will be able to use their own scripts in conjunction with Perl scripts posted on our scripting forum.

## Dynamic Discovery (Forwarding)

What has been covered so far will easily support a rich API. Dynamic Classification, or Forwarding as it became known, is a fairly simple idea but came to be significantly more powerful than originally anticipated. The first form of forwarding is how it was originally anticipated to be used. Typically, if one has a significant object hierarchy, there will be a certain amount of navigation that is required to get from a reference you have to a function or property you want to access. This leads to long chained function calls:

```
$x = $myObj->GetTheWibble->DoIt->FooBar->Length;
```

Our aim was to reduce the verbosity and therefore make the scripting object model simpler to understand. This would be achieved by ascertaining context from the previous calls that have occurred and simply forwarding, or passing on, the calls from one object to another. A simple example is similar to that used by Excel for the currently active worksheet. A call:

```
$workbook->Cell(0,0);
```

would be forwarded on to the active worksheet object for it to deal with. Another benefit of this idea is the ability for the target object, the workbook in the previous example, to delegate to the implementation in a completely different object without this being apparent to the script writer. This allows us to develop mixin classes to implement particular areas of functionality that can be reused.

To implement forwarding a class's type information is updated to contain an extra piece of information which causes **GetIDsOfNames** to call an additional function that returns a dispinterface which is then queried for the requested name. For example, consider the previous script example, the Workbook class would look something like Listing 2.

Listing 2

```
class Workbook :
   public IDispinterfaceImpl<Workbook>
  {
  public:
  ...
    DECLARE_PROPGET(ActiveSheet, Sheet *) const;
    ...
    BEGIN_CLASSINFO(Workbook)
      ...
      FORWARD_TO_PROPGET(ActiveSheet)
    END_CLASSINFO
  };
```

`Cell` would not be found during a query of the `Workbook` classes type information. The final forward declaration means that the `ActiveSheet` property is retrieved and `GetIDsOfNames` for `Cell` called on the `Sheet` object.

The second form of forwarding turned out to be the most powerful. Rather than working with context, this forwarder is in many ways similar to the default clause of a switch block. If a call cannot be satisfied through the API of the current object, the call is forwarded to a default method that is able to look at the name of the property/method called and decide what to do with it.

This gives the ability to look into application metadata to satisfy the call. This allows an 'Expando Property' configuration without having to use a different syntax for these properties over that used for the set of properties known a priori (Listing 3).

Listing 3

```
class Atom : public IDispinterfaceImpl<Atom>
{
public:
  ...
  BEGIN_CLASSINFO(Atom)
    ...
    FORWARD_TO_METHOD(ExpandoPropertyHandler)
  END_CLASSINFO
  IDispatch *ExpandoPropertyHandler() const;
};
```

The `ExpandoPropertyHandler` method simply returns an intermediate object whose sole responsibility is to lookup property names and support accessor requests. It only needs to support a very basic subset of the `IDispatch` interface. To keep it simple, this is handled through a base class that simply stubs out the methods that aren't used. A basic implementation of `GetIDsOfNames` delegates the allocation of a `DISPID` for the property name to the `GetIDOfName` method in the derived class (Listing 4).

The final point of flexibility given through forwarding is the ability to examine the arguments and types to a named function or property accessor

Listing 4

```
class ExpandoPropertyImpl :
   public BasicIDispatchImpl
{
public:
  DISPID GetIDOfName(
     const std::string& propertyName);
  VARIANT Invoke(const DISPID       dispIdMember,
                 const unsigned int wFlags,
                 const DISPPARAMS   *pDispParams,
                 EXCEPINFO          *pExcepInfo,
                 unsigned int       *puArgErr);
};
```

Listing 5

```
VARIANT ExpandoPropertyImpl::Invoke(...)
{
  VARIANT ret_val;
  if(wFlags == PROP_GET)
  {
    ret_val = GetPropertyForID(dispIdMember);
    if(pDispParams->cArgs != 1 &&
       ret_val.vt == VT_DISPATCH)
       ret_val->ppdispVal->Invoke(DISPID_VALUE,
       ...);
  }
  else if(wFlags == PROP_PUT)
  ...
  return ret_val;
}
```

and then decide what to do. It is important to note at this point that COM does not support function overloading so this behaviour could not be supported through the declarative style used with COMLite. This aspect is implemented during the `Invoke` call (Listing 5).

This approach is very similar to the idea of default methods as used in VisualBasic. The example given here shows how verbosity can be reduced in a user's script without reducing readability. For example, without the overloading on parameters the following script text would be required:

```
my $atom = $molecule->Atoms->Item(0);
```

but by overloading the arguments we can elide the `Item` call:

```
my $atom = $molecule->Atoms(0);
```

The only requirement for this is that a special `DISPID` is used to identify the method to be called by default.

A class can contain any number of active forwarders. This convenient technique allows a class to implement one aspect of an object's behaviour, such as Expando Properties, and that class be reused through forwarding to provide mixin style behaviour. As with all techniques where magic happens, cf. operator overloading in C++, care must be taken so that a consistent and unsurprising API is produced.

## Frosting

### Generic perl modules

Having built the basic COM interfacing extension, we can use some of Perl's advanced capabilities to ensure that the scripts remain easy to write and do not reveal the complexities of our underlying datamodel.

### Deferred call to allow indexed property access

We wanted to make it easier to access the cells in our spreadsheet component. The COM object exposes methods `get_Cell([in] int row, [in] int column, [out, retval] double value)` and `put_Cell([in] int row, [in] int column, [in] double value)` which would be called in the script like this:

```
my $energy = $workbook->Cell(4, 2);
$workbook->Cell(4, 2, $energy);
```

Notice that the `put_` and `get_` warts are missing from the Perl calls; the `Invoke` function decides which COM method to call according the number of parameters it is passed (two for `get_Cell`, three for `put_Cell`).

We felt that the setting call would be more clearly expressed as

```
$workbook->Cell(4, 2) = $energy;
```

The left-hand side of the assignment involves a call to the `AUTOLOAD` function; this was interpreted as a syntax error until we marked `AUTOLOAD` as an 'lvalue' [3] function.

There remained a problem with deciding which method to call: Perl parses the left-hand side as a call to the two-parameter `Cell` method and therefore inappropriately selects the `get_Cell` method. We considered changing `get_Cell` to return a new 'Cell' object which would be a reference into the workbook, capable of being assigned to or of returning the cell value according to context. However, this would have entailed a new class for each type of assignable property and the code would have to reside on the C++ side, an approach which was deemed too intrusive and was rejected.

Fortunately, Perl provides the powerful concept of tied variables.

```
tie my $invoker, 'DeferredCallTie',
   $deferredCall;
```

This creates an object `$invoker` of class `DeferredCallTie`, which implements three methods: a constructor called `TIESCALAR` and read and write accessors `FETCH` and `STORE`. (Listing 6)

Listing 6

```
package DeferredCallTie;
{
  use Exporter;
  use base qw(Exporter);
  use Scalar::Util qw(refaddr);

  # Data members for 'inside-out' class.
  my %deferredCall; # Instance of DeferredCall
                    # which we control.
  # Constructor; stores the DeferredCall,
  # which it will later invoke.
  sub TIESCALAR {
    my ($class,
        $deferredCall # DeferredCall object which
                      # we will invoke.
        ) = @_;
    my $newObject = bless \do{
       my $anon_scalar}, $class;
    my $selfId = refaddr $newObject;
    $deferredCall{$selfId} = $deferredCall;
    return $newObject;
  }

  # Calls the method in an rvalue context.
  sub FETCH {
    my ($self) = @_;
    my $selfId = refaddr $self;
    return $deferredCall{$selfId}->Call();
  }

  # Calls the method in an lvalue context,
  # which assumes that the method is a propput
  # taking $value as the new property value.
  sub STORE {
    my ($self,
        $value
        ) = @_;
    my $selfId = refaddr $self;
    $deferredCall{$selfId}->PutProperty($value);
  }
  # The destructor removes this object from
  # the class hash.
  sub DESTROY
  {
    my ($self) = @_;
    my $selfId = refaddr $self;
    delete $deferredCall{$selfId};
  }
}
```

We changed the **AUTOLOAD** function so that instead of immediately calling the **Dispatch** function and passing its return value back to the script, an object of type **DeferredCall** saves a description of the method call (its object, the name of the method and a list of the input parameters); the call to **Dispatch** is deferred until the calling context becomes clear. This **DeferredCall** is in turn wrapped in a new **DeferredCallTie**, which is returned to the script (Listing 7).

Now when the script assigns the energy to the worksheet cell, it calls **DeferredCallTie::STORE** with the energy variable as a parameter. Since **STORE** is necessarily called in an **lvalue** context, it knows to call **DeferredCall::PutProperty** and the **put_Cell** method is correctly selected. Conversely, when the script reads the cell value, it goes through **DeferredCallTie::FETCH**; this has no new parameters to add, so the two-parameter method **get_Cell** is selected through the normal call to **Dispatch**. If the **DeferredCall** is wrapping a method call rather than a property accessor, it will be constructed in a void context:

```
$workbook->ClearCell(4, 5);
```

Listing 7

```
sub AUTOLOAD : lvalue {
  my $self = shift;
  $AUTOLOAD =~ s/.*::/o;

  # This defers the call until we have the
  # full context.
  my $deferred = new DeferredCall($self,
                                  $AUTOLOAD, @_);
  # If we're called in void context, we need to
  # call the method now, as there is nothing in
  # the caller to prompt the tie to do so.
  if (!defined wantarray) {
    $deferred->Call;
    return;
  }
  # Otherwise, wrap the deferred call in a tied
  # scalar which calls the method once the
  # context is resolved.
  tie my $invoker, 'DeferredCallTie', $deferred;
  # We cannot use "return" in an lvalue context,
  # instead we just mention the value.
  $invoker;
}
```

In this case, there is no value access to prompt the **DeferredCallTie** to invoke the method, so **AUTOLOAD** does it directly.

#### Operator overloading

Perl supports operator overloading at the class level; this allows us to provide useful behaviour without having to write bespoke functions on our COM objects and we can make our objects behave the way the client would naturally expect. For instance, printing a Perl class object usually gives something like this:

```
print $atom;
> COM=HASH(0x2b07984)
```

which is ugly and not very informative to the user. The hex number is the address of the Perl hash variable wrapping our COM object, so different Perl variables referring to the same COM object print different values.

We overload the string conversion operator so that it prints the type and the address of the underlying COM object (Listing 8).

**AddressOf** and **TypeOf** access the C++ object and are implemented in the extension. The same object now prints as:

```
print $atom;
> Atom [0x0045a1c4]
```

and a test of equality of the Perl variables (which uses the string conversion) will correctly return **true** if they refer to the same wrapped object.

Listing 8

```
package COM;
{
  use overload '""'  => \&ConvertToString;
  sub ConvertToString {
    my $self = shift;
    my $address = AddressOf($self);
    if ($address) {
      return sprintf "%s [0x%08x]",
         TypeOf($self), $address;
    }
    else {
      # Null objects evaluate to 'false' to
      # allow this: if($object) {$object->Action;}
      return 0;
    }
  }
}
```

### Application specific packaging

The Perl script writer needs to access the extension somehow and this is achieved by using a Perl module that bootstraps it. By using certain Perl idioms it is possible to make this process largely transparent to the script writer. A number of special subroutines can be present in a module. They are called automatically by the Perl runtime as a result of a use package declaration and they act as package constructors and destructors. The `INIT` method is used to perform any initialisation required by the module.

#### Creating objects

New C++ objects are created using the COM package function `new` and passing the `ProgID` of the type of object to be created:

```
my $atom = COM->new("Atomistic.Atom")
```

However, this is not particularly novice programmer friendly. Our approach was to expose an object model that follows the style used by Microsoft Office applications. The root object in this case is of type Application and is automatically created during the `INIT` method described previously.

Access to all other objects then follows from this root. New objects are created using a factory method on an existing object:

```
my $atom = $document->CreateAtom("Si", ...)
```

#### Selective exposure through the Exporter

The final component that we use is the Perl Exporter. This package controls the external visibility of symbols declared within a package. The Application object, created during by the `INIT` function, is simply exported into the `MyAppsAPI` namespace. Since virtually every call into our API would start with `$Application` we provided a number of functions to provide deeper access into the object model to reduce verbosity. Our package declaration then looks something like:

```
package MyAppsAPI;
use Exporter;

our @ISA         = qw(Exporter);
our @EXPORT      = qw($Application);
our @EXPORT_OK   = qw(
                Documents Tools Settings etc.);
```

To again reduce the amount of verbiage, we group the contents of `EXPORT_OK` into a tag that makes it easy for the script writer to import everything into their script:

```
our %EXPORT_TAGS = ('all' => [ @EXPORT_OK ] );
```

Using all of these techniques means that to get access to `MyAppsAPI` functionality simply requires a single line of Perl:

```
use MyAppsAPI qw(:all)
```

This technique still gives the flexibility to require the package name be specified if name clashes occur. Notice also that the contents package are very small and require little, if any, maintenance.

## Conclusion

This article has outlined the basics of a novel implementation of an API suitable for using from a scripting language, such as Perl or Python. The core implementation allows the API writer to write code easily in a native C++ style.

The approach of using an independent wire representation (COM) also means that additional scripting languages can be supported through writing an independent language binding should there be customer demand.

The above techniques were used to develop the scripting feature in the recent successful commercial release of Materials Studio. In addition we are now using it to drive automated regression tests and as part of our integration with alternative consumer applications. ■

## References

[1] Win32::OLE – http://search.cpan.org/~jdb/libwin32-0.26/OLE/lib/Win32/OLE.pm

[2] Perl XS – http://perldoc.perl.org/perlxstut.html

[3] Perl lvalue – http://perldoc.perl.org/perlsub.html#Lvalue-subroutines-lvalue-subroutine%2c-lvalue

# Get Ready for the ACCU 2008 Conference

*"My fellow ACCUers, ask not what your conference can do for you, ask what you can do for your conference".*

Freely adapted from John F. Kennedy.

It might seem too early to think to the next ACCU conference – after all it will be in April next year – however, at the time you will be reading this the call for proposals will be out, and the conference committee will be already busy deciding various aspects of the program (keynotes, tracks, etc.).

Despite all the hard work, there is still something the committee cannot provide: your contribution.

In fact, the "big names" make only part of the program, and some of the most successful sessions are the ones presented by practitioners who may not be very famous, but have very interesting and useful things to say (Allan Kelly wrote already something along these lines in 2006 in his blog [1]).

Presenting at a conference can be challenging – speaking in public is never easy, especially for first timers, and doing it in front of an audience of experts is even more difficult – but is also very rewarding. Ric Parkin's article [2] gives a very good description of what it is like.

Furthermore, if you get a proposal accepted you have the right to attend the conference at a discount price, and your boss may be more inclined to give you the time off and pay for it – not to mention the fact that it looks very good on your CV.

If none your proposals is accepted – unfortunately, we have limitations of space and time – you still win: you will have learnt something new, and also have the material for one or more articles on CVu or Overload.

Personally, I think all the above are good enough reasons to give it a try.

All you need to do now is to check out the conference web-site for the details (http://www.accu.org/conference), then prepare and send your proposals at conference@accu.org. I'm looking forward to receiving them, and to seeing you all in Oxford next April.

Giovanni Asproni
Conference Chair

## References

1 Kelly, Allan, http://blog.allankelly.net, April 2006

2 Parkin, Ric 'Stand and Deliver', *CVu*, June 2007

# Introduction to Lua

Renato Forti introduces another scripting language.

## What is Lua?

Lua is an embeddable scripting language. It was born in 1993, developed by Roberto Ierusalimschy, Waldemar Celes and Luiz Henrique de Figueiredo at Lua.org, a laboratory of the Department of Computer Science of PUC-Rio (the Pontifical Catholic University of Rio de Janeiro in Brazil).

Today Lua is widely used in all kinds of applications and is considered a general-purpose language. Application types include: industrial applications, ethernet switches, distributed business applications, image processing, bioinformatics, web development, games and so on.

Some projects that use Lua [1]:

- **Far Cry**, a first-person shooter. Lua is used to script a substantial chunk of the game logic, manage game objects (Entity system), configure the HUD and store other configuration information. (http://www.farcry-thegame.com/)
- **Adobe Photoshop Lightroom** uses Lua for its User Interface (http://www.adobe.com/products/photoshoplightroom/)
- **Lua Player** is a port designed to run on Sony Computer Entertainment's PlayStation Portable to allow entry-level programming. (http://www.luaplayer.org/)

Lua was designed to be small, simple, portable, extensible and very powerful.

Lua is written in ANSI C, so can run on a variety of operating systems, including MS Windows, Unix, Mac OS9, Mac OSX, PlayStation, X-Box (yes, Lua is used a lot in Game Development), ARM, RISC, IBM Main Frames and a lot of others. For this article I will be using LUA on MS Windows.

Lua is free, open source (distributed under a liberal license – MIT license – http://www.lua.org/license.html), you can use it in your commercial applications at absolutely no cost.

When this article was written the the latest version of Lua was 5.1.1; you can download it from: http://www.lua.org/download.html

## Building Lua

After downloading you will need to unpack `lua-5.1.1.tar.gz`. I use WinRar to this. You can use your favourite compression utility program.

I will build Lua using Visual Studio 2005, but you can use any compiler that supports ANSI C.

I extracted the contents of the compressed file into the directory `c:\temp\lua-5.1.1`. To build, invoke your 'Visual Studio 2005 Command Prompt':

```
> cd \temp\lua-5.1.1\src
> ..\etc\luavs.bat
```

This will build Lua creating: `lua51.dll`, `lua51.lib` and `lua.exe` (stand-alone interpreter) in `<lua dir>\scr`.

Note: to build on UNIX, or for other build details see `<lua dir>\INSTALL file Setting`


**RENATO FORTE**

Renato Forte is a C++ programmer who works at Vegas Card, a financial company located in Brazil. He can be contacted at re.tf@acm.org


## Lua overview

### Configure environment

I will now use the stand-alone interpreter to dive into Lua. I will make a batch file to automate execution of the interpreter and make my life easier, and I will use Notepad++ to edit my Lua script. You can get Notepad++ from http://sourceforge.net/projects/notepad-plus/. The batch file needs to reside in the same directory as Lua, in my case `c:\temp\lua-5.1.1\src` or, if you wish, you can add `<lua dir>\src` to your PATH.

The batch file contains:

```
lua test.lua
pause
```

### First view

Create a TXT file in this directory and name it `test.lua`, open it in Notepad++ and write:

```
print("-------");
print("Hi Lua!");
print("-------")
```

Save, press F5 or choose Run on Notepad++ Menu and select `test.bat`, or run your `test.bat` directly. You will see:

```
C:\temp\lua-5.1.1\src>test.bat
-------
Hi Lua!
-------
C:\temp\lua-5.1.1\src>pause
Pressione qualquer tecla para continuar. . .
```

Note that I didn't end the last line in `test.lua` with a semicolon. Lua needs no separators between consecutive statements; for example, this is valid in Lua:

```
a = 10 b = 20 c = 30 d = 40 e = 50 print(a+b+c+d+e)
```

The result is the same if you use semicolons:

```
a = 10; b = 20; c = 30; d = 40; e = 50;
print(a+b+c+d+e);
```

However using semicolons is more readable/elegant.

### Fundamental types

There are no type definitions in Lua. Lua is a dynamically typed language. Each value carries its own type. (Listing 1).

As well as number, Boolean, string and nil, Lua has four other basic types and they are: table, function, thread and userdata.

Lua doesn't have the concept of class. You can simulate this concept using metatables, but this is out of scope for this discussion.

About comments. In Lua you can comment your code using `--` for a single line, and for multi-line comments using:

```
--[[
multi-line
comment
--]]
```

Listing 1

```
x = 10;
print(type(x)); -- print number

x = true;
print(type(x)); -- print boolean

x = "Oi!!!";
print(type(x)); -- print string

x = nil;
print(type(x)); -- print nil

--[[
type(x) function returns the
type name of an value.
--]]
```

### Number

This type represents real numbers.

```
x = 10;
y = 10e+10
z = 0.4

print(x+y+z);
```

### Boolean

Possible boolean values are **true** or **false**. If the boolean not is initialized it carries **nil**, and if you test it you will get **false**.

```
if (x == true) then
  print("TRUE");
else
  print(x);
end

x = true;

if (x == true) then
  print("TRUE");
else
  print(x);
end
```

Lua is case sensitive, so **TRUE** is a variable name but **true** is a reserved word.

### Nil

All undefined variables in Lua have the value **Nil**. **Nil** is used to represent the absence of a defined value.

### String

String is a sequence of characters and it can contain escape sequences (like C):

```
x = "1 - line 1\n2 - line 2\n3 - line 3\n";
print(x);
```

Available escape sequences are shown below.

| | | | |
|---|---|---|---|
| \b | back space | \t | tab |
| \f | feed | \\ | backslash |
| \n | new line | \" | double quote |
| \r | return | \' | single quote |

You can use **[[** and **]]** to enter multi line strings:

```
x = [[
1 - line 1
2 - line 2
3 - line 3
]];
print(x);
```

### Tables

Table is like a **std::map**, see Listing 2.

Listing 2

```
x = {}; -- create table
x[1] = "oi";
x[2] = "hi";
print(x[1]);
y = {};
y["oi"] = 1;
y["hi"] = 2;
print(y["hi"]);
print(y.hi); -- same thing as last line
```

### Operators

Lua has arithmetic, relational, logical and concatenation operators.

- Arithimetic

  Lua provides seven arithmetic operators, shown in the table below. Listing 3 shows an example of their use.

| | | | |
|---|---|---|---|
| + | binary addition | ^ | exponentiation |
| - | binary subtraction | % | modulo |
| * | multiplication | - | negation |
| / | division | | |

Listing 3

```
x , y, z = 10, 20, 30;

print(x+y+z); -- 60
print(x-y-z); -- -40
print(x*y*z); -- 6000
print(x/y/z); -- 0.016...
print(x^2); -- 100
print(x%y%z); -- 10
```

- Logical

  The logical operators are: **and**, **or** and **not**. These operators always result in **true** or **false**.

  All logical operators consider **nil** as **false** and anything else as **true** (Listing 4).

Listing 4

```
x , y, z = 10, 20, nil;

print(x and y); -- 20
print(x and z); -- nil

print(x or y); -- 10
print(x or z); -- 10

print(not y); -- false
print(not z); -- true
```

- Relational

  The relational operators are:

| | | | |
|---|---|---|---|
| < | greater than | >= | greater than or equal to |
| > | less than | == | equal |
| <= | greater than or equal to | ~= | not equal |

These operators always result in **true** or **false**, as shown below:

```
x , y, z = 10, 20, nil;

print(x>y); -- false
print(x<y); -- true
print(x>z); -- attempt to compare nil with
            -- number
```

- Precedence

  higher

| ^ | not | - (unary) | * |
|---|---|---|---|
| / | % | + | - |
| .. | < | > | <= |
| >= | ~= | == | and |
| or | | | |

lower

```
print(10+10*2^2); -- 50
-- is like
print(10+(10*(2^2))); -- 50
```

## Loops and conditionals

Let's start with **if**. The **if** statement works like the C statement, first test the **if** part then execute either the **then** part or the **else** part (the **else** part is not needed, it is optional). You need to finish **if** with **end** (Listing 5).

Lua has no **switch** statement, instead **elseif** is used to get similar functionality (Listing 6).

The **while** statement behaves as you'd expect with the loop only being executed if the condition evaluates to true. (Listing 7)

In **repeat** the body is always executed at least once. (Listing 8)

The **for** works as usual, but with some percularities, and it has two variants, the numeric and the generic **for** (Listing 9). This loop will execute 10 times, start at 10, finish at 1, and decrement -1 each time through the loop.

The generic loop uses integrator functions and I will not illustrate these in this article.

## Functions

In lua, functions can return one or more values. (Listing 10)

In the last line I used **..** between **result_a** and **" + "**; this is the way to concatenate two or more strings. If you want to concatenate numbers, you must use spaces between **..** and the numbers:

```
print(5 .. 5);
-- print(5..5); this will cause error.
```

Now we have a small background in 'Lua' structure and in the next article I will dive into C API. ■

Listing 5

```
a, b = true, false;
if(a ~= b) then
  print("a isn't equal to b")
end
x, y = 10, 20
if(x > y) then
  print("x > y")
elseif (x < y) then
  print("x < y")
end
x, y = y, x; --  swap x for y
if(x > y) then
  print("x > y")
elseif (x < y) then
  print("x < y")
end
```

Listing 6

```
a = 1;
if a == 1 then
  print("1")
elseif a == 2 then
  print("2")
elseif a == 3 then
  print("3")
elseif a == 4 then
  print("4")
elseif a == 5 then
  print("5")
elseif a == 6 then
  print("6")
end
```

Listing 7

```
a = 1;
while a do
  print(a);
  a = a+1;
  if(a == 10) then a = false end;
end
```

Listing 8

```
a = 0;
repeat
  print(a);
  a = a+1;
until a >= 10
```

Listing 9

```
for i=10, 1, -1 do
  print(i)
end
```

Listing 10

```
-- very simple function that return 3 values
function add(a , b)
  return a, b, a+b;
end

--call function
result_a, result_b, result_ab = add(4, 5);
print(result_a .. " + " .. result_b .. " = " ..    result_ab);
```

## Note

1 Source: Wikipedia - http://www.wikipedia.org/, Lua.org - http://www.lua.org/uses.html

## Bibliography

Roberto Ierusalimschy, *Programming in Lua – 2nd ed.* (http://www.inf.puc-rio.br/~roberto/pil2/), ISBN 85-903798-2-5

Lua.org: http://www.lua.org/

Mailing list: http://www.lua.org/lua-l.html

Community: http://www.lua.org/community.html

Wikipedia, http://en.wikipedia.org/wiki/Lua_%28programming_language%29

The Web Links listed here may not be valid in the future.

# Standards Report

## Lois Goldthwaite brings news from the recent WG21 meeting in Toronto.

The international C++ committee, WG21, met in Toronto for a week in July to work on the draft of the revised standard known so far as C++0x. This was an extra meeting in addition to the two previously scheduled for 2007, so for the first time in a number of years there was no corresponding C committee meeting in an adjacent week.

We did, however, have a full-day meeting of the Posix/C++ Study Group before WG21 got underway. This body, meeting under the auspices of the IEEE, is looking at the feasibility of drafting a formal binding between C++ and the Posix operating system. The Posix standard mandates system calls expressed in C, but there are other documents specifying how Fortran and Ada code can access system services. With WG21 planning to add language and library support to C++ for multithreading and error diagnostic messages, and perhaps other system-related functionality, no one wants to see two incompatible standards competing for the same programming community.

There is strong agreement on objectives, although some disagreement on the technical details, particularly over the subject of cancelling a running thread. Under the latest C++ proposal, cancellation only happens if the cancelled thread cooperates; in Posix a thread has no option when it is told to 'die die die!' On the other hand, there is still hot controversy in WG21 over the mechanisms for thread cancellation, so nothing has been decided yet.

The latest version of the Posix specification can be found at www.open-std.org/jtc1/sc22/open/n4217.pdf. The C++ library proposal discussed in Toronto is http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2320.html, although there will likely be a revised proposal available by the time you read this.

Members of the UK C++ and Posix standards panels will be participating in developing the 'Posix++' binding; if you are interested in taking part, please write to standards@accu.org for more information.

During the rest of the week, the C++ committee worked flat out to meet their objective of producing a draft standard suitable for balloting by the end of the next WG21 meeting early in October. A number of proposals were voted into the working paper – these are a few of them:

- A new keyword, **`decltype`**, which can query the type of an expression. This is handy in writing generic functions.
- A reworking of the definition of POD, which stands for 'Plain Old Data', a data structure which has the same layout and behaviour as it would if defined in the (plain old?) C language. POD types are useful because they can be operated on by both C and C++ functions, but to achieve C layout compatibility C++ has imposed serious restrictions on their behaviour (for example, if a class or struct has any user-defined constructor it does not qualify as a POD). The new rules address layout and behaviour separately, so there need be fewer compromises with good design in order to achieve interoperability with C.
- The ability to 'delete' a function signature, so that invoking it will result in a compiler error. Historically, if you wanted to prevent instances of some class from being copied, you would have to declare the signature of a copy constructor for the class, mark it as private access, and not provide an implementation. This would cause either a compiler or linker error if the invalid copy constructor was invoked. Not only was this non-intuitive and hard to teach, it disqualified the class from being usable in circumstances where a POD was required. In C++0x, the programmer's intent can be clearly stated:

  ```
  X(X const &) = delete;
  ```

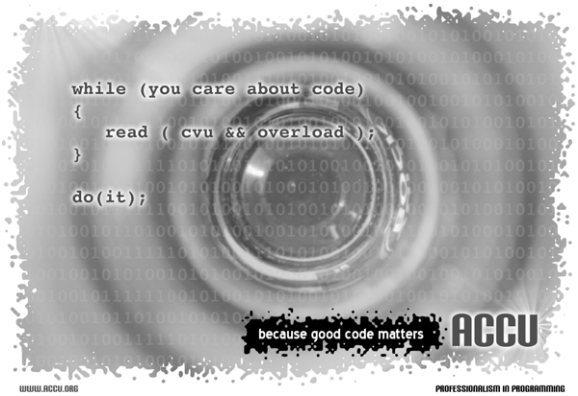
  and any attempt to copy an instance is an error, hopefully accompanied by a clear message as to why. Other member functions, and even stand-alone functions, can also be marked as deleted and uncallable.

Another bit of new syntax explicitly directs the compiler to create a special member function – the primary use case is to reinstate a trivial default constructor which has been suppressed by declaration of another constructor. Being able to reinstate the trivial default constructor is essential to the PODs proposal discussed previously. But this syntax could be used simply as documentation that the compiler-generated default member functions are being used deliberately.

- A library performance enhancement enables objects to be constructed directly into position in one of the standard containers, whereas C++98 requires existing objects to be copied into containers. Not only does this help runtime speed, it makes it possible to have collections of types which cannot be copied at all.
- The library group is also adding the capability to catch and store an exception for later examination and rethrow, perhaps even in a different thread.

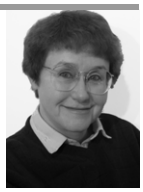
Other things that were discussed but not have yet been adopted include an extensible syntax for defining compile-time literal values of user-defined types, lambda expressions, and a syntax for attaching attributes, such as alignment specifications (another new feature), to objects.

Despite the amount of work accomplished in Toronto, even more remains to be done by the end of the next meeting, and serious thought is being given to extending the schedule rather than risking a sloppy job through hurrying to meet a deadline. If this happens, C++09 would refer to a revised standard which completed its final balloting in that year, but publication would probably not happen until 2010. The decision on whether to ship or slip will be made in Kona in October. ■




**LOIS GOLDTHWAITE**

Lois has been a professional programmer for over 20 years. She is convenor of the C++ and Posix standards panels at BSI. One of her hobbies is representing the UK at international standards meetings! Lois can be contacted at standards@accu.org.uk



# Regional Meetings

A round-up of the latest ACCU regional events.

## ACCU South Coast

**Report from Peter Hammond (pdhammond@waitrose.com)**

The ACCU south coast group has started holding meetings on the fourth Thursday of every month. Gail Ollis kindly volunteered to set things off in May with a reprise of her 'Advocating Agility' presentation from the conference, this time without the voice troubles that hampered her in Oxford. The Winnie the Pooh analogy clearly struck a chord with the audience, and there was lively discussion during the talk and afterwards over a glass in the social club bar.

At June's meeting, Mark Easterbrook gave a talk on the short message service on GSM networks, with the snappy subtitle 'hw txt wks'. This was a very interesting talk, which shed some light on how the mobile networks extract money from us, and what happens to those texts that never arrive.

The next two meetings will be on the 26th July and 23rd August, also at the East Christchurch sports and social club. Details of speakers are to be confirmed, look out for announcements on the accu-general and accu-southcoast mailing lists.

## ACCU London

**Report from Allan Kelly (allan@allankelly.net)**

The ACCU London chapter was back in July having missed a month in June. Aviv Handler from Co-herence (http://www.co-herence.com/) talked about the role and importance of the Product Manager. This was not a technical talk but it was highly relevant to software developers.

For those who don't know, a Product Manager undertakes inbound marketing, that is, they go out and talk to customers and potential customers about what problems they have. They add to this knowledge of technology and company intentions and, as if by magic, product the market requirements for a software product. Some Product Managers also perform outbound marketing – that is they arrange the marketing of the product that is products, although strictly speaking this is Product Marketing.

Again this month we used a new location for the meeting. This time 7 City Learning (http://www.7city.com) not far from Liverpool Street station. We have the ACCU publicity officer David Carter-Hichin and Paul Shaw of 7 City to thank for this and might well use the location again.

After the meeting we returned to a close-by pub. Unfortunately half the City seemed to have been involved in a fun-run and had booked all the pubs for receptions so we ended up in Wagamamas.

ACCU London will skip a month in August because so many people will be on holidays. We intend to be back in September and are in the process of arranging speakers. If you have any ideas for a speaker please let me know.

The other big change coming up for ACCU London is the loss of Paul Grenyer. Paul founded ACCU London when he moved from Norwich. Now that he is returning to Norwich, we are losing him. We in London owe a big thanks to Paul.

Our next meeting will be on Thursday 20 September when Pete Goodliffe – author of Code Craft – will be talking. We hope to be in the Liverpool Street area again. Watch ACCU General for the announcement or subscribe to the ACCU London mailing list (details on the website).

## ACCU Cambridge

**Report from Ric Parkin**

ACCU Cambridge had a meeting on the 5th of July, and if I say so myself, was a great success!

Approximately 25–30 people turned up to DisplayLink's hilltop offices. The rain made the famed Cambridge Rooftop View somewhat blurred, but an excellent hour talk from Pete Goodliffe on the different sort of 'Code Monkeys' entertained the masses. The audience participation part, deciding on designing a space probe in three minutes and 'What Monkey are You', added some initially nervous but ultimately fun intermissions.

The general feeling at the end was that we'll do another next month – the first Thursday of August looks likely, and DisplayLink should be okay to host again – and Ric Parkin has been volunteered to do an update of his ACCU talk on 'Semantic Programming'.

Thanks all who came – your enthusiasm IS important.

# Code Critique Competition 45

Set and collated by Roger Orr.

Please note that participation in this competition is open to all members, whether novice or expert. 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.

Listing 1

```
// Logger.h
#include <string>
#include <fstream>

class Logger
{
public:
   static Logger * instance(
     std::string dest = "logfile.txt" );
   ~Logger();
   void write( std::string );
private:
   static Logger * theLogger;
   std::ofstream f;
};

// Logger.cpp
#include "Logger.h"

Logger * Logger::theLogger;

Logger * Logger::instance( std::string dest )
{
   if ( ! theLogger )
      theLogger = new Logger;
   theLogger->f.open( dest.data() );
   theLogger->f.clear(); // << Help - why??
   return theLogger;
}

Logger::~Logger()
{
   if ( this == theLogger )
      theLogger = 0;
   f.close();
}

void Logger::write( std::string line )
{
   f << line << std::endl;
}

// Example.cpp
#include "Logger.cpp"

int main()
{
   Logger::instance( "example.log" )->
     write( "Starting main" );
   //
   Logger::instance()->write( "Doing stuff" );
   //
   Logger::instance()->write( "Ending main" );
   delete Logger::instance();
}
```

## Last issue's code

> I have built a simple singleton for logging and it seems to work, but I had to add a call to **`clear()`** the file stream after **`open()`** to get it to work properly. Does anyone know why this call is needed – I think my compiler's standard library has a bug?

Please answer this question, but don't stop there…the code is in Listing 1.

## Critiques

### From Nevin :-] Liber <nevin@eviloverlord.com>

#### The bugs

Let us look at

```
Logger * Logger::instance( std::string dest )
{
  //...
  theLogger->f.open( dest.data() );
  theLogger->f.clear(); // << Help - why??
  //...
}
```

The first bug is extracting the C style string from **`dest`**; **`dest.c_str()`**, not **`dest.data()`** should be used to do so, as only the former is guaranteed to '\0'-terminate what it returns.

Every time **`Logger::instance(...)`** is called, it attempts to open a file. The first call succeeds. The second and subsequent calls fail, because **`f`** already refers to an open file. [Side note: it doesn't matter whether or not **`dest`** is the same or different between calls, as it is in this code; the open still fails.] Because it fails, it performs a **`setstate(failbit)`** on the stream, and all subsequent writes are blocked until that bit is cleared. This is why the author needs **`theLogger->f.clear()`**.

#### Other issues

In **`main()`**, there is the line:

```
delete Logger::instance();
```

While I do applaud the author for cleaning up his resources, this is not the way to do it. Besides the encapsulation provided by Logger being broken (after all, why should **`main()`** know or care that the singleton is allocated on the heap), Logger does not clean up its internal state. Notably, **`Logger::theLogger`** now points to a non-existent object; subsequent calls to **`Logger::instance(...)`** could result in a crash.

The way it is currently implemented, there is no runtime polymorphic choice to make on what logger to instantiate, so it need not be allocated in the heap. We could do what is done in the Meyer Singleton (there isn't just a single Singleton pattern out there, all with different tradeoffs) and use a static variable inside a function, which is initialized the first time the block containing the variable is entered.

**ROGER ORR**

Roger has been programming for 20 years, most recently in C++ and Java for various investment banks in Canary Wharf. He joined ACCU in 1999 and the BSI C++ panel in 2002.
He may be contacted at rogero@howzatt.demon.co.uk

## Is a singleton really necessary?

One question the author needs to ask: is a singleton really necessary? Singletons tend to complicate the design, and since they are essentially global variables in disguise, they suffer most of the drawbacks of globals (tighter coupling, harder to test, harder to extend, less flexible, etc.). Would a global variable have been sufficient? Better yet, is there a way to pass an instantiated logger to the folks that need it? That being said, for the purpose of this critique, I am going to assume that the author has sound reasons for wanting a singleton, and will move on to explore implementing a better one.

In a real application, `Logger::instance()` (without a parameter) could be called a lot. The way it is implemented, each time it is called it creates and destroys a temporary `std::string` (sometimes involving a heap operation, depending on the implementation of `std::string`) of its default parameter. Also, `Logger::write( std::string line )` makes an unnecessary copy of line. Some people might consider this stuff to be premature optimisation; I consider it to be avoiding premature pessimization, as it doesn't take much engineering effort to improve it.

So what would make a better singleton in this case? I will apply the Principle of Separation of Concerns: classes which manage resources should be distinct from the resources themselves. Refactoring the code so that the logger itself is a distinct class from that which enforces its uniqueness, I come up with this design (class `LoggerSingleton` replacing the author's class `Logger`):

```
class LoggerSingleton
{
  class LoggerImpl
  {
    void operator delete(void*);
    std::ofstream f;

  public:
    explicit LoggerImpl(char const* dest)
    : f(dest) {}
    void write(std::string const& line)
    { f << line << std::endl; }
    void write(char const* line)
    { f << line << std::endl; }
  };

public:
  static LoggerImpl* instance(char const* dest
    = "logfile.txt")
  { static LoggerImpl logger(dest);
    return &logger; }

  static LoggerImpl* instance(std::string
    const& dest)
  { return instance(dest.c_str()); }

  LoggerImpl* operator->() const
  { return instance(); }
  LoggerImpl& operator*() const
  { return *instance(); }
};
```

Notes on LoggerSingleton:

- `LoggerImpl`: I have chosen to make `LoggerImpl` a private class of `LoggerSingleton` to limit access to it: only `LoggerSingleton` can create it, hold on to a pointer to it, etc., while all of its public member functions (remember, constructors are not considered member functions) can be called by anyone who can get a pointer/reference to the object.
- `LoggerImpl::operator delete(void*)`: this is private and unimplemented, in order to change those uses where client code does a delete `LoggerSingleton::instance()` into a compile time error. Technically, this violates Separation of Concerns; it is only in there because the original code calls delete on the instance.
- `LoggerImpl::LoggerImpl(char const*)`: Takes the destination file as a C style string, and initializes the ofstream using a member initializer.
- `LoggerImpl::write(...)`: To avoid a pessimization, there are two versions of write: one which takes a C style string, and one which takes a `std::string` by const reference.
- `instance(char const*)`: This implements the Meyers Singleton (using a static variable inside a function, as alluded to earlier. This singleton pattern was first introduced by Scott Meyers. It also has the side benefit of not requiring a .cpp file if all the code is inlined in the class). Also, to avoid a pessimization, instance takes a C style string as a default parameter. That way, subsequent calls only pay the penalty of passing a pointer (which could get optimized out) instead of creating a temporary `std::string`.
- `instance(std::string const&)`: This function is for interface compatibility with the old Logger class. The `std::string` is passed by const reference instead of by value to avoid making an unnecessary copy. It calls `LoggerSingleton::instance(char const*)` because that is the function which owns the actual object.
- `operator->() const& LoggerSingleton::operator*() const`: While not strictly necessary, these are convenience functions. Accessing the object via

  ```
  Singleton::instance()->write("blah")
  ```

  is pretty clunky. With these functions, a variable declaration such as

  ```
  LoggerSingleton aLogger;
    //...
  aLogger->write("blah");
  ```

  could be used to access the object more convienently. This declaration could be local, global, or a static in a header; they all give access to the same singleton object through a pointer like interface.

### From Peter Hammond <Peter.Hammond@baesystems.com>

The simple answer to the question "why is the call to `clear()` needed" is that the code is attempting to open an `fstream` object that is already open, which is illegal. It is stated in section 27.8.1.3-2 of the standard [1], but neither MSDN [2] or Josuttis [3] mention it, so you could be forgiven for not knowing.

The more interesting question is how did you come to be reopening the file in the first place? If the `open()` had succeeded, then all but the first write to the log would have gone to the file `logfile.txt`, which is probably not what was intended. The problem is that the design lacks both symmetry and encapsulation; encapsulation because the logic is spread out over several places, and symmetry because the opening and closing of the file are not logically paired. It should be unusual in well-written C++ to use `fstream::open()` or `fstream::close()` explicitly, relying instead on the constructor and destructor to manage the underlying file for you.

There is no user-defined constructor for the `Logger` class, which ought to raise an alarm in a class that is managing a resource – in this case, the file. The destructor is defined, but has a rather unusual test for self in it. This is particularly strange since the point of singleton is to ensure that only a single instance of the class gets constructed: `this == theLogger` ought to be an assertable pre-condition. Also, it is a general characteristic of implementations of the Singleton pattern that they do not get deleted,

but simply leak at the end of the application, since there is nothing to clean them up.

The optional presence of the string argument on the instance method shows up another drawback of the singleton pattern, namely that if non-default initialisation is required, something has to be responsible for initialising it, which brings in a system-level dependency.

A better design would be to let the `Logger` class's constructor and destructor to manage opening and closing the file, and give the instance method the sole responsibility of managing that instance. An implementation using these techniques is given in listing 1.

### References


[1] *The C++ Standard* (BS ISO/IEC 14882:2003), Wiley, Chichester, UK, 2003.

[2] Microsoft Developer Network 2003 edition.

[3] N Josuttis: *The C++ Standard Library: A Tutorial and Reference*, Addison-Wesley, Reading, MA, 1999


Listing 2

```
// Logger.h
#include <string>
#include <fstream>

class Logger
{
public:
  static Logger * instance(
    std::string dest = "logfile.txt" );
  void write( std::string );
private:
  ~Logger();
  Logger(std::string dest);
  static Logger * theLogger;
  std::ofstream f;
};

// Logger.cpp
#include "Logger.h"

Logger * Logger::theLogger;

Logger * Logger::instance( std::string dest ) {
  if ( ! theLogger )
    theLogger = new Logger (dest);
  return theLogger;
}

Logger::Logger (std::string dest)
  : f(dest.c_str())
{}

Logger::~Logger()
{}

void Logger::write( std::string line )
{
  f << line << std::endl;
}

// Example.cpp
#include "Logger.h"

int main()
{
  Logger::instance( "example.log" )->
    write( "Starting main" );
  //
  Logger::instance()->write( "Doing stuff" );
  //
  Logger::instance()->write( "Ending main" );
}
```

## From Stephen Love <stephenlove@south-staffs-water.co.uk>

(co-authored by Steve Love and Nigel Dickens)

"It is indeed a most unwelcome shower, Watson." My good friend Sherlock Holmes peered through the smoke of his pipe with a perceptible – and mischievous – glint to his eyes.

"I am aware, Holmes, that the number of your clients has been, well, let us describe it as having been not overwhelming, of late," I remarked perhaps a little unkindly, "and I fear that this observation is unworthy of you. I realise that you will have seen the telegram from Asquith inviting me to the cricket match this afternoon, and that the rain will indeed cause a cancellation of the event." In truth, neither of us had been given reason to leave our rooms at 221b Baker Street for nearly two weeks. This enforced idleness, so alien to the nature of my friend, was beginning to irk us both.

Expecting a crestfallen and sulky response, I was preparing a sweetener for my acidic remark when Holmes leaped from his armchair, his eyes still gleefully ablaze. Chuckling, he re-filled his pipe from the slipper on the mantelpiece, and lit it so vigorously I was sure all the tobacco would be smoked before he could resume his seat. "My dear Watson, I was in fact unaware you had contemplated the weather. My bemoaning of the inclement conditions was entirely selfish."

This puzzled me, for as I have already noted, no clients had visited for quite some time, and nothing in the mail had excited any interest in my companion, who rarely left Baker Street without some definite purpose. "You have an engagement, then?"

"You are luminous today, Watson, but no, not really an engagement. It seems my reputation for solving little programming problems has expanded, just as any reputation I may have for other more obviously criminal ones has appeared to contract." [Interested followers may like to look up our previous engagement of the kind in CVu 12/5, September 2000].

"A programming problem, Holmes?" I enquired, bewildered. "Surely nothing the weather has to offer could inconvenience you in that regard?"

"Quite the reverse, I'm afraid," he replied ruefully, once again seating himself before the fire, concentrating on keeping his pipe alight. "I perhaps do not practice the programming problems as diligently as I should, particularly in regard to C++ code such as this." He took a note from his jacket pocket and passed it to me. "My copies of those illustrious references by Stroustrup, Josuttis and Langer and Kreft – which are such an invaluable aid to me – are on loan to Mycroft, who occasionally finds such skills called upon by our government. I'm sure copies could be found at the library, however, but even I am unwilling to venture out in this downpour."

"Surely there must be material enough here to make a start, Holmes," I admonished him. "I have a few thoughts of my own."

"Yes, I rather thought you might!" Holmes declared, laughing. "Very well then, let's hear what you've determined already."

"Ignoring, for the moment, matters of style," I began, eliciting a short laugh from Holmes, "what immediately strikes me as wrong is the deletion of an object that doesn't get created anywhere. A little investigation shows that the creation using `new` occurs hidden away in the `instance()` method. This asymmetry looks to be an obvious source of potential bugs, as does the rather curious logic in the destructor itself. It seems to me that the file object used to write to the log gets closed once on destruction, but opened every time the log is written to."

"Excellent stuff, my dear Watson, and you've reached right to the heart of the matter, as always. And what of your matters of style, may I ask?"

Warming to my subject, I continued. "I might well ask whether the author of this code comes from a background of a different language, perhaps C#. I suggest perhaps the author was schooled in C# or something similar, and

has come to C++ with the idioms of that previously learned language in mind.

"What leads me to this observation is primarily the parameter declarations of the member functions of the Logger class, but there are other indications. When passing strings or objects of a type that may be expensive to copy, it is common for functions in C++ to accept arguments by constant reference. In C# of course, all objects are implicitly passed by reference, and so the issue does not arise. Accepting string arguments by value causes an unnecessary copy of the string contents each time the function is called, even in the case of the default arguments to the `instance()` method."

"Bravo," encouraged Holmes, "you are certainly on form this afternoon. What else do you deduce from the code?"

Now, the habits and mannerisms of my companion had become extremely familiar to me in our years of friendship, and all the more acutely due to our enforced close proximity of recent weeks; it was therefore clear to me that my exposition had aroused Holmes' own interest in the topic, and that he was eager, even impatient to apply those of his powers I have documented at length in other tales. Sensitive to the sometimes-fragile ego of my friend, I demurred. "I believe I can ascertain nothing further from this code." I passed the listing back to him, and proceeded to light my own pipe.

After a few moments, Holmes put the note on the occasional table beside his chair, and contemplated the ceiling. "You've done exceedingly well, Watson, and I fear you rather underestimate your own powers." At this, he looked briefly at me, and I confess I must have done rather well at looking modest, so gratified was his expression. He continued: "Nevertheless...nevertheless. I think there is more that can be gleaned from this short snippet."

With his habitual air of one giving a lecture (and please do not misunderstand me, as always it was a pleasure to see him so absorbed in something other than his tiny syringe during such a quiet time for his professional abilities), Holmes stood beside the fireplace, one arm on the mantel, his other hand gripping his long-stemmed pipe using it for all the world as if it were a conductor's baton.

"I believe you have hit directly upon the truth that the author of this code is more comfortable with a different language, although I suggest Java rather than C#. The lack of an initial capital for the member function names suggests so, and Java is more commonly taught as a first language, but I would not swear my life to it. In any case, it is of no importance.

"What is rather more – well perhaps sinister is too strong after all, let us say 'of concern' – is the indication that the author is more used to an environment in which memory is automatically managed, that is to say a garbage collected environment. Either C# or Java would both fit this description, along with several others. There are two indicators for this, one of which you have already identified: the manual deletion of the logger object at the end of `main()` suggests it may have been added as an afterthought. The second indicator reinforces the first, the setting of the `theLogger` member variable to zero. This is a common practise in a garbage collected environment as a strong hint to the runtime that the object is no longer referenced.

"It is interesting to note, in passing, the use of zero rather than NULL or one of its cousins. This, along with the style of header-file inclusion (note there or no `.h` suffixes for the standard library headers), the explicit stating of the `std::` namespace rather than a using directive, and correct declaration of `main()` all suggest an author who has tried hard to write this code the correct way."

"All of that is very clear, Holmes, and I believe you're quite right about author's desire to write the code correctly, but what about the direct question? Why the need to clear the stream after opening it? I still cannot fathom that." It may be that you would accuse me of disingenuousness in my earlier attempt at preserving my friend's pride, but I was genuinely puzzled upon this point.

"Well, Watson, for the answer to that question I am afraid I shall have to resort to the formal authorities. I perceive that an abatement to the rain is upon us, and I shall take the opportunity to make my visit to the library. Do you wait here, my dear chap, on the off-chance that we have a visit from another client! I shall be back within the hour."

And with that, Holmes perched his best topper on his head, wrapped his cloak about his shoulders and, cane twirling, departed for town, leaving me with my pipe and the early evening papers.

True to his word, Holmes returned just an hour later, doffing his hat to Mrs Hudson on her way out. "Ah!" I exclaimed, "Your timing is as impeccable as ever. Mrs Hudson has been kind enough to bring tea and I believe the buttered scones shall be all gone if you're not quick about it!"

Holmes hung his cape, hat and cane on the stand by the door, and rubbed his hands together, clearly eager for the repast. "I didn't quite miss all of the rain, Watson, as you see, so I shall enjoy some tea and cake before I make my report on what was, if I may be just a little presumptuous, a very enlightening afternoon."

With the last of the tea poured, and the plate of scones empty, we both lit cigarettes and retired to the arm chairs by the fireplace. Holmes, in typical style, enjoyed the smoke for a minute or two, deliberately making a show of his efforts.

"It's been quite some time since I looked into C++, as you know Watson, and it really has been a refreshing experience to once again consult the honest authorities in this domain."

"It's a stroke of good luck you weren't caught 'consulting' someone else by Lestrade," I remarked with a grin.

"Hah! Yes, you're quite right! A pretty picture he would have painted! Nevertheless, it is quite as wrong to ignore one's shortcomings as to be modest of one's strengths, so if I admit to referring to Stroustrup's *The C++ Programming Language*, Langer & Kreft's *Standard C++ IOStreams and Locales* and Josuttis' *The C++ Standard Library*, I feel no worse for it!

"But I digress. I think we have an answer for our friend, but it may not be the one he or she was expecting. The root cause of the problem in the code, I believe, is the use of a Singleton class to manage access to the log. I have noticed this Singleton's hand in a number of problems of this kind, Watson, and it is indeed responsible for much pain and distress among programmers in all languages.

"Indeed I have been engaged during this intolerable lull in preparing a small monograph on the matter as something so malignant should not be permitted to persist unmolested.

"The primary crime committed by the Singleton is that it is a global variable, in disguise. This in itself causes difficulties with exception- and thread-safety, and additionally can cause code to become convoluted just to make it work! Observe the way our friend must manually delete the internal content of the Logger class. In a similar vein, notice the Logger class containing an instance of itself, surely a great source of errors. I will pick on one – in the destructor, checking `this` against the static member variable is a trick which I think highlights what I believe to be confusion brought on, in the end, by the use of a Singleton. The `instance()` method of `Logger` creates a new Logger object, which is the one and only instance which can have its destructor run. In that destructor, it checks needlessly to see if it is the instance, and if it is – as it always will be – effectively sets its own `this` pointer, aliased as the static `theLogger` variable, to zero.

"The alternative is to make the logger class a real object, with a normal instance, and just through this most of our difficulties vanish." At this Holmes produced a pen and began to scribble.

"Consider this alternative," he said, passing his notes to me. (Listing 3)

"Note that, despite the fact it is not a Singleton, there is still only one instance of the log object. Also note there is no need for a destructor, and that the file will open and close to match the lifetime of the log object which owns it. It is for this reason I have not taken the extra step of only depending on a `std::ostream` reference, and having that passed in to the Logger itself.

Listing 3

```
#include <string>
#include <fstream>

class Logger
{
public:
  Logger( const std::string & dest =
    "logfile.txt" );
  void write( const std::string & );
private:
  std::ofstream f;
};

Logger::Logger( const std::string & dest )
: f( dest.c_str() )
{}
void Logger::write( const std::string & line )
{
  f << line << std::endl;
}

int main()
{
  Logger log;
  log.write( "Starting main" );
  //
  log.write( "Doing stuff" );
  //
  log.write( "Ending main" );
}
```

"The primary difference this would make to real code is that instead of other functions and classes accessing the `Logger::instance()` directly, they would have to be told about the existence of the log by having a reference to it *passed* in to them, say via a constructor."

"But what would stop those classes or functions from creating new instances of the log, thereby breaking the rule that the Singleton pattern enforces, that there can only ever be one instance?" I asked.

"Ah, well that would require a change in the design, but it would certainly not be for the worse. Imagine that `Logger` is a pure abstract class, with the `write()` method pure virtual, implemented in a separate class to which only `main()` has access. The benefits of this are two-fold: clients' functions and classes who require access to the log depend only on the log itself, and have no need to know about file streams and such, and they can work with any instance of the log which satisfies the `Logger` interface – which may include 'do-nothing' or 'mock' instances for testing, or logs which communicate over a network. The possibilities become endless instead of, well, singular!"

I was gratified that my friend's attention had been drawn long enough from chemical stimulants to produce this result, but there was still one remaining question. "There is one point, Holmes, upon which you could satisfy my curiosity," I remarked in as off-hand a way as I could manage.

"Well, Watson, what is it, then?"

"The author's original question – why the need for the call to `clear()` each time the `instance()` method was called?"

"Ah yes!" he cried, "I had almost forgotten, since I removed the need for it anyway." I was hard-pressed to suppress a smile at this. "Attempts to open an already open file stream will fail, in much the same way as for a door. Calls to `open()` must be matched exactly with calls to `close()`, which is why the constructor of a file stream taking a c-string file name automatically open the file, and why the destructor closes it. Therefore, without the call to `clear()`, only the first call to `instance()` would successfully open the file, and I suspect the original program's output – before the extra call to `clear()` was added to make the output match the expectation – was just 'Starting main'."

"Regarding monographs," I ventured "have you made any progress on your dissertation regarding paramaterisation from above?" This was rash of me I know but in the glow of success I dared to bring up this unfinished business. Holmes' dark eyes flashed, then with barely concealed relief he turned to face the door to our chambers.

"Now this has been a most diverting and illuminating problem, Watson, but I think – unless you have further questions? – that we must turn our minds to more serious matters, for I do perceive the heavy footstep of Mr Gregson of the Yard on our landing, and I feel certain he has come to consult on matters far different from those we've been discussing."

## Commentary

The original code has a number of problems in addition to the one presented by the programmer; and it is hard to decide which is the biggest such problem.

The original issue was a question about the need for `clear()`, which is a good example of a bad fix. The call was needed to recover from the error condition caused by opening the file on every call to `instance()` but the additional line simply hides the original fault. There are various ways of helping empower the programmer to resolve this sort of issue for themselves; for example getting them to do a walk through what the code actually does when executed.

There are various implementation issues

- the code makes explicit use of `open()` and `close()` on the stream which, as Peter says, is 'unusual' in well-written C++ as it is common to simply use on the `ctor` and `dtor`.
- passing strings by copy where a const reference is more usual
- creating a string for the no-arg call to `instance()`
- incorrect use of `data()` rather than `c_str()`. This is quite a common bug, which is not helped by the existence of some other string classes where this is the right method to call!

However, I feel the design issues are bigger in this case, such as:

- writing your own logger is usually less effective than using one, such as log4cxx, which is typically more configurable and has better performance
- is a singleton the right pattern for the logger? I had hoped to provoke Kevlin Henney to write an entry on 'parameterise from above' – perhaps another time! Even when the answer to the question is 'yes'it is still often better to split the class up, as Nevin did.
- there is no error handling (eg if the log file won't open logging fails silently)
- the code makes no attempt to be threadsafe, should it?
- there is a problem with a 'resurrecting' logger if something calls the `instance` method after the initial logger has been deleted

## The Winner of CC 46

All three entries did a good job of solving the issue presented by the programmer and providing solutions with varying degrees of change to the original code.

I am awarding the prize for the best critique to Steve & Nigel for their entry; it not only covered a lot of ground technically but was entertaining at the same time.

## Code Critique 47

(Submissions to scc@accu.org by Sep 1st)

"I have this small program here and am wondering why, when compiled with full optimisation it does not produce any output at all? It works fine without optimisation."

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int tab[13][20]={
   {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   {0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0},
   {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0},
   {0,0,0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,1,0,0},
   {0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1},
   {1,0,0,1,0,0,0,1,0,1,0,0,0,1,1,0,0,1,0,0},
   {0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,0,0,1},
   {0,0,0,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0},
   {1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,2,0},
   {0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,0,0,1},
   {0,0,0,0,1,1,0,1,1,0,0,0,0,0,1,0,1,0,0,0},
   {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0}};

int tabx[13][20];
int sumy[20];

int licz(char *konfig)
{
  int i,n,l,el,ile=0;
  char c;

  memset(&sumy,0,sizeof(sumy));
  memset(&tabx,0,sizeof(tabx));

  for (i=0; i<=12; i++)
   for (n=0; n<=19; n++) {
    c=konfig[i];
    l=atoi(&c);
    el=l*tab[i][n];
    if (el>0) sumy[n]=sumy[n]+el;
    tabx[i][n]=el; }

  for (n=0; n<=19; n++)
     if (sumy[n]==3) ile++;
  return(ile);
}

int ile1(char *s)
{
  int i,n=0;
  for (i=0;i<=12;i++) if (s[i]=='1') n++;
  return(n);
}
```

```
void main()
{
  char komb[13];
  int a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad;
  int xx,w,ile;
for (xx=0;xx<=4;xx++)
  for (a1=0; a1<=1; a1++)
   for (a2=0; a2<=1; a2++)
    for (a3=0; a3<=1; a3++)
     for (a4=0; a4<=1; a4++)
      for (a5=0; a5<=1; a5++)
       for (a6=0; a6<=1; a6++)
        for (a7=0; a7<=1; a7++)
         for (a8=0; a8<=1; a8++)
          for (a9=0; a9<=1; a9++)
           for (aa=0; aa<=1; aa++)
            for (ab=0; ab<=1; ab++)
             for (ac=0; ac<=1; ac++)
              for (ad=0; ad<=1; ad++)
               {
sprintf(komb,"%d%d%d%d%d%d%d%d%d%d%d%d%d",
a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad);
                 w=licz(komb);
                 ile=ile1(komb);
                 if ((w==1) && (ile<=3))
                 {
                  printf("%s\t%d\t%d\n",
                  komb,w,ile);
                 }
               }
```

Please try to help the programmer find the answer to this question (and future ones).

You can also get the current problem from the accu-general mail list (the 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. ■

Prizes provided by Blackwells Bookshops and Addison-Wesley

# 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.

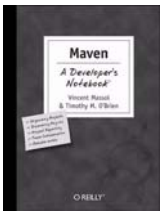
After you've made your choice, email me and if the book checks out on my database, you can have it. I will instruct you from there. Remember though, if the book review is such a stinker as to be awarded the most un-glamourous "not recommended" rating, you are entitled to another book completely free.

## Software Development

### Maven: A developer's notebook

**by Vincent Massol & Timothy O'Brien, published by O'Reilly, 191 pages. ISBN: 0-596-00750-7**

**Review by Michel Greve**




This book is a soft cover edition and contains no CD. This book is a hand on guide for Maven 1.0.

The book has a nice layout and reads well. Every paragraph has a number of sections.

- Task to perform
- How do I do that?
- What just happened?
- What about?

A paragraph describes a task and how to perform that task. Every step, including the output, is shown, so you know exactly what to do and what to expect. Besides the actual steps, they give a brief explanation what you just did and sometimes they give some alternatives to the chosen tools. E.g. SVN is explained, but they also give a brief explanation how to do this with CVS.

The book is organized in six sections:

- Maven Jump-Start
- Customizing Maven
- Multiproject Maven
- Project Reporting and Publishing
- Team collaboration with Maven
- Writing Maven Plug-ins

It starts with setting up a Maven environment. This part is quite extensive. It takes you by the hand installing Maven and how to start with a project. How to use Maven behind a proxy and it helps you use Maven on a project. Eclipse, source control and project documentation (and more) are all clearly explained. In the next chapter they dive deeper in the configuration of your Maven project. By developing self defined goals and by tweaking property files for the plug-ins you get the desired result. In the third chapter you learn how to create multiple artifacts with a Maven installation. This is of course without duplicating anything. The fourth chapter is about reporting. They talk you through multiple reports where one report task already showed you how to do it. So there are some wasted pages here. The chapter about team collaboration shows you how to use Maven with multiple users, so that every user has the same environment. Personally I don't think this is really important, because some pages ahead they are talking about CruiseControl. Unfortunately, I couldn't get CruiseControl working properly. I don't think that the book is wrong. I did execute the task on a machine with a newer configuration (the book was printed in June 2005). The last chapter is about writing your own plug-ins with jelly, a scripting language. Jelly is replaced with Java in Maven 2.0. The appendix shows the plug-ins used in the book and how to install them. Although I appreciate that they list all the plug-ins, the rest of the appendix isn't worth the paper.

Unfortunately the book is dated. The version for Maven used in the book doesn't work anymore, so I worked with version 1.1 beta 3. Also the site generation and CruiseControl didn't work properly although I downloaded the source code from the website.

The book itself is very good. I like the clear explanation of the tasks performed and the task driven approach. The book gets you up and running in no time. It's a pity that they don't have a version of all the tools (and a repository for Maven) on their website. The book would have stood the test of time. If they updated the book I would buy the book, but this version is not recommended.

### Why Software Sucks... and what you can do about it

**by David Platt, published by Addison Wesley, 242 pages, ISBN: 0-321-46675-6**

**Review by James Roberts**


This book attempts to be a light-hearted summary of faults of computer software – with a certain amount of outraged rant regarding examples of poor design (i.e. even when the software works as it was supposed to, it is harder to use than it really ought to be). The prose is very easy to read throughout, with some humour (not laugh out loud, as the blurb on the back claims, but not bad).

The author has one basic message, which is reiterated with various examples throughout the book: 'know your user, for he is not you'. This works well as a leitmotif – the constant repetition being a useful underlining of the message.

An interesting (possibly intentional) feature of the book was that I felt that the early chapters applied to me directly. For example asking me to think about the way user interfaces are designed, with a good summary of security and privacy issues from a real-life, 'what will people really do' perspective. In particular I thought that the security chapter was excellent, as the author took a pragmatic view of balancing security versus usability rather than insisting on unrealistic (but notionally very tight) levels of security at all times.

Later chapters, describing programmers in rather extreme and stereotypical ways, left me a bit cold. Nothing wrong with stereotyping programmers as such – but I couldn't see how these chapters were aimed at improving

## Bookshops

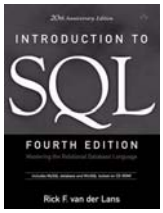
The following bookshops actively support ACCU (offering a post free service to UK members – if you ever have a problem with this, please let me know – I can only act on problems that you tell me about). We hope that you will give preference to them. If a bookshop in your area is willing to display ACCU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

- **Computer Manuals** (0121 706 6000)
  www.computer-manuals.co.uk
- **Holborn Books Ltd** (020 7831 0022)
  www.holbornbooks.co.uk
- **Blackwell's Bookshop**, Oxford (01865 792792)
  blackwells.extra@blackwell.co.uk

software. I would certainly have preferred to have had more chapters along the lines of the first few – its not as if there is a shortage of examples of poor software design in the world.

In summary, generally a good book. However, it is let down by lack of focus in a couple of the later chapters. Perhaps these are subjects that Mr Platt is more interested in than me. Know your user – for he is not you. Indeed.

## Introduction to SQL




**by Rick F.van der Lans, published by Addison Wesley, 1025 pages + CD, ISBN: 0-321-30596-5**

**Review by James RobertsPages**

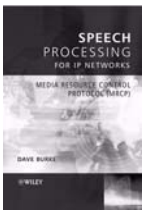
This book is designed as an tutorial for SQL. It does a good job of explaining the functionality of SQL in a step by step manner. Each of the points explained by the book is liberally illustrated with examples and student exercises. In particular the book does very well in highlighting the behaviour of null columns – explicitly highlighting how nulls are handled in its explanation of individual SQL features. Query syntax is formally described as it is introduced (BNF), which is also echoed in the appendix.

As the introduction highlights, there is a wide selection of SQL implementations, each of which takes a different compromise with respect to the standard. The author picks MySQL as a reasonably full and available example SQL engine – and does a good job of highlighting areas where other implementations are substantially different. The book comes with a CD containing a version of MySql and the WinSQL user interface tool. In my opinion these are a good pair of tools to use for learning the basics of SQL.

On the downside, I found the book's style difficult to skim read – many of the points are introduced by example, and are not highlighted typographically. Although this is quite reasonable for the tutorial function of the book, it would make it less useful as a reference book after the first reading. Also, I found that on some of the non-interactive elements (transaction control and isolation levels for example) the book was a little hazy and could have been a little better explained.

However, as a basic introduction to SQL this is a pretty good book – although as it comes in at over 1000 pages, I would probably recommend a book on weightlifting as a necessary precondition to its use.

## Speech Processing for IP Networks: Media Resource Control Protocol (MRCP)




**by Dave Burke, published by John Wiley & Sons (2007), 354 pages. ISBN 978-0-470-02834-6**

**Review by Ivan Uemlianin**

Highly Recommended

Publisher's website: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470028343.html

Author's website: http://www.daveburke.org/speechproc.html

Media Resource Control Protocol (MRCP) is a new IETF technology which provides a standard internet interface to speech processing resources like synthesis (i.e., text-to-speech) and recognition. This interface involves an acronym soup of other IETF and W3C standards including SIP, SDP, RTP, SSML, SRGS, NLSML, and PLS. Although MRCP is stable and in use in the wild, to date there has been no tutorial or documentary material other than the IETF specifications.

This book will come as a double relief to anyone working in the area: first that such material has arrived, and second that it is of such high quality.

The book is in five parts, progressing from background material, through the mechanics of MRCP, to an example of MRCP and VoiceXML in action.

Part I gives background on MRCP and speech processing in general. The chapter on speech processing is best regarded as interesting supplementary (i.e., optional) reading, though a few references to more solid literature are given. The chapters on MRCP provide a worthwhile historical and architectural context.

Part II explains the nature of MRCP sessions. An MRCP session is a complex entity, involving the Session Initiation Protocol (SIP) and the Session Description Protocol (SDP) to set up two separate channels: a media session running over the Real-time Transport Protocol (RTP) to carry the audio data, and a control session running over TCP to carry control messages in MRCP message format. Consequently, this section carries a lot of responsibility.

Part III covers the xml formats used in the bodies of the MRCP control messages: i.e., Speech Synthesis Markup Language (SSML), Speech Recognition Grammar Specification (SRGS), Natural Language Semantics Markup Language (NLSML), and Pronunciation Lexicon Specification (PLS). As you can imagine, this section is tedious but necessary.

Part IV describes the resources that an MRCP server can provide: synthesiser, recogniser, recorder, and (speaker) verifier.

The final Part V introduces VoiceXML, describes how VoiceXML and MRCP interact and demonstrates this interaction with a small application example.

Three so-called appendices give overviews of the deprecated MRCP version 1, HTTP and XML.

The writing is clear and direct, and the coverage is comprehensive, thorough and explicit throughout. Although the book is not explicitly split into 'tutorial' and 'reference' sections, it fulfills both uses admirably. It is authoritative and dependable.

Virtually every topic covered has an explicit example: from just a snippet of XML or an SIP message header, to fully annotated walkthroughs of SIP or MRCP client/server sessions. These examples are never overlong and are always to the point.

There are a couple of minor errors in chapter 2 on the basic principles of speech processing:

- p12 & p31: it is not possible to produce sounds that are simultaneously voiced and unvoiced (e.g., the sound associated with 's' in 'is', pronounced like a 'z'). The example given is a voiced fricative, the noise (i.e., aperiodic acoustic signal) being caused not by devoicing, but by constriction in the vocal tract.
- p14 Figure 2.3: the items labelled allophones (possible phonetic variations within a phoneme) are actually triphones (ordered sets of three phonemes).

These errors are very minor and do not obstruct understanding. As chapter 2 is probably the only dispensible part of the book I imagine few readers will even come across them.

The appendices are 'so-called' because they are not actually appended to the book: they are available only as PDFs from the author's website [2]. With the 'appendices' on HTTP and XML it's no great loss, but Appendix A on differences between MRCP versions 1 and 2 is important and should have been included in the real book.

My only real disappointment is that the code from the walkthrough examples is not available for download. These walkthroughs are effectively verbatim transcriptions of MRCP sessions and as such would be extremely useful to people developing or testing related software.

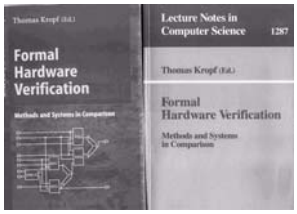
MRCP is a very new technology and this book its only substantial documentation, barring the IETF's RFCs. Apart from a couple of minor quibbles this book is the model of what a first book on a new technology should be like. I recommend it highly to anyone working on speech processing over IP, or indeed to anyone thinking of writing a book on a new technology.

### References

[1] http://www.llaisdy.com/static/tech/mrcp/index.html

[2] http://www.daveburke.org/speechproc.html

## Hardware Verification: Methods and Systems in Comparison




**by Thomas Kropf, published by Springer (1997), 348 pages. ISBN 3540634754**

**Review by Colin Paul Gloster**

This book is not intended for newcomers to hardware verification. I had been lectured for less than a semester each in VDM (for specification, and

not necessarily for hardware); Z (as for VDM); and PVS (for hardware verification) before reading this book.

In each chapter, an advocate of a rival tool applied it to a subset of the same examples from IFIP WG 10.5 as the book's other tools for the sake of comparison. Some chapters contained code for these comparisons, others contained only prose thus thwarting comparison. Some coauthors admitted that their techniques are inferior for some of these examples. The appendix documenting the examples is incomplete referring to the Internet but the editor advised me 'Finally it became so outdated that we took it from the internet'.

One example is called both 'Benchmark 17' (on Page 69) and 'Benchmark 11' (on Page 71). This is one of many spelling errors in all chapters (except for the well spelt COSPAN chapter) so people involved in verification make mistakes.

Bad identifiers abound, e.g. on page 93: 'X, Y, Z are disjoint sets of variables, viz. the input, state, and output variables respectively.' By the next page I couldn't remember which was the input; the state; nor the output.

Familiar characters are used bizarrely. Additionally inconvenience is caused due to the unusability of most of these characters in source code. E.g. on page 13: a capital T sans serif 'represents an inconsistent or both-true-and-false truth value'. In fairness, ASCII notations outside of this book also have unconventional meanings (e.g. * for multiplication instead of convolution).

On page 144 a long, error-prone piece of code (`<< v[2], v[3], ..., v[n]:= v[1], v[2], ..., v[n-1] >>`) is dismissed partially because 'syntactically it is clumsy and not easy to generalize to an arbitrary value of n' but its replacement is sorely lacking a loop:

```
<< v[2]:= v[1] >> +
<< v[3]:= v[2] >> +
   ...
<< v[n]:= v[n-1] >> +
```

After reading this book, I did not feel that I would choose a tool I did not already know.

# Java

## JBoss 4.0 The Official Guide

**by Marc Fleury, Scott Stark, Norman Richards, published by published by Sams Publishing, 2005, 634 pages. ISBN 0-672-32648-5**

**Review by Andrew Marlow**


Not recommended.

The chapters are not grouped in sections and they have a standalone feel (which I comment on later in this review), so to mention what the book covers one has to mention all the chapters. These are, in order: installing and building the JBoss server, the JMX micro-kernel, naming, transactions, EJBs, messaging, connectors and security on JBoss, web applications, MBean services miscellany, the CMP engine, web services, hibernate and Aspect Oriented programming.

It certainly seems from the table of contents that the book will be a comprehensive reference work. However, appearances can be deceptive.

This book has a misleading title. It is not a guide to JBoss. It assumes a lot of knowledge about J2EE and J2EE servers and dives right in to how to configure JBoss without explaining much about what JBoss is or how you would use it to develop, deploy and administer a J2EE application that uses JBoss.

A high level of understanding of J2EE and JBoss are prerequisites for reading this book. This alone is insufficient to give the book a 'not recommended' rating. The other reasons are poor idea and factual presentation, excessive and unhelpful XML listings, poor structure and insufficient introductory and explanatory material.

First, even if the book is not a JBoss guide I would expect some introductory material that explains what JBoss as a whole is for and what the main purpose of each component is. This was lacking.

The presentation is poor throughout. The vast bulk of the book is page after page of XML configuration file fragments. This gives the impression that it has been hastily thrown together with XML given at every opportunity in order to pad the book. The diagrams that show certain nodes for a particular part of an XML configuration also seem to be padding and are very poorly produced. The print is sometimes tiny, far too small to read. This also applies to many of the UML diagrams. Some of the diagrams look like they were done in colour originally and much larger and then later a decision was made to photo-reduce them and produce them in black and white. One part of the text even asks the reader to refer to the 'blue part', but all the diagrams are done using a dark grey shading. This adds to the impression of low quality.

The explanations are very superficial and there is hardly anything that helps to place the facts in context. The lack of detail gives me the impression that the main focus is to cover the XML configuration parameters. In several places where one would hope to find some detail or background one is referred to an external specification (e.g for JBossMQ, JCA (connectors), JSP 2.0 (Tomcat), and Hibernate documentation).

The writing style also seems to be quite dry.

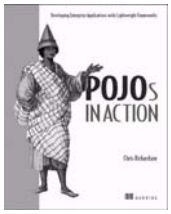
Finally, the book seems to lack structure. The table of contents seems to divide things up well but the chapters actually turn out to be quite distinct. There is nothing to link them together or show the reader where things are leading. Each chapter also lacks structure as it typically has a one or two paragraph introduction to the topic (insufficient) followed directly by page after page of poorly produced XML with insufficient supporting explanations. This lack of structure renders the book unsuitable as a reference tool. The standalone approach to each chapter means that the book as a very uneven flow. The early chapters are gruelling, with details on the JBoss micro-kernel architecture being covered in the second chapter. Other later chapters, such as the ones on security and messaging are a little lighter. The one I found the easiest and most informative was the last chapter on Aspect Oriented Programming (AOP).

The book is available on-line (http://docs.jboss.org/jbossas/jboss4guide). Some people feel, as I do, that is can be worth buying a book that exists on-line, so that one can annotate the hard-copy. However, in this case it is probably not worth it. This is because there is so much tutorial information missing and the information that is purely related to which XML configuration parameters there are for a given component can just as easily be found using the online documentation. Some chapters are extremely light, and the reader is referred to the main documentation for the component (e.g the Hibernate chapter is only five pages long and refers you to the book *Hibernate in Action* by Manning). This is another reason to prefer the online documentation.

## POJOs in Action

**by Chris Richardson, published by Manning 2007, ISBN 1-932394-58-3**

**Review by Christer Lofving**




Since the (at least to some people in the J2EE world) unexpected decline and fall of EJB 2.0, I have had a feeling of a shortage. Not a shortage of alternate technologies, but of appropriate information about them. I am thinking about 'hardcopy' alternatives to the scattered knowledge (mainly in Web forums) about stuff like Hibernate, Spring, EJB 3.0 and Ibatis. OK there are some good titles out there, about Spring and Hibernate for example. But so far I haven't noticed any effort to give a more complete overview of the different J2EE lightweight APIs, nowadays growing up like dandelions in the ashes after the EJB Colossus. And there is indeed nothing that could aspire for the epithet 'Bible'. Maybe that situation is to an end by this publication. The title itself, *POJOs in action,* is misleading. It implies to be only about the simple, yet canonical, Java bean with its getter and setter methods. But nothing could be more from the truth. Almost every author of technical books nowadays make acknowledgements to his/her family, thanking them for their support and patience. In this case I believe there IS some substance in the dedications. Because a very pedantic effort has been done to create detailed, yet simple to follow, examples to highlight what the text is about all over the pages. With some experience of Java and J2EE you can easily

follow along the code, without writing anything by yourself. The whole approach is something between developer/architect, and I am convinced both categories would profit from reading it. Hibernate is covered to an extent that you almost not need any other book about it. EJB 3 is explained in a far better way than for example in IBMs 'red books', and the chapters about persistence and transactions in this 'new world' feel almost invaluable. One drawback, (if it now really is a drawback), is that it feels unsatisfactory to read only a single chapter. Let's suppose you want to know more just about facade design. It doesn't give enough information to read only that part. Therefore, this book should be read in its entirety, and it really deserves it. For a developer educated in the EJB 2.0 paradigm, moving to more lightweight J2EE APIs, I can't imagine any better way to get up-to-date than getting a copy of this. Still, the target group is more or less experienced J2EE developers, and a pure beginner in J2EE will soon get lost.

Recommended.

## Design Patterns in Java


**by Steven John Metsker and William C. Wake, published by Addison-Wesley, ISBN: 0-321-33302-0**

**Review by Omar Bashir**

Positives: A must read book for intermediate Java programmers who are increasingly moving into or involved in module-level design. A very interesting book for all Java developers and pattern practitioners as it presents an alternative classification of patterns.

Negatives: Verbose with unnecessary details at times. Demonstrates the application of patterns using a complicated example from a rather specialised domain.

### Detailed review

When faced with a design problem, prudent developers normally spend a considerable time trying to figure out which pattern to use. Implications of using inappropriate patterns range from an inflexible design to complicated control flows resulting in a rigid and brittle implementation. This book will help developers in answering the which pattern question.

Authors present an alternate and a finer grained classification of GoF's patterns that may help developers chose the right patterns for their design problems. Instead of the three classes presented by GoF (Creational, Behavioural and Structural), this book presents 5 classes. These are Interface, Responsibility, Construction, Operation and Extension. Patterns have been classified through their intents and in some cases, the authors have had to provide a deeper interpretation of the pattern intents to justify this classification and the inclusion of patterns in their respective classes. This finer grained classification helps map a particular design problem to an appropriate class, from where it is relatively easier to find the right pattern.

While being structured, the book has an informal expression that may make it easier for some readers. The book contains numerous exercises or challenges, solutions to which are provided at the end of the book and are helpful in practising the concepts explained in the book. Where a pattern being discussed is similar to other patterns, authors discuss the circumstances where one would be more applicable than others. Rather than simply demonstrating the implementation of patterns using Java as a programming language, authors go to some length in explaining classes and interfaces available as a part of J2SE that provide the basis of implementation of some patterns within a Java application.

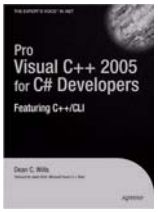
The book mostly uses examples from a single, rather more specialised, application domain of firework manufacturing and operation. Because this is a relatively specialised domain, authors have had to provide a significant amount of domain knowledge in the book. This has made the book a bit lengthy and requires the readers to understand the domain before understanding the problem the example attempts to solve. It may not be possible for many readers to deal with two unknowns concurrently. It would have been better to use an example from a relatively more familiar application domain.

The description of some patterns in this book deviates slightly from the GoF's description. This particularly relates to pattern participants and their relationships. A typical example of this is the description of Mediator. This necessitates reading this book in conjunction with the GoF's *Design Patterns* book where the participants and their relationships promote a higher degree to reusability and flexibility. Finally, a couple of patterns, namely Facade and Builder have not been explained adequately. The example to explain the Facade does not highlight the advantages of using a Facade.

I would recommend this book. The finer grained classification makes the application of patterns more intuitive for developers. Implementation of patterns in Java has been adequately addressed. However, a study of patterns as described by GoF is also required in conjunction with this book. GoF's book provides an in depth description of the individual patterns whereas this book classifies them appropriately that helps developers select suitable patterns for the design problems they encounter. Furthermore, this book explains the implementation of these patterns in Java, which will be helpful for junior and intermediate-level Java developers.

# Windows and .NET

## Pro Visual C++ 2005 for C# Developers




**by Dean C Wills, published by Apress, 379 pages. ISBN-10: 1590596080**

**Review by: Simon Sebright**

This appears to be one of four books in the Apress Visual C++ 2005 series. It's really about C++/CLI, not C++, i.e. the new context-sensitive keywords, etc. that Microsoft have built into C++ and the compiler/linker to allow you to write for the .Net environment in something that feels a bit like C++.

Its point of view is to explain to C# developers what they can do with C++/CLI and aims to point out subtle as well as major differences.

I found the overall structure of the book rather vague, as he himself says it's not supposed to be a step-by-step guide, but a collection of things he thinks are useful to know. As such, I can't really recommend this book as a reference.

The first part of the book is more technical, covering core language concepts with lots of nice code examples in the two languages, particularly pointing out where the same syntax might mean different things.

In the middle, we have a random digression to cover code for supposed interview questions, for example calculating the optimum choice of letting nearly-unbreakable light bulbs fall from a building. This I felt was rather just showing off, because his answers used no features of C++ of note, and were basically C in disguise.

I was less satisfied with the second, 'Advanced', section of the book. It covers generics, templates, and then moves on to nitty gritty C++, including the preprocessor and 'Native C++'. The latter is woeful, allocating more time to `stdlib.h` than **`stl`**, and with more example code than text. His main **`stl`** example uses **`deque`**, should be **`vector`** in my opinion as the container of default choice. **`list`** has a cursory mention, as does **`auto_ptr`**, which apparently can help you if you forget to call **`delete`**.

I was rather put off in the first place because there is an introductory section not only about the author, but about the technical reviewer, not something I have seen before!

To summarise, an interesting first half if you want to know a bit about how C++/CLI and C# compare (for example C++-style destructors map on to **`IDispose`**), but the rest is a red herring. I won't give the book a not recommended status, because there is enough good material in there, though.

Oh, and you can buy the companion eBook for $10 – what a cheek!

[continued on back page]

# ACCU Information

Membership news and committee reports

## View From The Chair

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

At this year's Annual General Meeting, I and the committee inadvertantly triggered considerable discussion over the appointment of the Association's Honorary Auditor. The initial proposal to appoint a new firm of accounts was rejected. I will just note that accounts themselves were accepted subject to clarification, which will be covered in a later piece. There then followed a discussion over the meaning of auditing and the role of the Honorary Auditor.

The constitution itself, in article 7.11, says

> The Honorary Auditor shall be appointed by the Annual General Meeting.

This is, depending on your perspective, either helpfully flexible or unhelpfully vague. The meeting seized the helpfully flexible opportunities. While, in the past, Honorary Auditor had been taken to be an external accountant, following discussion from the floor, the AGM instead chose to appoint a member from outside the committee as Honorary Auditor:

> The association appoints Ewan Milne as auditor, to inspect the prepared accounts.

There was some further discussion, concluding with a further motion from the floor

The committee should consider those constitutional changes that might be required in order to regularise the accounts.

Following discussions with the committee, we have arrived at the following.

### Role of the honorary auditor

During the preparation of the end of year accounts, any external accountants will be dealing with the Treasurer and/or the Chair. The accountants are, however, ultimately employed by and are acting for the membership. Clearly, it's impractical for the accountant to report to every member. Instead the accountant is responsible to the Honorary Auditor, who acts on behalf of the membership.

We do not believe it necessary for all correspondence to be copied to the Auditor, but copies of the financial statements and end of year accounts will be sent to the auditor prior to the AGM. End of year accounts must be approved by both the Chair and the Honorary Auditor before presentation to the membership at the AGM.

If, at any time, the accountants feel there are unresolved questions or outright suspicious behaviour in the books, they should report their concerns directly to the Honorary Auditor. The Honorary Auditor can then make any necessary investigation or take any necessary action.

Should instructions given by the Chair or the Treasurer be in conflict with instructions given by the Honorary Auditor, the Honorary Auditor is the higher authority.

Clearly there's a bootstrapping issue. The Chair will make the initial contact, introducing the Honorary Auditor to the accountant, outlining the accountant's responsibility to the Auditor and so on. Thereafter, changes to those arrangements may only be made by the Honorary Auditor.

### Appointment of the auditor

While not formally adopted in a motion, the AGM discussion endorsed the model used by voluntary organisations in Holland, outlined by Aschwin Marsman. The organisation appoints two people as auditors. Each auditor is appointed for a two year term, but the appointments are staggered. Each year, therefore, one auditor steps down and is replaced. Ewan will stand for the position of Auditor again at the 2008 AGM, on the understanding that he will stand down in 2009. He will seek another person to stand with him, to be appointed until 2010.

### Preparation of the accounts

The preparation of the accounts is the job of the committee. Currently, the day to day management of the accounts is handled by the Treasurer. The Association's accounting year runs to the 31st of December. The 2006 accounts were prepared by Lowson Ward of Birmingham, and the committee expects to use them again to prepare the end of year accounts for 2007. The appointment will be reviewed with the Honorary Auditor following the AGM.

### Changes to the constitution

Since the constitution simply says the the Auditor will be appointed by the AGM, we do not believe that any changes to the constitution are necessary. The arrangements for appointing the Honorary Auditor and their role can be put to the AGM and adopted as a motion. Should future circumstances require a change in arrangements, the AGM can adopt a new motion.

We trust these proposals make sense, and will meet with the Assocation's approval at the 2008 AGM. Should you wish to discuss them please do contact Ewan at auditor@accu.org or Jez at chair@accu.org.

## Membership Report

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

August was previously the end of our fixed membership year, and so many of your memberships are soon due for renewal. The label on the front of your journal mailing has your expiry date, and you will be emailed reminders as that date approaches (unless you've opted out of receiving emails). This may be the first time you've had to renew via the new website – it should be simple to use, but don't hesitate to contact me if you have any trouble. The website also has a newly activated feature – once logged in you'll be able to update your contact and mailing details yourself. Finally, if anyone's interested in receiving a £5 discount on their membership, contact me for details of how to set up renewal by standing order.

A number of members have mentioned social networking sites, particularly Facebook and LinkedIn, to me recently. We've been wondering how closely ACCU should work with the ACCU groups that have been started on some of these services. I'm interested to hear your thoughts about this, so please email me any ideas about how we can use these groups to support your involvement in ACCU.

## Website Report

**Allan Kelly**
**publicity@accu.org**

### The end and the beginning

Is it two years or three that I've been reporting on the ACCU website project? It started when I joined the committee (2003?) and I thought it would be interesting to write an 'officer without portfolio' report. Then I took on the challenge of re-developing our website and it all went from there.

Since then I've reported on our first attempt – a failed outsource project – and the more successful second – a successful outsource project. And all the twists and turns: book database, advertising, CVu online, Overload online, etc. etc. Well this report brings the project to and end. Yes there are a few loose ends to tie up but essentially I now consider the project done, and thus I won't be writing any more website reports, and I will be stepping back a bit.

This month (July) saw the transition of the ACCU mailing lists onto the new server. Rather than the lists being hosted on old Brian – hidden under someone's desk at de Montfort University – the lists and the website live on a shinny server somewhere in a data centre in New Jersey. They are largely managed out of Germany by Tim Pushman, a long standing ACCU member and our chosen sub-contractor to develop the website and everything else.

The website is not an end to itself. It is great to have a shinny new website but the real motivation for re-developing it was something else. We need a good website for two reasons. Firstly it is our face to the world, it shows the world that the ACCU is alive and tells them what

we do. Second, the website is a platform for us to conduct ACCU activities. The move of the mailing lists completes this platform.

I had hoped that the new website would bring in more members. It seems to have brought in a few more but not the hundreds or thousands I would have like to see. To my mind this just goes to prove that while a revised website was necessarily, by itself it is not enough, we need to use the platform to attract more people.

So what next? Well I think David Carter-Hitchin and Seb Rose, as Publicity Officer and Advertising Officer respectively, should have some ideas here. The website can serve to generate advertising and increase our profile. I expect great things from David and Seb!

The ACCU also has to decide what the role of 'Electronic Communications Officer' is these days. Steve Dicks has done an admirable job of keeping the old systems up and running for as long as I can remember. The committee needs to work out what roles we have in place to manage the new website.

Then there are few loose ends to tidy up on the website, around mailing lists, membership, etc. The committee officers responsible can work with Tim directly on this.

And as for me... well I feel a sense of achievement. The end of this project has coincided with several other events and I'll be glad to reclaim my time. As some people know, a few months ago I declared myself an independent consultant specialising in helping companies improve their development activities (more on my company website www.softwarestrategy.co.uk). I really need to devote more time to this project.

Second, at this year's EuroPLoP conference I was elected co-chair. End to end this is a three-year commitment covering two conferences. (Kevlin Henney and myself are EuroPLoP regulars, several other faces are familiar from ACCU conferences but I always hope more ACCU'ers should get involved in the patterns community. If you feel like trying your hand at pattern writing, or just want to join in the conference, get in touch or look at the EuroPLoP website http://hillside.net/europlop/).

I'm not resigning from the ACCU committee just yet, although I may do as the EuroPLoP work builds up. But I do intend to take a backseat on the website from here on.

Finally, I'm hoping that I'll be able to get back to writing for Overload. My period running the website project has coincided with my writing a book (O, since you asked, it's called *Changing Software Development* and should be out in January 2008 from John Wiley & Sons.) The book has taken most of my writing time and the website my ACCU time, so hopefully… no promises.

# Book Reviews (continued)

## .NET Internationalization

**by Guy Smith-Ferrier, published by Addison-Wesley, 630 pages. ISBN-10: 0321341384**

**Review by Simon Sebright**


Recommended

This is a very strong book on the subject of i18n (Internationalization) of .Net projects, be it forms or web applications. It covers issues in good depth, with much detail and a commanding writing style.

Along with the book are online code examples and tools. I haven't looked at these, but from the snippets in the book, they look to be well written. The tools are essentially extensions to the Visual Studio toolset for dealing with i18n of projects.

The book is full of interesting examples where different languages and cultures bring new concepts to challenge the developer. For example, calendars where there are not necessarily 12 months in a year, or left-to-right issues, or cultures where there is more than one way to sort strings.

The i18n issue covers good practice even in non-localised programs. Not hard-coding things, using resource files effectively, etc., so I learned some useful things even though I am not working on this issue directly.

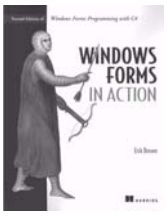
Index pretty good, although admittedly the only term I actually wanted to look up wasn't there.

Again, recommended. Even if you don't have an i18n issue to deal with, it's good to know the concepts presented in this book

## Windows Forms in Action

**by Erik Brown, published by Manning, 803 pages. ISBN-10: 1932394656**

**Review by Simon Sebright**




Recommended with reservations

This book aims to get you up to speed with WinForms 2.0 using C#. It takes you through most of the usual things you need to worry about – menus and other tool strips, common dialogs, custom dialogs, controls (buttons, text, etc.), calendars, progress controls, 'bells and whistles', custom controls, explorer views, MDI, data binding, and odds and ends (printing, drag and drop, web browsing, settings, deployment).

Throughout the book, he focusses on one central application, a photo-album editing suite, although he does occasionally uses other examples where this application is not suitable to demonstrate something. Generally, this works well, and I even thought of creating such an app myself (I wasn't convinced by some of his code design!) based on this work. Sometimes it grated a little though, and in comparison with another windows forms book I have recently read, it has pros and cons.

One thing I really liked was the fact that the example code was not riddled with stupid comments such as `// Create a widget`; next line `Widget w = new Widget();`. We can well do without that style, and this book uses a format of describing the code in the book with a two-column table, which works well.

I didn't try to download and compile any of the code, but generally, I saw very few mistakes in the printed version. Some of the techniques he uses (returning early if parameters are null, for example) are not my thing, but it generally made sense.

I thought the coverage of databinding was rather weak, a case where limiting things to the application under development didn't work that well.

But, most topics were covered with what appeared to be a good command of the subject. I had recently read another good book on this subject, and was pleasantly surprised to read new things here, and also sometimes better explanations of concepts.

Overall – recommended with reservations. Those reservations are that certain subjects are dealt with rather lightly, for example, data binding, printing, deployment. But, these are major topics in themselves, and no book on WinForms can cover everything in detail. The concepts are here. Finally, the index is pretty comprehensive.