the magazine of the accu

www.accu.org

{cvu}

Volume 27 • Issue 1 • March 2015 • £3

{cvu}

Volume 27 Issue1
March 2015
ISSN 1354-3164
www.accu.org

Features Editor
Steve Love
cvu@accu.org

Regulars Editor
Jez Higgins
jez@jezuk.co.uk

Contributors
Silas S. Brown, Ian Bruntlett, Thaddaeus Frogley, Pete Goodliffe, Ralph McArdell, Chris Oldwood, Roger Orr, Mark Radford, Emyr Williams

ACCU Chair
chair@accu.org

ACCU Secretary
secretary@accu.org

ACCU Membership
Matthew Jones
accumembership@accu.org

ACCU Treasurer
R G Pauer
treasurer@accu.org

Advertising
Seb Rose
ads@accu.org

Cover Art
Pete Goodliffe

Print and Distribution
Parchment (Oxford) Ltd

Design
Pete Goodliffe

accu

Private Funding

Recently a large manufacturer of consumer 'smart' televisions came under fire when it emerged that the voice recognition feature built-in to the devices was recording the audio and sending it off to some 3rd party for analysis. The company later made clear who the 3rd party was, and also the fact that the devices were using industry standard encryption to send the data, and that the listening feature had to be intentionally activated (i.e., the TV wasn't always listening). There are similar parallels with a certain well-known search provider using voice recognition for searching – a quick Goo...sorry, search for 'Ear of Sauron' should enlighten those not already familiar with this. Such stories aren't unique, nor are they even new, but it seems that they are becoming more prevalent.

That's either because it's happening more often, or because people are starting to sit up and take notice. Or both, of course. It's different to the idea of the government having an open licence to eavesdrop on all your communications in the name of national security – although that too has become a hot topic in the media. The idea that the consumer electronics and Internet Browser companies might be **selling** our data for the purposes of either advertisement targetting or just simply reducing the unit cost of their goods is unsettling to many people. And yet, many many more people are happy for that to happen, and freely give their privacy away in return for some free or cheap service that they value more. Which raises the point that if the vast majority of people don't value their privacy enough to complain about it being eroded, then everyone will lose it. Which is why the fact that these things **are** hot topics is important.

Of course, in the matter of voice recognition for TVs or searching, if you're unconvinced that your data is not being mis-used to make someone else money, then at least you're maybe in a better position than many other people in that you might be able to make your own voice recognition feature and use that, instead of trusting to someone else's. Perhaps it's time for another hobbyist electronics revival. If you do, then I expect lots of other readers would be delighted to read about it here!

Steve

STEVE LOVE
FEATURES 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 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

REGULARS

FEATURES

SUBMISSION DATES

C Vu 27.2:	1st April 2015
C Vu 27.3:	1st June 2015
Overload 127:	1st May 2015
Overload 128:	1st July 2015

WRITE FOR C VU

Both C Vu and Overload rely on articles submitted by you, the readers. We need articles at all levels of software development experience. What are you working on right now? Let us know!

Send articles to cvu@accu.org. The friendly magazine production team is on hand if you need help or have any queries.

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

Coders Causing Conflict

Pete Goodliffe lights the blue touch paper and retires to a safe distance.

Words kill, words give life;
they're either poison or fruit – you choose.
~ Proverbs 18:21 (MSG)

Unless you're a recluse, you will not go through life without meeting conflict in some form or other. It's a fact of human existence. No one is immune; conflict is inevitable.

For the programmer, conflict can be bad: non-productive and hurtful. Have you ever written some code and then feared jibes from your colleagues? No fool would write code like that, are you stupid or something?

Sadly, statements like that are not unheard of in the workplace. But they are clearly not at all appropriate. No one should suffer aggressive criticism, be demeaned, or needlessly discouraged. That kind of conflict is dangerous – it can knock your confidence and ruin effective team work.

But (and pay attention now) conflict is not *necessarily* a Bad Thing.

There is a valid place for healthy discord in the construction of high quality software. 'Conflict' does not necessarily mean war. Conflict doesn't have to be an out-and-out shouting match, a cat fight, a tirade of unhealthy criticism, or a form of passive-aggressive power struggle. It can simply be a disagreement that, once resolved, will lead to a better solution. We *can* use conflict for good.

Perhaps we can use different terms to avoid confusion, or make this sound less sensational. We'll talk about 'disagreement' versus 'conflict'. Whereas conflict is confrontational, aggressive, abrasive disharmony, 'disagreement' is just what it says: failure to meet consensus on a matter. Sometimes it can be a deep, strong disagreement. But it doesn't have to be disrespectful, rude, or acrimonious.

It's perfectly possible, even expected, to like and respect a person, but to not agree with a coding decision they've made.

We must learn to harness disagreement to craft better code, and to not take it as a judgement of (in)ability. Being challenged about your opinion, or about the quality of a design can lead to unexpected solutions. Other people's thoughts and opinions can spark an idea that will lead to an ultimately better solution. Indeed, without conflict we might only make inspired boring, incorrect software. In this respect, conflict is natural, normal, even necessary.

We don't have to turn all 'disagreement' into 'conflict'. Indeed, the professional programmer actively seeks to avoid doing so.

In this article, we'll look at how we can be good programmers, good team mates, and use conflict/disagreement for the benefit of the code, the developers, and the customer.

Reactions

What's your natural reaction when someone disagrees with you?

Some programmers respond well: phlegmatically, with good grace, not taking offence. Other programmers take it incredibly personally, and the merest whiff of disagreement knocks their confidence.

I remember once being involved in a heated design discussion about a software system, in which a number of programmers were becoming increasingly animated about their opinions. It was a loud discussion, with much hand waving and some energetically expressed points. At no point was anyone being rude, or personally insulting. We were enjoying the design experience; we really cared about exploring to find the best result. However, there was one programmer who sat on the periphery, and couldn't engage. Personal circumstances had led to him being more timid and reserved, and he didn't feel able to contribute at the same level as his colleagues. He felt that the discussion was a fight; and feared the outcome. In our excitement we didn't realise he felt intimidated by the discussion. We missed his input, and he felt unable to contribute.

This shows how different people view disagreement in different ways, and can react and engage differently.

People view and engage in disagreements in different ways. To work effectively in a team you must appreciate this, and be able to relate to team-mates appropriately.

These different reactions stem from a person's temperament, culture, and their personal situation. It may even depend on how much sleep they had last night (parents of young, sleepless, children know what I mean). Of note: be careful about making generalisations around gender. I know some very timid, introverted male programmers who struggle with criticism and some female coders who can put their point across with force!

Reacting well

Disagreement is not undesirable or to be avoided. Handled in a constructive and appropriate way it is a very healthy thing indeed. It's the way we *choose* to react that determines whether it's productive or dysfunctional. There are a few key things that help us react *well*.

Avoid being defensive

When someone disagrees with you, the natural human reaction is to become defensive and argue. But stop; try to avoid this as a first reaction, arguing further rarely helps resolve anything. Rather than assert your opinion, consider whether they do have a valid point. Perhaps you *should* revise your opinion, especially if it was a decision you made some time ago. Circumstances change over time, code never stands still, designs evolve, and the assumptions around which you made a decision may no longer be valid.

Beware of pride

It's good to *take pride in your work*, to care about what you craft. But don't be too proud of yourself, and fear a public dent in your public profile.

Pride can turn a simple disagreement into a full-on quarrel. Don't let your fear of being shown to be wrong lead to acrimonious conflict. When you realise you've made a mistake be prepared to admit this, rather than continue to fight to save face.

And of course, when you *are* wrong, don't be too proud to admit it. Gracefully.

Expect to be wrong

Remember that you will not always be right. Expect this.

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@goodliffe.net or @petegoodliffe

Since there is almost always more than one way to solve a problem, a good discussion may throw up a new, better, alternative. The best software development is collaborative, drawing on the abilities of an entire team; harnessing the wisdom of their crowd.

Don't make it personal

Jerry Weinberg, in his seminal book *The Psychology of Computer Programming* [1], speaks of *egoless programming*. This is a great illustration of an appropriate reaction when your work is criticised. Leave personal feelings at the door, don't take offence, look at your work impartially.

Now it's true that we can't completely disconnect from this. Nor should we: taking as pride in what you craft leads to a better outcome. But don't see everything as a personal attack.

Don't avoid disagreement

One technique to avoid conflict is to actively avoid situations where it might arise. Don't ask people's advice. Don't review code. Don't have design meetings. Don't engage in water-cooler design discussions.

That's not a winning approach.

'Remember how Conway's Law teaches us that a software's design tends to follow the lines of communication in the team that built it. In order to construct cohesive, well-functioning software, we need a cohesive well-functioning team!

Otherwise form will follow dysfunction; if you avoid talking to people (say, in order to avoid confrontation) then your software will also have poorer internal communication.

Take responsibility for respect

Mature programmers take responsibility for fostering healthy communication in the team. To do this, we must learn to interact well, to handle conflict professionally, and to *respect* people.

If you respect the people you talk to, you will try to disagree without being *disagreeable*. A good lens to view this through is the *Retrospective Prime Directive* [2]:

> Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.

In any discussion, it is sensible to assume that the people you discuss with are working to the best of their knowledge and ability, and that they are working for the best software outcome, not to apportion blame or make you look silly.

The atmosphere of a team is not determined solely by the leader; it is defined by the members of that team. There are certain team values that make this work: everyone is valued; everyone is important; everyone has a right to be heard, and has as valid point to make.

How to disagree well

How we handle conflict determines whether it is productive or destructive. Handled well, disagreement will lead to novel solutions, to learning, and to better software. Handled badly, though, it can lead to bitterness and resentment. Badly handled conflict can even lead to violence. (Yes, it's not stereotypical for programmers to resort to fisticuffs, but weirder things have happened.)

Learn to deal with conflict well. Meet criticism and disagreement with an open mind, not snappy reactions.

Leave personal feelings at the door, don't take offence, look at your work impartially.

Dealing well with conflict is an important life skill. It's something many of us have to learn; few people are naturally politically savvy. Whilst you should definitely not aim to manipulate people, knowing how to interpret people's reactions and deal with them sympathetically and appropriately will definitely help you become a better programmer. This, too, is not stereotypically a strong point for introverted developers.

Listen

Remember that a conversation involves listening as well as talking. Make sure you *actively* listen to other people. It can be very tempting to presume you know what another person thinks, and to start arguing with this. But you may not yet accurately understand their opinion. Stop. Listen. Actively.

If nothing else, this is a form of giving them respect.

You don't want to waste your time arguing with an opinion that no one has! So first listen. And try to understand their opinion. A classic technique is to repeat back their point of view, reinforcing their point in your mind, and forcing you to listen. Doing this also gives you more time to formulate a considered response.

Know when to stop

Sometimes you won't reach an agreement. Know when conversation is no longer productive. Be able to end conversation gracefully without having reached a conclusion. Perhaps schedule another time to discuss the matter after you've had time to walk away and consider it for a little longer.

Don't keep fanning the flames of a destructive conversation.

Sometimes you can start disagreeing with someone by mistake! You're talking at cross purposes, when actually you have the same opinion. When you realise you're 'strongly agreeing' stop the pointless prattle and move on!

Talk well

Whilst listening is crucial, it helps to be able to articulate your point well. How often have you been frustrated by an inability to explain a point well in the heat of discussion, when a multitude of erudite descriptions enter your mind after the conversation has finished?

Practice:

- saying things succinctly (why use 100 words when 10 will do?)
- not 'talking over' other people – let them finish
- explaining your idea from the other person's perspective; they don't have all of your tacit knowledge
- make sure they understand the point that you are making correctly (without patronising them!)
- watch the kind of words you use; words have power, you can't take them back – there is no control-z for a conversation!

Understand your emotions

Dealing with emotions that arise during 'heated' discussions can be tricky. Anger, frustration, disappointment, insecurity, and pride can all begin to make you behave less rationally. Do you feel threatened or excited, empowered or emasculated? Do you tend to cave in when heated discussion picks up, or instead feel a rising need to validate yourself; to 'win' and become Top Dog?

Identify how you tend to react in these kinds of situation. This can help you adapt unhealthy and non-constructive behaviours, and plan out ways to improve your behaviour for the next time.

Reflect on this now.

When you realise you're 'strongly agreeing' stop the pointless prattle and move on!

Using ACCU Membership for Unique IDs

Silas S. Brown considers the case for identity.

Java package names, SGML/XML schema names and similar things are often expected to include the developer's domain name to ensure uniqueness. While this in itself certainly doesn't provide any real security (others can falsely use your domain name simply by typing it), and it doesn't even guarantee uniqueness over time (if in future you transfer ownership of your domain to someone else, then your old code will be using names you no longer own), but in conjuction with other measures it does seem reasonable – that is, if you happen to have a domain name handy.

You could use a subdomain that you obtained on one of those 'free dynamic DNS' services, but your continued ownership of this will depend on the whim of that company. But it might be easier, and also more reliable, to simply use the domain of a long-established organisation of which you are a member, and add in your membership number or user ID to make sure it's unique. For example if I were to say

```
  package org.accu.m94137.appName;
```

(94137 being my ACCU membership number), that shouldn't collide with anything by mistake. Notice that the package name does not have to be a real domain: ACCU don't have to set their DNS server to actually resolve m94137.accu.org to an IP address, nor to set up a machine at that IP to handle any particular query that says it's for m94137.accu.org; it's only for uniqueness.

If the ACCU committee could state officially that members are entitled to use their membership numbers on accu.org when they need domain-based unique IDs, that would be extra reassurance that ACCU won't be using these subdomains for some other purpose later, and would confer a small extra benefit to members at negligible cost to the ACCU. But even if they don't officially commit to this, I think the chances of collision are very slight. ■

SILAS S. BROWN

Silas S. Brown is a partially-sighted Computer Science post-doc in Cambridge who currently works in part-time assistant tuition. He has been an ACCU member since 1994 and can be contacted at ssb22@cam.ac.uk

Coders Causing Conflict (continued)

Watch your body language

Having a passion for your opinion, and caring about the code means that you're involved. This is a good thing. You get animated in discussions. You wave your hands, speak excitedly. You're anxious to get your point across, to share the marvellous insight that you just had.

But don't be overbearing.

Don't reinforce a power struggle with your body language. Speak softly, don't raise your voice or call the other person stupid.

Avoid a closed or confrontational posture that conveys you do not respect or value the other person.

Seek another opinion

If you can't reach a consensus, bring someone else in for a casting vote. This is a straightforward strategy to resolve a disagreement, if you all respect the third party.

But remember that sometimes adding more people can sometimes make it *harder* to reach a single point of agreement, as there are more voices to be heard!

What if you spot others arguing? Should you step in to help resolve conflict? Should you offer to *be* the third party? It may not be your battle to get involved with. But it may be worth helping resolve. Again, treat others with respect. Perhaps you should first ask if you can help in any way.

The resolution

You don't have to agree with everyone all of the time. And you don't have to hide from conflict; it can be a healthy part of collaborative software development. Disagreeing openly can foster conversations that lead to better software.

- However, it's important to always treat people with respect. Healthy communication builds a team, it is never unnecessarily rude, personal, demeaning or political. Be aware that healthy disagreement can turn into bitter conflict. n

Questions

1. What do you recall as the greatest moment of conflict you've encountered in your programming career? How did you resolve it? What were the results of that conflict (both on the software, and the team)?
2. Do you think you are currently the cause of more disagreement or resolution in your project?
3. Does conflict tend to arise more between closely working people in the same team (e.g. developers-with-developers, or developers-with-close-knit-testers), or between departments (e.g. developers-with-management)? Why?
4. What do you think are the most important skills needed to resolve conflicts effectively?
5. How do these skills fit into your daily coding regimen?
6. What measures (if any) can a team/company adopt to prevent conflict, or to help resolve conflict in a healthy way? What have you seen that actually works?
7. Do cultural differences play any part in the shape of healthy working relationships, and the way you work with team members? Why?

Reference

[1] Gerald Weinberg (1971) *The Psychology of Computer Programming*
[2] http://www.retrospectives.com/pages/retroPrimeDirective.html

LAMP on Ubuntu

Ian Bruntlett shares his notes on setting up a basic web application.

Sometime in 2014 I started working my way through an Internet programming book – *Learning PHP, MySQL, JavaScript, CSS & HTML5* (LPMJ). Whilst it is a good guide, I had to research certain system administration details myself. This document was written using my personal notes, text books and a computer with a fresh install of Ubuntu 14.10. As approved of by that book, I have reused some of their examples. Please note this is a study exercise – commercial websites are likely to do things differently for scalability and security.

First my personal preferences. I installed Synaptic Package Manager because I like its front end – it helps me explore the packages available. I also installed ttf-mscorefonts-installer which in turn installs some free fonts from Microsoft – in particular it provides me with Times New Roman and Comic Sans MS. I also installed an editor – emacs – because I like it :)

Packages to install..

Here are the actual names of the packages involved:

- apache2, apache2-doc – this is the webserver and its documentation in particular /usr/share/doc/apache2-doc/manual/en/index.html
- mysql-server, mysql-client (for the server you will need to decide on a 'root' MySQL password – this is not the same as your 'root' Linux password). Do not lose it.
- php5, php-doc (manual in /usr/share/doc/php-doc/html/index.html)
- php5-mysql

Setting files up

Start up a terminal window/shell window. When you are typing these commands, replace **`ian`** with your username. Type in this command:

```
sudo chown ian:ian /var/www/html
```

The directory /var/www/html is used by Apache to find its HTML files. This command changes the owner and group ID of that folder so that you can put your HTML files there. There is an index.html file, owned by root, already in that directory. To view it, start a web-browser and type in an address of localhost. You should see the Apache2 Ubuntu Default Page. As we are going to modify that file for our own purposes, do this:

```
cd /var/www/html
mv index.html index_original.html
```

From now on I am going to refer to var/www/html as 'HTML Home'.

Our first webpage

Still in HTML Home, start a text file and type in Listing 1, saving it as `index.html`.

You can then check the integrity of the HTML script by clicking on the 'Site to validate etc' link, select 'Validate by File Upload' and use the 'Browse' button to get to /var/www/html/index.html

Listing 1

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ian's LAMP index.html</title>
    <meta http-equiv="Content-Type"
          content="text/html;charset=utf-8">
  </head>
  <body>
    <h2>Ian's LAMP files test area</h2>
    <ul>
      <li><a href="http://validator.w3.org/">
         Site to validate HTML5 files.</a></li>
    </ul>
  </body>
</html>
```

Listing 2

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ian's LAMP experiments</title>
    <meta http-equiv="Content-Type"
          content="text/html;charset=utf-8">
  </head>
  <body>
    <h2>Ian's LAMP <?php echo __FILE__ ?> for CVu
    magazine </h2>
    <p>
      <?php
        echo "blank";
      ?>
    </p>
  </body>
</html>
```

Configuring Apache for ease of development.

Apache – displaying errors present in PHP.

To display syntax errors, assign rights to edit `/etc/php/apache2/php.ini` and in that file, set these settings on: **`display_errors`**, **`display_startup_errors`**.

They are very useful. After changing these settings either send the Apache process a SIGHUP or reboot your computer.

When you want to use your development computer as a webserver (not always the best idea), set these (**`display_errors`**, **`display_startup_errors`**) settings to Off.

Our first piece of PHP5

Insert this line into `index.html`:

```
<li><a href="http://localhost/blank.php">An
almost blank PHP file.</a></li>
```

And put the HTML file in Listing 2 in HTML Home, named `blank.php`.

Save it, press F5 and click on 'An almost blank etc' file. In the **`<h2>`** tag pair, you should notice there is a bit of PHP there. Congratulations, you have created and run your first PHP programme.

IAN BRUNTLETT

On and off, Ian has been programming for some years. He is a volunteer system administrator for a mental health charity called Contact (www.contactmorpeth.org.uk).

Our system has been set up to have a webserver (Apache), a scripting language (PHP5) and a database server (MySQL).

Creating a database

Go to a command prompt and type in "mysql -u root -p" and type in your MySQL root password. This gets you to the MySQL command prompt.

```
ian@turing:/var/www/html$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with
; or \g.
(some redundant details removed).
Type 'help;' or '\h' for help. Type '\c' to clear
the current input statement.
mysql>
```

Before we can store any information, we need to set up a database. Type in these commands:

```
create database publications;
use publications;
```

And you should see:

```
mysql> create database publications;
Query OK, 1 row affected (0.00 sec)
mysql> use publications;
Database changed
mysql>
```

Creating a database table

Also at the MySQL prompt, type:

```
drop table classics;
CREATE TABLE classics (
 author   VARCHAR(128),
 title    VARCHAR(128),
 category VARCHAR(16),
 year     SMALLINT,
 isbn     CHAR(13),
 INDEX(author(20)),
 INDEX(title(20)),
 INDEX(category(4)),
 INDEX(year),
 PRIMARY KEY (isbn)
) ENGINE MyISAM;
```

Populating a database table

Here are some SQL statements to populate the above `classics` table:

```
INSERT INTO classics (author, title, category,
year, isbn )
VALUES('Mark Twain','The adventures of Tom
Sawyer','Fiction', 1876,'9781598184891');
```

And to verify that those statements worked, type this into a MySQL prompt:

```
select * from classics order by author;
```

You should get the result shown in Figure 1.

Accessing a database on a web page

First you need to have database login details. In the LPMJ book, a file called `login.php` is used. Type Listing 3 into `/var/www/html/login.php`:

Listing 3

```
<?php // login.php
  $db_hostname = 'localhost';
  $db_database = 'publications';
  $db_username = 'root';
  $db_password = 'Your Password Here';
?>
```

Listing 4

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ian's LAMP and PHP experiments</title>
    <meta http-equiv="Content-Type"
          content="text/html;charset=utf-8">
  </head>
  <body>
    <h2>Ian's LAMP <?php echo __FILE__ ?> from
      Chapter 10 Accessing MySQL Using PHP</h2>
    <p>
    Doing database stuff
<?php
  function connect_to_db($host,$user,$passwd)
  {
    echo "<br>Hi from connect_to_db()<br>";
    echo "<br>login details : Username $user,
    Hostname $host<br>";
    $db_server = mysql_connect($host, $user,
      $passwd);
    print "<br>db_server = $db_server<br>";
    if (!$db_server) die("Unable to connect to
      MySQL: " . mysql_error() );
    else echo "Connected to server<br>";
    mysql_select_db("publications")
    or die ("Unable to select database: "
     . mysql_error() );
    echo "Database selected<br>";
    return $db_server;
  }
  function one_at_a_time_results($result)
  {
     echo "Hi from one_at_a_time_results $result
       <br>";
     $rows = mysql_num_rows($result);
     for ( $j=0; $j<$rows; ++$j)
     {
       echo 'Author: '   . mysql_result
         ($result, $j, 'author') . '<br>';
       echo 'Title: '    . mysql_result
         ($result, $j, 'title') . '<br>';
       echo 'Category: ' . mysql_result
         ($result, $j, 'category') . '<br>';
       echo 'Year: '     . mysql_result
         ($result, $j, 'year') . '<br>';
       echo 'ISBN: '     . mysql_result
         ($result, $j, 'ISBN') . '<br><br>';
      }
  }
```

Figure 1

```
+----------------------+-------------------------------+-------------+------+---------------+
| author               | title                         | category    | year | isbn          |
+----------------------+-------------------------------+-------------+------+---------------+
| Mark Twain           | The adventures of Tom Sawyer | Fiction     | 1876 | 9781598184891 |
+----------------------+-------------------------------+-------------+------+---------------+
1 row in set (0.00 sec)
```

The Developer's Sandbox

Chris Oldwood wants to be in control.

We often talk about developers working in 'isolation' or use the term 'sandbox' to describe an environment that cuts ourselves off from the outside world. But what exactly do we mean by these terms – what is on the inside and what is on the outside? There are often many different sandboxes in the development process too, for example the build pipeline and the various test environments. Some of these are 'larger' than others, where size could be measured in terms of the number of processes, machines and network paths collaborating together.

The aim of this article is to look at various different sizes of development-level sandboxes and explain what problems I've commonly encountered when using them. As will become apparent later you'll see what I personally consider is the ideal sandbox for day-to-day, user-story development work. That last qualification is important because when I'm doing support or localised system-level testing I probably need to loosen the sandbox constraints to bring in some external dependencies, but ideally under tightly controlled conditions [1].

Component/integration/acceptance tests

To be doubly clear this is not about unit tests – it's about developing and running component, integration and acceptance level tests. Unit tests naturally have no dependencies but once we have run those and our confidence starts to build it's nice to start bringing in fast running tests that talk to real dependencies to start gaining further confidence that all the units have been assembled correctly and are still working together as intended.

The kinds of services I've been developing in recent years have been developed outside-in, starting with a failing acceptance test and then moving inside the service, sometimes leading to writing a combination of integration, component and/or unit tests before 'unwinding the stack'. Whilst they often have a real (e.g. out-of-process) database and messaging service in play for the acceptance tests they all still run to completion within a couple of minutes. Even so they exercise the majority of code paths which leads to high degree of confidence in the functional aspects of any change before it is committed by the developer.

Hence this article is about where those 'heavier' 3rd party dependencies live and how we can cope with the potential disruptions they have a habit of producing. Once we leave the simplicity and beauty of the in-memory sandbox that unit testing provides we enter a realm where side-effects can become persistent. If we're not careful we start chasing our own tails due

CHRIS OLDWOOD

Chris is a freelance developer who started out as a bedroom coder in the 80s writing assembler on 8-bit micros; these days it's C++ and C#. He also commentates on the Godmanchester duck race. Contact him at gort@cix.co.uk or@chrisoldwood

LAMB on Ubuntu (continued)

Listing 4 (cont'd)

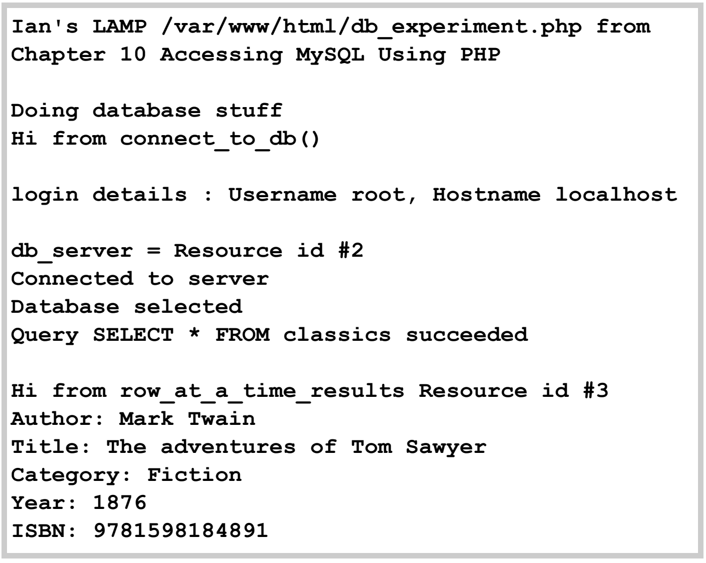
```
  function row_at_a_time_results($result)
  {
    echo "Hi from row_at_a_time_results $result
      <br>";
    $num_rows = mysql_num_rows($result);
    for ( $j=0; $j<$num_rows; ++$j)
    {
      $row = mysql_fetch_row($result);
      echo 'Author: '   . $row[0] . '<br>';
      echo 'Title: '    . $row[1] . '<br>';
      echo 'Category: ' . $row[2] . '<br>';
      echo 'Year: '     . $row[3] . '<br>';
      echo 'ISBN: '     . $row[4] . '<br><br>';
    }
  }
  require_once 'login.php';
  $db_server=connect_to_db($db_hostname,
    $db_username,$db_password);
  $query = "SELECT * FROM classics";
  $result = mysql_query($query);
  if (!$result) die ("Database access failed: "
    . mysql_error() );
  echo "Query $query succeeded<br><br>";
  //one_at_a_time_results($result);
  row_at_a_time_results($result);
  mysql_close($db_server);
?>
    </p>
  </body>
</html>
```

Then you need a programme to access the database in HTML Home. Something like /var/www/html/db_experiment.php (Listing 4).

Once that programme is stored, it can be run from within a web browser. Start a browser and type in this address: http://localhost/db_experiment.php

You should see something like Figure 2.

And that's it! ■



```
Ian's LAMP /var/www/html/db_experiment.php from
Chapter 10 Accessing MySQL Using PHP

Doing database stuff
Hi from connect_to_db()

login details : Username root, Hostname localhost

db_server = Resource id #2
Connected to server
Database selected
Query SELECT * FROM classics succeeded

Hi from row_at_a_time_results Resource id #3
Author: Mark Twain
Title: The adventures of Tom Sawyer
Category: Fiction
Year: 1876
ISBN: 9781598184891
```

Figure 2

to test failures outside our control, or worse, start to ignore test failures we believe our changes could never have caused; that is the start of a very slippery slope.

Network-level sandbox

At the extreme our degree of isolation might only be to avoid us affecting the production system. More typically we create named environments, such as DEV and UAT, which are usually partitioned on a (virtual) machine-wide basis.

From a developer's perspective this kind of environment means there is some form of shared infrastructure in use, such as a database, file share or message bus. Once we have any form of shared resource we start to bring in the possibility of noise to the development process, which, as mentioned earlier can manifest itself as test failures outside our control. Nothing kills a state of 'flow' more readily than an unexpected test failure and shared resources increase the likelihood of that happening.

When any test fails my immediate reaction is that it's my fault – I'm always guilty until proven innocent. Not every line of code (production or test) I come across is easy to reason about and so I have to assume the worst. Tolerating transient test failures just leads to distrust and an eventual 'blindness' whereby the test provides no value because its failure is ignored. Eventually someone will get fed up and just comment the test out altogether or add the 'ignore' attribute so that it never runs. Now the test is just an illusion that buys us false hope.

the sharing of services ... can make remote working even more painful

Barriers

Years ago the need to share infrastructure was borne out of cost – not every developer could afford to have an instance of SQL Server or Oracle on their machine due to the high licensing costs. In today's world there are 'developer' editions of the big iron products which help to alleviate this cost. The NOSQL databases are usually free and only come with the kind of limitations a developer would never breach as part of their normal development cycle anyway. The same goes for web servers and message queuing products too. What is more likely to make this set-up unusable is either a draconian usage policy [2], where you have no rights to install anything, or the machine is woefully underpowered and couldn't take the extra strain of additional services in tandem.

Putting aside these reasons why you might have to suffer the sharing of services another problem they create is that it can make remote working even more painful. If the organisation does not provide a VPN or some form of remote desktop then you cannot easily work outside the confines of the office. Even with a decent broadband connection I've seen test suites take an order of magnitude longer to run because of the latency that starts to dominate on all the underlying remote service connections created during the test runs. In some cases the firewall covering the VPN may only be configured for the 'standard' network traffic (think SharePoint) and so you might be blocked from accessing your modern NOSQL database due to its use of unusual port numbers. As for working on the train during your daily commute or business trip, that would just be a non-starter.

Partitioning data

The usual technique for dealing with shared infrastructure is to partition either at the schema-level or data-level. For databases you can have your own named database within an instance and for message queues the queue name could encompass some derivable prefix or suffix, such as your login or developer machine. These values work nicely for an out-of-the-box configuration option but as we shall see later being able to easily override them, such as through a defaulted environment variable, is often desirable anyway [1].

At least with this kind of partitioning you are only really sharing the service itself with other developers. Hopefully the product is reliable, which is presumably why you picked it, the data volumes are low, and so the chances of failure are mostly down to hardware issues of some description. The use of virtual servers would make this kind of problem largely a thing of the past if it wasn't for all the bureaucracy that can be required to get the VM up and running again on another host. The security patching cycle that goes on every month has also been known to take down a crucial development server or two in the past so it's not all plain sailing.

The second option is to partition at the data level. This usually involves adding prefixes and/or suffixes to the data in such a way that you can uniquely identify your own test data to distinguish it from your colleague's. This can be a useful technique but it starts to have an impact on both your production and test code as your design inherently has to acquire the ability to pre and post-process data down in the stack. If you're lucky this will already be a required part of the design and the application of some classic design patterns, such as DECORATOR, will minimise the impact. If not you're adding complexity to the code which might be avoidable via other means. Granted it may not be a huge leap in complexity, but it is still something which differs (code wise) between what you run in development and what is actually running in production.

Due to the potentially low-level nature of trying to work this way, it also affects the way you write your tests. Now you have to be mindful of other developers and so can't just truncate a data table or purge a message queue. Instead you have to delete only your own data from a table or carefully drain the message queue without disturbing the order of other messages. I'm not even sure the latter is entirely possible (I've only seen teams take the hit on the disruption when tests are run concurrently, i.e. just keep running them until they do pass).

Sometimes you are not in complete control of all the data which is generated – database identity columns are a case in point. If you rely on them to generate the IDs for your entities you either have to write a different query to identify your data or make sure you capture the identity values in some way so you can refer to them later. Identity columns might not partake in transactions so you can't assume that *N* inserts will result in rows with *N* consecutive identity values when run concurrently with other tests.

Aside from server failures the other big disruption to using shared relational databases with data-level partitioning is schema changes. If another developer needs to change the schema or some shared code, such as a stored procedure, it affects everyone. This also implies that the database must always be running the latest code – reverting the database back to match the current production schema is out of the question without interfering with the rest of the team. Whilst NOSQL databases are inherently schema-less this does not mean schema problems do not happen. On a fast moving codebase with heavy refactoring the schema may be changing rapidly without formally bumping any internal 'schema version number' such that breaks in serialization occur. They should be infrequent, but it's important to be aware that it's still possible when sharing infrastructure.

Machine-level sandbox

Being able to work isolated from the rest of the team (infrastructure wise) is a useful step up. Once you remove what's going on around you from the potential sources of noise you then only have yourself to blame when something goes wrong; although maybe a group policy update or security patch will still catch you unawares every now and then.

Being master of your own castle gives you the power to play and tinker to your heart's content without the fear of disrupting your entire team. Want to restart your database, IIS, or the messaging service to blow away the cobwebs? No problem, just do it as there's no need to coordinate this kind of activity with everyone else.

This doesn't mean that your setup can be entirely ad hoc though. Although you can assume that the service name will be 'localhost' in any configuration it helps if developers stick to a consistent set of port numbers, etc. so that the default configuration stored in the version control system should just work on any developer's machine. This is especially useful for getting a new joiner up and running quickly.

Shared local services

Exactly because everything is running locally on every developer's machine you don't have to play games with the names of databases or message queues as you can all use the same name. The ability to easily configure such settings is useful for other test environments or scenarios, but for the common case – developing user stories on the default integration branch (e.g. trunk) – it should just work as is.

Whilst this set-up is more stable than having to share infrastructure it does not come without it a few of its own problems. Sometimes you might feel like a part-time system administrator, which is A Good Thing from a DevOps perspective, but can be a distraction when you really just want to get your story finished. For example IIS and IIS Express play games in the background that can leave your test failing, despite you putting in the working implementation, only to find it was still using the old, cached failing implementation. User rights are another area of contention as services tend to run under privileged accounts which makes debugging harder. Being forced to develop as an administrator is not a good habit and is probably what makes some companies nervous enough to disallow any local administrative rights altogether [2].

you don't have to play games with the names of databases or message queues as you can all use the same name

Not sharing services also implies that there are many more copies of that service around, which means that there will likely be different variations as each developer upgrades his or her machine at different times. Whilst 'it works on my machine and the build server' is great for your confidence, you might still end up helping out the one colleague who can't get it working on their machine because of some weird issue related to them having a different version of a product or driver. Of course this kind of problem can be easily mitigated by having some notes on a wiki or a set of scripts that ensure everyone gets to install the same version at each upgrade. This also ensures that those who may be inclined to always get the latest and greatest beta directly from the vendor are also in sync.

Internal services

3rd party products are, or should be, fairly stable. Aside from bug fixes one doesn't tend to switch major versions of a database on a whim so the problems cited above are probably a little over dramatic. Where this problem can start to creep in is with internal (i.e. the business's) services which you'd hope would evolve more rapidly as the company's priorities change. If you work on a team that is responsible for developing a number of small, independently deployable services, then you will have to decide how much coupling you want to take-on to balance detecting API breaks early versus reducing disruption to others though API breaks unrelated to what they are working on.

When the team develops a number of services that are dependent on one another, even if they're independently deployed, there is a temptation to either include them all in the same solution, or try and reuse them for integration and acceptance testing. Doing this in the build pipeline makes perfect sense as you are usually interested in building layers of trust by integrating more services to ensure they still work together. However, I'm not convinced that this set-up is quite so desirable on a developer's workstation.

If you're working on the interface between the two services then you're probably interested in those problems showing up. The majority of the time however is unlikely to be spent that way and other features will should not be directly dependent either as that would be a sign of unnecessarily tight coupling. If those services are not mocked locally within the solution, then, any time you integrate upstream changes for the solution you're working on, you need to update the dependent solutions too, lest you run the chance of an 'impedance mismatch' occurring when running unrelated tests.

Part of the motivation for moving to much smaller services with more well defined contracts is to decouple their evolutions and that has a knock-on effect into the amount of isolation you can afford to take on. This is especially true when you have a build pipeline that can easily verify each service in isolation and then deploy the services together and verify their respective contracts are being honoured by their consumers.

Source folder sandbox

One of the key concepts that the rise in Functional Programming is bringing to the forefront is the notion of side-effects, and that is exactly what makes testing noisier than it should be. Most of the problems mentioned above are the result of side-effects – the output of binaries and test run data that leaks outside the (supposedly transient) test environment. This leaked state then pollutes further test runs for ourselves and potentially others until we can revert back to a known good state.

Whilst isolation at the machine-level generally provides us with our biggest bang-for-buck, it's possible to strive to eliminate even the use of shared local services so that each source folder becomes an independent test environment. Some of the systems I've worked on have been able to have two source folders of two different revisions of the same product be built and tested concurrently. This then easily leads to an ability to run them side-by-side too which has been very useful for investigating unintended differences not picked up by the automated tests.

What makes this finer-grained sandbox achievable is the ability to host more of the dependent services in-process, or if out-of-process, then it should be configured, started and managed by the test suite itself rather than being a classic machine-wide daemon. For databases this could be done using an in-process version like SQL Server Compact Edition or SQLite [3], or an out-of-process mock such as CouchbaseMock [4]. With .Net based web APIs the newer OWIN stack is designed to support hosting multiple APIs in the same process, albeit under different app domains. Even the traditional TIBCO messaging service can be started on-the-fly.

Naturally this implies that for a distributed system you are not running exactly the same versions of the those services, but the point of the exercise is not to replicate the production environment, but to trade-off ease of development with the ability to pick up fundamental bugs caused by simple integration mistakes. It's always easier to debug on your local machine as you have your entire toolbox at your disposal.

Running two codelines [5] side-by-side where there are services that listen on network ports likely won't work out-of-the-box as it's often easier to fix the port numbers to create a more deterministic configuration. However, as was mentioned earlier, it helps if key settings such as these can be easily tweaked, perhaps through the use of environment variables, to make it painless to expand the sandbox as needed.

Epilogue

I once started a new job and was presented with a 50-page document describing what I needed to do to set-up my development machine. Admittedly it covered a few other things as well, but even so the process seemed far too convoluted just to get into the code.

Nowadays I'd expect to install the core language toolset and the version control client and pretty much be done, at least, to get started that is. From this meagre configuration I should then be able to create a working copy of the latest source code from the repo, build it and run the core test suites to verify that I'm all set up. At this point I should now be in a position to start making most normal code changes that could be pushed to production via the standard build pipeline. The only scenarios initially out of my reach would require me to know more about a production-like set-up, but that will come in due course.

It's not always possible to create and develop in the sandbox of your own choosing but where possible I've found it well worth the effort to strive to

What Do People Do All Day?

Thaddaeus Frogley shares his day to day activities as a games programmer.

I work as a programmer for BossAlien, an award-winning game development studio based in Brighton, UK. We are part of the broader NaturalMotion team based in Oxford and largely operate as an independent studio.

I am currently the lead programmer on the client team of *CSR Racing*, a popular drag racing game for mobile phones and tablets. We operate as a live-game team on *CSR Racing*, meaning we work on updates, listen closely to player feedback, add content and features and fix bugs. My time is split between team leadership work, programming and what we refer to as 'data-wrangling'.

The team

Our office of about 60 staff in Brighton maintains a relaxed informal atmosphere and internal work culture.

The programmers have a range of seniority, from fresh graduates all the way through to grizzled veterans, with years of experience distributed in exponential steps.

We have a mostly open plan office on a single floor, with two meeting/conference rooms, teleconferencing facilities, a kitchen and a shower. Some people, myself included, use standing desks. There is music played in the office which draws from a shared playlist, with different parts of the office having separate control over volume. People play musical instruments, nerf guns and board games at work.

The software

The game client is written mostly in C# (221kloc) and is based on Unity3D, a very popular game engine. In addition to C#, we work with Objective C (25kloc) on iOS, and Java (34kloc) on Android. The server side, both player facing APIs, and internal tools are written in PHP and C#, but that is handled by the infrastructure team, so I don't have to worry about it. We also use a number of 3rd party SDKs, libraries and plug ins.

In addition to the code itself, the game also consists of 'prefab' files, which describe much of the visual and logical structure of the game, including the UI, in a proprietary format specific to Unity3D. Most of these are created using the GUI tools that make up the Unity3D editor, but total 1304kloc! Finally, there are the binary assets: audio, graphics, textures, geometry and other non-textual data. In total the working copy is ~7GB, and the built deliverable is close to 300MB per platform.

We use git to manage all the source assets and the build products. We have recently had to take corrective action and reorganise to reduce the size of the full repo, which had ballooned to 35GB. This was done by purging the build products from the main repo. We have started to use git submodules to support reuse and to logically separate other parts of the repo.

The inspiration behind the series...

This article was inspired – you could even say 'requested' – by Chris Oldwood's post on accu-general:

> **A plea – more articles on what kind of programming you do**
>
> Hi All,
>
> Something I've tried to do in my own articles in *C Vu* and *Overload* is to try and give some context to the kind of programming I do, because I know it's different from what many others do. In particular I'm interested to know what kinds of constraints, or lack of them others have to put up with.
>
> For example, I remember an accu-general thread about unit testing and how that might/might not be as easy to apply in the gaming industry. Those in the embedded market have historically avoided C++, when perhaps it's certain features of C++ they couldn't afford. Floating-point maths is apparently a no-no in finance, unless you're doing large volumes of risk calculations and then performance trumps precision (well, in the bits of finance I worked in). How does the inability to patch a video game because it's delivered on a read-only cartridge affect the development process?
>
> I don't know about anybody else in ACCU, but I want to know more about the kinds of stuff other people do. And in particular what makes it different to what I do. I appreciate it's often tricky to know what's different (unconscious incompetence) but in those cases when you have had to make a trade-off – what was it and why? When have you read a blog post or book about some cool technique and then shouted at it because it has no place in your industry/organisation/etc?
>
> I'm sure the editors of our *C Vu* and *Overload* journals would be more than happy to receive more content...

Hopefully, this will become a series.

We use feature and release branches in git. As I write this, there are currently 49 branches on the remote. We also have, independent of git, a system for delivering game assets (art, and configuration files, we aim to be able to control as much as we can via data that we deliver live over S3), which has its own branching system. This means we often have a parallel

THADDAEUS FROGLEY

Thaddaeus started programming on the ZX81 when he was 7 years old, and has been hooked ever since. He has been working in the games industry for over 20 years. On Twitter he is @codemonkey_uk or reach him by email: thad@bossalien.com

The Developer's Sandbox (continued)

remove as much noise as possible from sources outside my control. The more constrained you make your sandbox the more confident you feel about exploring the codebase and the system without constantly looking over your shoulder to see if any of your meddling has disturbed your team mates. ■

References

[1] 'Testing Drives the Need for Flexible Configuration', Chris Oldwood, *Overload* 124
[2] 'Developer Freedom', Chris Oldwood, *C Vu* 26-1
[3] http://www.sqlite.org
[4] https://github.com/couchbase/CouchbaseMock
[5] *SCM Patterns* by Stephen P. Berczuk with Brad Appleton

branch structure. Our builds – asset system – exists in three stacks: Dev, QA, and Release. These various orthogonal configuration axis provide us with a lot of power and flexibility, but can be quite confusing for the uninitiated.

We avoid code ownership and try to share knowledge about systems around the team, using a mixture of code review, mentorship, regularly changing areas of responsibility and documenting work flows. Code is expected to be self-explanatory first and well commented second. We don't have a fixed coding style or standard, instead using the convention that your code should be consistent with the code around it.

The builds for QA and for Release are produced by a build machine, using Pulse, and a set of custom build scripts. A change (code or art) can be tested in the editor in seconds. Certain asset changes need to be 'bundled', which can take a minute or so. A development build can be created and deployed to a device in the time it takes to make a cup of tea. Iteration times are very important and anything that slows the team down is considered a high priority problem.

The hardware

Most of the programmers have Apple laptops with external monitor(s), keyboard, mice and headphones. Specific individuals may have a PC or Windows laptop instead or as well depending on what their work requires. Most of the build machines are Mac Minis or retired laptops. People can take their laptops home and can work remotely if required. We have a single rack of servers for internal services, file shares, routers, etc., using AWS for the player facing servers. There is a pool of iOS and Android devices of for testing development builds on.

The process

We have to juggle a surprising number of challenges. Players in *CSR Racing* are always looking for new content and new things to do, so we are consistently adding new features based on demand. It's critical for us to listen to and respond to player feedback. There are also new platform updates to be aware of, so we are always working hard to keep SDKs for integrations with external services up to date. We also have to be mindful to harden our system security as hackers sometimes look to cheat and exploit the game. As a competitive game, it's critical that we maintain a consistent level of fairness.

We have a release and high level features plan, which is discussed regularly. The time line for this is maintained centrally across teams and is frequently updated and referred to, with notes about staff vacancies and other important calendar events. We closely monitor progress and shift resources to where they are needed in order to keep our commitments. We regularly discuss capacity and strive not to over commit. Overtime is rarely requested, and is never obligatory. Individual tasks and bug reports are kept track of using JIRA. Work is organised into releases of two types: Code Drops, and Data Pushes. With code drops we are gated by the platform holders and have roughly a 3-week latency from a Release Candidate to the update going live on App Store(s). A Data Push, on the other hand, contains no code (though some configuration files come close to a complexity that could be described as programming!), but game data that is pulled by clients with an internet connection from our S3 file store. This doesn't go through a publisher approval process and can be subject to change as little as hours before it is published to players.

Whatever is pending release, our QA team is also constantly working on testing features in various stages of development. Email is an important tool for our asynchronous communications within the team, but we also have lots of face-to-face time, including regularly scheduled one-on-one chats.

We are able to release different code and data for each of the platforms we support, but we strive for feature parity and same day releases on all platforms.

Every day is different, but for me a typical day might start with reviewing the previous day's commits, and setting up any relevant code reviews. I'll likely then check my email, respond to questions about the project and do some code reviews. I'll check JIRA and possibly triage any new bugs with the QA lead, and the project manager. If it's a Tuesday or a Thursday, we'll have a core team stand up meeting. I am responsible for the overall technical health of the project, so I have to play it by ear, identifying things that need doing, and either doing them, or making sure the right person does. This might be debugging and bug fixing, implementing new code, or helping plan code work. It could be auditing the branches in git and making sure the features get merged in where they are needed. Sometimes it's testing, sometimes it's setting up data to implement new features using data driven systems. We have kick-off meetings, and post-mortems for bigger chunks of work, so we can reflect on and improve our processes actively. Our general philosophy is to let people use the tools they prefer. I use Sublime, and TextWrangler. Some people use Xameran, others use MonoDevelop. The technology stack we use means debugging often has to be done via logging, as remote debugging the C# using MonoDevelop has stability problems. In addition to using git on the command line, we use SourceTree.

Recruitment is an important part of the company strategy, so group CV review is a regular feature of office life. Applicants who pass the CV screen are invited to do a remote code test, the results of which are discussed by the whole code team. The ones that pass that filter are invited to come in for a face to face interview.

The future

CSR Racing continues to be one of the world's most popular racing games on mobile and tablet.

Our goal is to continue to delight players – both those new to *CSR Racing* and veterans.

While we do that, we try to improve the tools, workflows and the code itself so that all this can be done at the same rate, but with fewer people, freeing up more of the team to help with the development the company's next game(s).

End

If you'd like to hear more about the development of *CSR Racing*, I will be talking about it on Wednesday the 22nd of April, at the ACCU 2015 Conference in Bristol. ■

Simple Android programming with WebKit

Silas S. Brown shares his trials with developing for mobile devices.

I don't like modern smartphones. It seems the manufacturers are trying to trick people into thinking that the lack of real buttons is some kind of progress, as a lame excuse for their having become too lazy to make keyboards. Nowadays if you want a keyboard you are expected to buy it as a separate add-on and perform a juggling act on the streets.

I'm still waiting for the great keyboard revival, but meanwhile other people are asking me to develop for touch devices and I reluctantly bought a Sony Xperia Z Ultra for a project. Much to my annoyance, within one month of purchase it started registering false screen-presses by detecting moisture from my breath: they obviously didn't realise people with limited eyesight might hold it close. And needless to say I can hardly type on the thing (an application called 'Hacker's Keyboard' with its good Dvorak layout helped a bit, but it's still not as good as a real keyboard); the pocket Bluetooth keyboard I ordered never turned up.

Anyway, the canonical way to program these things is to download the Android Developer Tools (ADT) [1]. The Eclipse-based ADT is a bit – how should I put this? – 'wobbly'. Not every version installs OK. One version I downloaded ended up not being able to create new projects (well, it could create new projects, but the empty project wouldn't build, although it could still open projects created with an earlier version of ADT). Sometimes when using ADT you have to wait for it to finish its background operations before it'll do what you want without errors, because the code that's supposed to do this automatically doesn't always work. Sometimes there are other timing-related bugs, so you generally have to act nice and slowly as if not to confuse the poor computer. Sometimes you have to press F5 on the Package Explorer to force a refresh, although that's supposed to happen by itself when necessary but it doesn't always work. And every version of the ADT is different, so if I tell you what to do in one version then the instructions likely won't work in another. Bring back the command line!

If you do manage to create a project that builds, I'd suggest the first thing you should do is to create a WebView and write some HTML for it. That way you can get an application up and running quite quickly. The HTML can include Javascript that calls back into the Java code (and, as it turns out, C code: more on this later), or you could just write a Javascript-based Web application and use ADT to package it into an 'app' as a convenience for offline use (but please don't write an app that just browses an online site with no additional functionality: apps like that are generally annoying, as users tend to think 'I can browse the web myself thank you very much'; to justify a dedicated app, it either has to include its own offline content or else apply some kind of offline processing to the pages it retrieves so it's not just a poor imitation of the general Web browser).

The project's `res/layout/activity_main.xml` file should look like Listing 1, and the code itself should be something like Listing 2. Then you can place the HTML into the `assets` folder, the main file being `index.html`: I would suggest making it 'mobile friendly' as in Listing 3. Then refresh the ADT (highlight the Package Explorer and press F5), wait, and try it out with Run / Run As / Android application. ADT will use a real device if connected, otherwise it will start an emulator, and either way, if it works, you'll be left with an APK file in the `bin` directory which you can distribute to others if they have 'Unknown sources' enabled in their 'Application settings' or 'Security'.

Listing 1

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_height="fill_parent"
   android:layout_width="fill_parent"
   android:orientation="vertical">
  <TextView
    android:layout_height="wrap_content"
    android:layout_width="fill_parent" />
  <WebView
    android:id="@+id/browser"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent" />
</LinearLayout>
```

Listing 2

```
import android.webkit.WebView;
import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity {
  WebView browser;
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    browser = (WebView)findViewById(R.id.browser);
    browser.getSettings().setJavaScriptEnabled(true);
    int size=Math.round(16*getResources().getConfiguration()
                .fontScale);
    browser.getSettings().setDefaultFontSize(size);
    browser.getSettings().setDefaultFixedFontSize(size);
    browser.getSettings().setDefaultTextEncodingName("utf-8");
    browser.loadUrl("file:///android_asset/index.html");
  }
}
```

Listing 3

```
<html>
  <head>
    <meta name="mobileoptimized" content="0">
    <meta name="viewport" content="width=device-width">
  </head>
  <body>
    You should not have to zoom in to see this.
  </body>
</html>
```

SILAS S. BROWN

Silas S. Brown is a partially-sighted Computer Science post-doc in Cambridge who currently works in part-time assistant tuition. He has been an ACCU member since 1994 and can be contacted at ssb22@cam.ac.uk

Listing 4

```
class MyObject {
  public MyObject() {}
  @android.webkit.JavascriptInterface
  public String test(String in) {
    return "You called test with "+in;
  }
}
browser.addJavascriptInterface(new MyObject(),
                               "myObject");
```

Notice Listing 2 has three 'size' lines to set the font size according to the system's font size setting. I'm putting this in 'right off the bat' because I believe getting it right is important (some people need big print, and, if you don't, you probably will when you're older). It's a pity Google didn't make this the default behaviour with their WebView component.

There are quite a few things missing from our application: the Back button won't work, 'pinch to zoom' won't work, Javascript alerts won't work, local storage won't work, the page state is reset when you rotate your device, and there is so far no way to specify that certain links should open in the 'real' browser rather than our app. All of these can be fixed by adding more code to 'switch things on', and it gets tedious, so if you're interested I'll refer you to my 'html2apk' code [2], which addresses all of these issues and also adds a Javascript object for clipboard interaction.

Adding Java and C functionality

You can provide extra Javascript objects implemented in Java by writing something like Listing 4. Then you can call these from the Javascript in your web pages, e.g. **`document.write(myObject.test('hi'));`**

To write in C, you'll additionally need to download the Android NDK (Native Development Kit?) [3]. You'll then need a JNI directory, containing an `Android.mk` file like Listing 5 and an `Application.mk` file like this:

```
APP_PLATFORM := android-1
APP_ABI := armeabi
```

Then your `my_program.c` should start with **`#include <jni.h>`** and should incorporate functions like Listing 6. Then back in `MainActivity.java` you'll need something like this to make your C function available to Java:

```
static { System.loadLibrary("MyProgram"); }
static synchronized native String jniMyFunc
  (String in);
```

After that, you can make it available to your Javascript by changing Listing 4 to call your newly-available C function. I'd suggest not bothering with C on Android unless you have either a lot of existing C code or a dire need for speed: Java or even JavaScript is enough for most simple purposes and involves less setup on this platform.

Listing 5

```
LOCAL_PATH:= $(call my-dir)
LOCAL_SRC_FILES := my_program.c
LOCAL_MODULE := MyProgram
LOCAL_MODULE_FILENAME := MyProgram
include $(BUILD_SHARED_LIBRARY)
```

Publishing

When you're ready to ship your 'app' (I tend to use the full word 'application', but these 4-syllable words are hard I know) on the so-called 'Play Store' (are we all children now? are all Android versions named after sweets as a childish rebellion against some other company using healthy fruit?), you'll have to use File / Export / Export Android Application (it lets you create a keystore and private signing key), then pay for your Play Store account (I really think it should be free to upload if you're doing the platform a favour by contributing a completely free and no-adverts app for the benefit of the community, but at least the payment is only a one-off; Apple have a higher payment and they make you renew every year, which is why they won't be getting any iOS apps off me anytime soon), and finally upload the APK and wait for it to be published on the Store. You could just skip all this and have your users download the APK file from your own site with 'Unknown Sources' enabled, and you might wish to do it this way for a private app that you don't want the entire public to see, but having your app on the Play Store is a significant convenience for many users.

When uploading future versions, you'll need to increase the version number in the `AndroidManifest.xml` file (otherwise Play Store won't accept it), and I'd suggest mirroring that change in the second copy of AndroidManifest that's in the `bin` directory (the mirroring is supposed to be automatic, but due to general 'wobbliness' it seems that doesn't always happen). I'd also suggest using the Play Store 'beta test' facility just to double-check your app did not somehow get corrupted during the upload process as one of mine did. I don't see how, as they're cryptographically signed, but somehow an app which worked perfectly well for me didn't work after my friend had uploaded it to his account on the Store: the Javascript-to-Java callback wouldn't run. By that time we had over 1000 users, most of whom had automatic updates enabled on their devices, so I got several phone calls, making me feel like Star Trek's engineer Scottie having to fix the engines quickly before the ship explodes, and all I did was to increase the version number one more time and ask my friend to re-upload, after which the problem mysteriously disappeared. But not before several users had turned off automatic updates on their devices (which is a worry: I hope their devices will still get the security-critical updates). After that we decided, no matter how well an APK file works for us, we'll always use the Play Store's own Beta Test facility to double-check nothing bad somehow happened during the upload process. I still don't know what happened (do Google servers introduce bit-flips sometimes?) – everything about this platform seems to be 'wobbly'. Please don't run anything safety critical on it. ■

References:

[1] http://developer.android.com/sdk/
[2] http://people.ds.cam.ac.uk/ssb22/gradint/html2apk.html
[3] https://developer.android.com/tools/sdk/ndk/

Listing 6

```
JNIEXPORT jstring JNICALL
Java_your_package_name_here_MainActivity_jniMyFunc
  (JNIEnv *env, jclass theClass, jstring jIn) {
    char *i=(char*)(*env)->GetStringUTFChars(env,jIn,NULL);
    char *o=malloc(strlen(s) + 20);
    // TODO: check o != NULL
    strcpy(o, "You wrote ");
    strcat(o, i);
    (*env)->ReleaseStringUTFChars(env,jIn,startPtr);
    jstring ret=(*env)->NewStringUTF(env,o);
    free(o); return ret;
}
```

Raspberry Pi Linux User Mode GPIO in Python

Ralph McArdell finds stream based input/output to be more convenient.

In this article I am going to discuss an alternative approach to using GPIO on a Raspberry Pi in Python to that taken by an early version the RPi.GPIO package [1]. The approach occurred to me in 2012 while making a start at hardware interfacing and programming using a Raspberry Pi with some LEDs, switches and the like that had been collecting dust for 20 to 30 years. To read and write data to the GPIO lines, I thought I would start with Python and the RPi.GPIO package.

After playing with RPi.GPIO it became clear that it had a few shortcomings, most obvious of which was that there was no support for waiting on a change in GPIO input pin value. You had to poll input pins to read their value and could not wait for input pin state change notification events. I knew the sys filesystem GPIO interface supported this as I had read the Linux kernel documentation on GPIO [2], and even the GPIO sys file system supporting source code used by the Raspberry Pi's Linux kernel [3].

There were other ways that RPi.GPIO went about things that seemed a bit awkward to me, such as the fact that each read or write needed to open, write to and close various files in addition to the read or write associated with the GPIO operation. Another was the pin identification number mode used to indicate whether pin numbers represented raw GPIO pin numbers or Raspberry Pi P1 connector pin numbers. It all worked but seemed a bit clunky and inefficient. As RPi.GPIO was at an early stage of development, I would not be surprised if more recent versions have addressed many – if not all – of my concerns.

Putting my research into the Linux user space sys filesystem GPIO support to good use, I had hacked together some proof-of-concept Python code that allowed waiting on input pin edge events. The additional I/O for each pin read or write I thought could be reduced by having some concept of an open pin – which implies a matching concept of a closed pin. Once I started thinking in terms like open and close, the idea to look at Python's stream I/O for possible concordance popped into my head (I should note that I am not a full time Pythonista). After some poking around I found the open and I/O documentation pages for Python 3 [4, 5].

This led me to become side tracked from other Raspberry Pi I/O interfacing I had intended to look into to investigate further whether a similar model could be used for GPIO, that is: an open function returns an initialised object for the I/O operations requested that is associated with a GPIO pin. Like the Python 3 stream I/O implementation the exact type of the object returned would depend on the requested I/O mode for a pin. It also occurred to me that the abstraction could be extended to cover groups of pins.

The results of my investigation are freely available on Github [6] as the Python 2.7 package dibase.rpi.gpio. It currently is not a fully packaged Python module and I have only updated the pin id support to cater for the Raspberry Pi revision 2.0 boards' GPIO pin layout and not the newer B+, A+ and compute module variants. There is documentation available on the Github repository's wiki [7], which explains how to use the package. In the rest of this article I would like to go into some of the design decisions and implementation details.

The big picture

As I have mentioned, I was basing the overall 'shape' of the package on that of the Python 3 stream I/O. In addition to an **open** function and the types for the objects returned by **open**, the Python 3 stream I/O defines some abstract base classes, some of which provide default implementations for some methods, which as a long time C++ practitioner seemed perfectly reasonable to me.

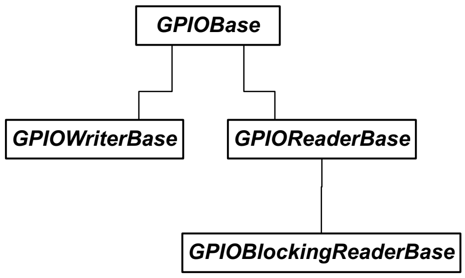


Figure 1

With stream I/O, we use a pathname string to identify the item to be opened. For GPIO, pins are identified by a value that is in a (small) subset of the positive integers. It seemed to me some concept of a pin id would be useful to validate such values and to support both Raspberry Pi P1 pin values (and maybe names) and the underlying chip pin id values.

One of my pet hates in code are magic values – numbers definitely and often strings as well – and the GPIO sys filesystem interface has many magic strings to build pathnames and for special values written to files. So some way to abstract and centralise such magic knowledge seemed appropriate.

As with most code modules there would almost certainly be a set of errors, in this case most likely in the form of exceptions that would be required.

So to recap the package would require:

- an **open** function – in fact I ended up with two: one for single pins and one for groups of pins
- GPIO abstract base classes
- a set of concrete classes fully implementing the ABCs to handle the GPIO operations
- pin id value, validation and mapping abstractions
- abstractions for magic knowledge such as pathnames
- exception classes

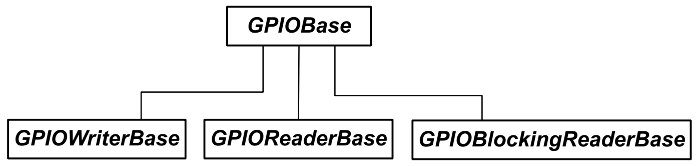
What bases?

The set of abstract base classes and the operations they support would need to be different from the stream I/O case. While there would be a case for a root **GPIO** abstract base class similar to the Python 3 **IOBase** type that declared operations common for all GPIO modes, the set of those operations would differ. In particular it seemed a good idea to separate out the read and write operations into separate sub-abstract base classes as bidirectional I/O is not supported by the BCM2835 chip used by the Raspberry Pi – that is a GPIO pin is either input or output but not both. A further distinction was whether read operations block on-edge events or not.

My first thought, depicted in Figure 1, was that **GPIO** blocking reader types would be a sub-type of (non-blocking) **GPIO** reader types.

RALPH MCARDELL
Ralph McArdell has been programming for more than 30 years with around 20 spent as a freelance developer predominantly in C++. He does not ever want or expect to stop learning or improving his skills.

Figure 2



But when it came time to actually testing and implementing a blocking read type it became apparent that the read operation would benefit from a timeout parameter which, obviously, is not required in the polling only reader case, so the actual GPIO abstract base class hierarchy ended up as shown in Figure 2.

For the **GPIOBase** class, I started by cherry-picking obviously useful methods from **IOBase**: **close**, **closed**, **readable** and **writable**.

To distinguish blocking readers from non-blocking readers I added a query method in the style of **readable** and **writable**: **blocking**.

As a GPIO pin was represented in the sys filesystem by a specific file path that would be open so as to be written to or read from, there would be a file descriptor that could be said to be associated with each pin – so I also initially included a **fileno** method. However, later on I realised this did not scale to the pin group case, as a group of pins would be associated with a group of file descriptors so the **fileno** method of **GPIOBase** was replaced by a method called **file_descriptors** instead which returns a list of file descriptors. The **fileno** method was then defined for each single GPIO pin implementation class in terms of the **file_descriptors** method.

The **readable**, **writable** and **blocking** query methods are defined by the **GPIOWriterBase**, **GPIOReaderBase** and **GPIOBlockingReaderBase** to return **True** or **False** as appropriate.

The **GPIOWriterBase**, **GPIOReaderBase** and **GPIOBlockingReaderBase** base classes also add the appropriate write or read operation. **write** takes the value to be written and **read** returns the value read; **GPIOBlockingReaderBase**'s **read** also takes a timeout parameter. Implementations can take advantage of the Python type system so that the values read or written can be either single or multiple GPIO pin values in various forms allowing single pin and group of pins implementations to use the same interfaces.

What did you want to open?

With stream I/O, what is to be opened is specified as a pathname string. With GPIO, a single or group of GPIO pins would be opened and GPIO pins are identified by a small positive integer specific to the chip(s) in question. These numbers, formatted as strings, are used in the GPIO sys file interface. The raw values for the Broadcom BCM2835 chip used in the Raspberry Pi have values in the range [0, 53]. However only a subset of these are wired up for user use on the Raspberry Pi board via the P1 header connector, and the unpopulated P5 header added to revision 2.0 boards. Most of these GPIO pins have alternative functions other than simple GPIO as detailed in the Broadcom BCM2835 ARM peripherals document [8].

I wanted to be able to refer to pins in various ways: raw chip GPIO pin number, Raspberry Pi P1 connector pin number and, preferably, by a name indicating the function of the pin, as used by the Raspberry Pi circuit diagram [9] and elsewhere. Additionally it would be nice to prevent inadvertently specifying a raw GPIO pin number which was not available via the P1 connector, but not prevent specifying such pins altogether. At the time there was only one revision of the Raspberry Pi circuit boards so there was no P5 and no complications as to which chip GPIO pins were brought out to which P1 pins. Sometime after the initial implementation I got around to extending the pinid module to support Raspberry Pi revision 2 board GPIO [10] – including P5 – with mapping and validation performed with respect to the revision of the Raspberry Pi board in use.

The design I went with has a primary class called **PinId**, which is supported by various validation and mapping classes. **PinId** extends **int** and provides class methods to create pin ids from integer values that represent either raw GPIO pin numbers (**gpio**, **any_chip_gpio**) or Raspberry Pi P1 and P5 connector pin numbers (**p1_pin**, **p5_pin**). In all cases, a **PinId** object is returned if the passed value is valid; otherwise, an exception is thrown. The value of this object is an **int** in the range [0, 53] representing a BCM2835 GPIO pin value. The validations and possible mappings performed vary for each of these factory methods: **gpio** and **any_chip_gpio** do no mapping but check the value is in the valid set of pins: [0, 53] for **any_chip_gpio**, and only the GPIO pin numbers of those pins that are connected to the P1 or P5 connector for **gpio**. **p1_pin** and **p5_pin** first map the passed pin value from a P1 or P5 pin number to the BCM2835 GPIO pin it connects to, if any, and if good returns the value from **gpio** for this mapped pin number.

In addition to the four factory methods, **PinId** also provides class methods named for each Raspberry Pi P1 and P5 connector GPIO pin, each having the form **pN_xxx**, where **N** is 1 or 5 and **xxx** is the function name of the pin in lowercase – **p1_gpio_gclk** for example. Each of these methods takes no parameters (other than the class) and calls the **p1_pin** or **p5_pin** factory method with the P1 or P5 pin number having that function. Note that while the functions for P1 pin names should never currently fail as in both current boards revisions the same set of pins connect to GPIO pins, the functions for P5 pin will fail and raise an exception if called while running on a revision 1 board that has no P5 connector.

prevent inadvertently specifying a raw GPIO pin number which was not available via the P1 connector

Originally the pinid module was totally self-contained with all the data for mapping and validation of integer pin values defined within the module. Annoyingly having to be hardware-revision aware blows that out of the water. Now the major revision as provided by the **dibase.rpi.hwinfo** module is used to switch the sets of P1 and P5 **connector-pin-to-GPIO-pin** maps to those relevant for the board revision in use. In order to do this the `/proc/cpuinfo` pseudo file has to be interrogated – although once read the data is cached.

Do what, how?

The Python stream I/O **open** function – similar to the C library **fopen** function – takes a second parameter specifying the so called mode in which to open the item specified by the file argument. The GPIO **open** functions, **open_pin** and **open_pingroup**, follow this model. There seemed no immediate need for further parameters such as the Python **open** function's buffering parameter.

Like the stream I/O **open** function, the mode parameters of **open_pin** and **open_pingroup** are expected to be short strings. From **open** I kept the **r** and **w** characters for read (input) and write (output) modes – although only one can be specified – and added an optional second character that indicates the blocking mode for pin reads – the choices are **N**: none, **R**: block only until a rising edge transition from low to high, **F**: block only until a falling edge transition from high to low and **B** (for 'both'): block until either change transition. The only valid blocking mode for pins opened for writing is **none**. If the second blocking mode character is not given then it defaults to none. If no characters are given (i.e. an empty string is passed) then the default of non-blocking read mode is used (i.e. it is a synonym for **rN**); this is the default if no mode argument is given.

When I came to implement pin groups it occurred to me that such groups can be given either as a set of bits in an integer or as a set of Boolean values in a sequence and this added two additional mode characters **I**: multiplex pin group values into the lower bits of an integer and **S**: pin values are discrete Boolean values in a sequence. These are notionally the third character in the mode string but can be the second, in which case the blocking mode defaults to none. Note that the format of the data representing a pin group is set when opening a group of pins – as each variant is handled by a separate class.

Open unto me...

The **open_pin** and **open_pingroup** functions parse the mode string to determine the specific type of object to create and return. The returned object will conform to one of the three abstract base class types **GPIOWriterBase**, **GPIOReaderBase** or **GPIOBlockingReaderBase**; however, the types returned from **read** and passed to **write** will differ. **open_pin**, for single pins, always traffics in Boolean values and, when writing, values that can be converted to Boolean via the usual Python conversion rules – with the exception that a string **0** (a zero character) converts to **False** – i.e. a low pin state. **open_pingroup**, on the other hand, traffics either in integer values or in sequences of Boolean values – which when writing each value can be a value convertible to Boolean, with **0** values converting to **False**.

a pin could have its exported state change between the check and actually trying to export the pin

As implementation proceeded it became obvious that the various modes – direction, blocking and (for pin groups only) data format – had various related aspects. For example direction mode was specified, and needed checking, as a character in **open_pin** and **open_pingroup** mode parameter strings; likewise, a given direction mode dictates the string written to a GPIO pin's sys file system direction controlling file. It seemed natural then to group all of these concerns into a class. Hence there are supporting types **DirectionMode**, **BlockMode**, and (for pin groups only) **FormatMode** that handle the various aspects of each of the modes.

How common!

As mentioned in the pre-amble, one of the things I wanted to achieve was to reduce the per-IO call overhead by moving certain repeated operations to a once-performed open and, thereby, the inverse operations to a once-performed close. In fact the open logic ends up in the **__init__** methods of the **GPIOWriterBase**, **GPIOReaderBase** or **GPIOBlockingReaderBase** implementation classes. In order to be good citizens, a pin or pin group should close itself when destroyed and support the Python with statement that controls context management – similar to using in C# or RAII in C++. All this adds quite a lot of boiler plate to a class' implementation.

Luckily it turned out that much of this code was common to either the pin or pin group implementations of **GPIOWriterBase**, **GPIOReaderBase** and **GPIOBlockingReaderBase** and so could be pushed up into a common base class: **_PinIOBase** for pin implementations and **_PinGroupIOBase** for pin group implementations. This reduced the specific implementation classes to having to usually only implement (an often minimal) **__init_** method and the write or read I/O method. The exception being the **PinBlockingReader** class which annoyingly needed to add extra validation in the middle of the otherwise common base **__init__** processing flow. This was achieved using the template function pattern and having the base **__init__** call out to an overridable method called **cb_validate_init_parameters** which **PinBlockingReader** overrode to inject its additional logic while the other two pin implementation GPIO classes relied on the default base behaviour.

Some nitty gritty details...

So what exactly did I change in the flow of calls to the sys file system, from that performed by the early RPi.GPIO package?

First you have to understand how you get access to a GPIO pin via the sys file system interface.

A pin has first to be exported by writing its chip GPIO number ([0, 53] in this case) to an export file. On doing this the driver creates and populates a pseudo directory for that pin, if it is available – meaning not already exported. When done with a pin it is unexported by writing the GPIO number to an unexport file, which removes the directory created by writing to export.

Once exported, the desired direction and – in my package's case – edge modes are set by writing specific values to specific (pseudo) files in the exported pin's directory and the file to which values are read or written to opened as appropriate. Then the file is closed and the pin unexported.

In the early RPi.GPIO package I played with each read or write (input or output) from/to a pin, went through the whole process of unexporting if exported, (re-)exporting the pin, setting up the direction (another file open, write and close), writing or reading the value and unexporting the pin.

In my model I export the pin, set the direction and edge modes and open the value file for the pin, during object creation via **open_pin**. The read/write operations do not need to repeat those steps and just get on with the reading or writing to the open value file. Then on close the value file is closed and the pin unexported.

I chose a policy that it was an error if a pin were already exported when attempting to open it because it could be exported and used by a separate process in a different way (for output instead of input say). The code therefore checks to see if such a directory exists before trying to export a pin and raises a **PinInUseError** if it is. Note that this is not 100% foolproof as a pin could have its exported state change between the check and actually trying to export the pin. However, in most use cases one would expect GPIO pin use to be fairly static and so such a situation should only occur by accident when initially setting up pin / process assignments or during revamps of such assignments.

This policy does have one ramification though: a pin *must* be unexported when it is finished with otherwise the next execution of the program using the pin will fail as it will still be exported and therefore deemed to be in use. Hence the effort to ensure a pin is closed correctly: by calling **close** explicitly, when an object is destroyed and on exit from a with statement block that is controlling such an object.

However, such a situation may arise in which a pin is left exported (especially during development and testing!) so I added a function called **force_free_pin**, which will unexport a specified pin if it is exported and return a Boolean indicator as to whether it did so. However, I found that using pin and pin group I/O objects controlled by a **with** statement to be an effective way of preventing pretty much all such mishaps (I suppose it is possible that the process terminates in a way that is outside the control of the Python runtime which would presumably defeat any clean-up).

As to the matter of reading and writing high and low values from or to a pin, well non-blocking reads and writes are a matter of reading or writing '1' or '0' from/to the zeroth position of the open value file. Blocking reads involve using the **select** system function – which is presented as the Python **select.select** function – to wait on the value file or timeout expiration and then – if not a timeout – performing a read as per the non-blocking case.

For pin groups, there are six classes implementing pin groups – three trafficking in pin values multiplexed into the lower bits of an integer and three trafficking in sequences of Boolean values. They rely on the single pin types by calling **pin_open** for each pin in the group passing in the same mode for each pin. Closing a pin group just iterates through the sequence of open pin objects and calls close on them. As for single pin GPIO objects, **__del__** and **__exit__** call close to ensure all pins in a group are closed.

Pin group non-blocking reads are simply a matter of reading each pin and composing the correct type of composite value. Pin group writers cache the current value and use it to only write to pins whose state has changed. The two pin group blocking reader types pass all the group's pin objects

Kevlin Henney: An Interview

Emyr Williams continues the series of interviews with people from the world of programming.

For regular attendees of the ACCU Annual conference, Kevlin doesn't really need an introduction. He is a well-known author, engaging presenter, and a consultant on software development. He was the editor for the book *97 Things Every Programmer Should Know*, and has given keynote addresses not just at ACCU but at other conferences as well.

How did you get in to computer programming? Was it a sudden interest? Or was it a slow process?

I was aware that computers could be programmed, and the idea sounded interesting, but it wasn't until I was able to actually lay hands on a computer that I think it occurred to me that this was a thing that I could do myself.

What was the first program you ever wrote? And what language was it written in? Also is it possible to provide a code sample of that language?

I can't remember exactly, but I suspect it probably just printed "Hello" once. I strongly suspect that my second program printed "Hello" endlessly – or at least until you hit Ctrl-C. It was written in BASIC, and I strongly suspect that it was on a UK-101, a kit-based 6502 computer.

EMYR WILLIAMS

Emyr Williams is a C++ developer who is on a mission to become a better programmer. His blog can be found at www.becomingbetter.co.uk

Raspberry Pi Linux User Mode GPIO in Python (continued)

(which support a `fileno` method) to `select.select` and so wait on a change to any of the group's pins. They also use a cached value and only update those pins values that were indicated as changed from the information returned from `select.select`.

Odds and sods

As with any set of code, as you go along error conditions pop up and to cater to those in the GPIO package I created a module – `gpioerror` – for GPIO specific errors. There is a top level exception called `GPIOError` which sub-classes the Python Exception type from which all the other GPIO specific exceptions derive - although some do not do so directly but sub-class other GPIO exceptions. Currently each exception class simply defines a static error string as its doc comment and passes its doc comment to its super class in its `__init__` method.

I have mentioned that the sys file system for GPIO relies on specific pathnames and values to be written to files. Special values written to files are generally specified by the pin module types that group the related aspects of direction and blocking modes. The sys file system pathname magic strings and knowledge of how to create specific pathnames form a set of functions in a sysfspaths module – some of these are nullary functions (take no arguments) that return static path fragments – the `gpio_path` function returns the base absolute path to GPIO support in the sys file system for example. Other function take a pin id value and use this to create and return pin specific paths – the `direction_path` function returns the path to the direction (pseudo) file for a given pin id number for example.

Testing, testing...

In case you were wondering, yes, there are tests and yes, they were developed in step with the code they tested. There are two sets of tests – unit tests which, as per usual, do not rely on any modules other than that under test and what I called at the time system tests – which in this case meant tests that had to be executed on a specific system – a Raspberry Pi running Raspbian or compatible Linux system in this case. In the time since I named them I have thought of an alternative name that I think is less ambiguous: platform tests – tests that need to be executed on a specific platform. Some of the so-called system tests require user interaction – either to create input (e.g. toggle switches) or observe output (e.g. check on the lit state of an LED). Since creating such tests I have tended, in other projects, to split out tests which require user interaction and call them interactive tests. Note that platform and interactive tests are probably sub-types of integration tests, although I suppose it could be argued that some of them tend towards the usual meaning of system tests. ■

References

[1] RPi.GPIO: http://pypi.python.org/pypi/RPi.GPIO
[2] Documentation/gpio.txt in the Linux kernel tree, as located at: https://github.com/raspberrypi/linux/blob/rpi-3.2.27/Documentation/gpio.txt
[3] As far as I could tell this was the generic support provided by gpiolib: https://github.com/raspberrypi/linux/blob/rpi-3.2.27/drivers/gpio/gpiolib.c
[4] http://docs.python.org/3.3/library/functions.html#open
[5] http://docs.python.org/3.3/library/io.html
[6] https://github.com/ralph-mcardell/dibase-rpi-python
[7] https://github.com/ralph-mcardell/dibase-rpi-python/wiki
[8] http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf
[9] http://www.raspberrypi.org/wp-content/uploads/2012/10/Raspberry-Pi-R2.0-Schematics-Issue2.2_027.pdf
[10] http://www.raspberrypi.org/archives/1929

while (you care about code)
{
read (cvu && overload);
}
do(it);
because good code matters
ACCU
WWW.ACCU.ORG
PROFESSIONALISM IN PROGRAMMING

These days I am more likely to disavow any knowledge of BASIC than I am to provide code samples in it – but I think you can probably guess what those examples I just mentioned would look like!

What would you say is the best piece of software you've ever written? The one you're most proud of?

Difficult to say. Possibly the proof-of-concept C++ unit-testing framework I came up with a couple of years ago, that I dubbed LHR. I don't know if it's necessarily the best, but it incorporated some novel ideas I'm proud of.

What would you say is the best piece of advice you've ever been given as a programmer?

To understand that software development concerns the management of complexity.

If you were to go back in time and meet yourself when you were starting out as a programmer, what would you tell yourself?

As a professional programmer? Don't worry, it's not all crap. As a schoolboy? Yes, it really can be as much fun as you think it is..

Do you currently have a mentor? And if so, what would you say is the best piece of advice you've been given by them?

I don't currently have anyone I would consider a mentor, but there are a number of people I make a point of shutting up and listening to when they have something to say.

You are well known for giving excellent talks on various topics to do with Software Engineering, I recall the one you did at ACCU Conference last year. How did that come about? And how scary was it to leave the security of a regular 9 to 5 job and go solo?

I worked as a principal technologist at QA, a training and consultancy company, for a few years. Training was part of my job role and that gets you comfortable with presenting and thinking on your feet. Conference presentations are a little different as the objective of a talk and the environment of a conference are not the same as a course or a workshop, but there's enough overlap that practice at one supports practice in the other.

As a principal technologist at QA I enjoyed a great deal of autonomy and so the transition to working for myself was not as jarring as it might first appear. Meeting people at conferences also opened more opportunities than I had perhaps realised were available when I was associated with a larger company.

I'm not sure I could have gone straight from working for someone to being independent. Actually, that's not quite true: I went from being an employee to being a contractor many years ago, but I didn't find that fulfilling.

And following on from that, what advice would you give to someone who's looking to go it alone?

Make sure you know what your motivation is for going it alone, that your expectations are realistic and that you have some work lined up!

I'm guessing you work from home, if so, how do you keep the balance between work time and family time?

A question I've wrestled with for years and still not one I'm sure I have a good answer to! I am, however, far better at turning off than I used to be, recognising that work time is an interruption from family time and not the other way around. As I travel a lot the work–family distinction is often reinforced by whether I'm at home or away, so I try to get more work-related things done when I'm away because it doesn't distract from family. I notice that when I'm working and at home the context switch can be harder because the context is effectively the same.

How do you keep your skills up to date? Do you get a chance to do some personal development at work?

I attend conferences, I talk to people I meet (and people I don't meet) and I read. I probably get a lot more breadth than depth, but I temper that by focusing on things that interest me – so I'll freely admit to being more driven by interest than necessity.

I've seen that you contribute to the Boost libraries as well. How did you get involved in that? And what advice would you give to a prospective developer looking to get involved in such a project? Or any open source project for that matter.

My involvement came about primarily because of my involvement in the C++ standards committee and writing articles about C++. That said, although I have a continued interest in Boost, I am no longer an active contributor, having long ago passed maintenance of my contributions to others.

As for advice on doing it: if you think you want to get involved, then you should. It's worth spending your time familiarising yourself with the ins and outs and mores of your project of interest, asking questions, getting a feel for what you can best contribute and how. If you're a developer, don't assume it's going to be coding where you stand to learn or contribute the most – maybe it's code, maybe it's tests, maybe it's documentation, maybe it's something else.

What would you describe as the biggest "ah ha" moment or surprise you've come across when you're chasing down a bug?

That good practice I ignored? I shouldn't have ignored it. I don't know if that's the biggest surprise – in fact, it's the exact opposite – but it's the biggest lesson. There's nothing quite like the dawning, creeping realisation that the bug was easily avoidable.

Do you have any regrets as a programmer? For example wishing you'd followed a certain technology more closely or something like that?

Listing regrets or indulging in regret is not something I really do, which I would say is no bad thing – and not something I regret..

Where do you think the next big shift in programming is going to come in?

Realising that there are few big shifts in programming that change the fact that, ultimately, it's people who define software. We have met the enemy and he is us.

Are you working on anything exciting at the moment? A new book? Or a new piece of software?

There's a couple of code ideas I'm kicking around that I think are quite neat, but perhaps more for my own interest, and a couple of book projects that have my eye.

Finally, what advice would you offer to kids or adults that are looking to start a career as a programmer?

Look at what's happening now, but also look at what's gone before. If you can figure out they're related, you're doing better than most.

Not quite the reaction you were expecting to the latest release?

Clearly stated...

It may not be the software. How clear are the release notes? What about the product manual, online help, training materials, ...?

Changes may meet a business need, but if what worked yesterday doesn't work today, people may resent them. And when today's way involves extra steps, people work around them. After all, their priority is getting the job done.

Result: those shiny new features remain unused, and your application appears not to live up to its promise.

If you would like some help in turning nervous cats into contented ones, get in touch.

T 0115 8492271

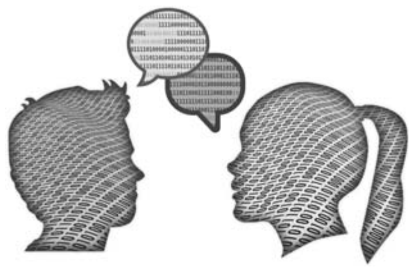
E info@clearly-stated.co.uk

W www.clearly-stated.co.uk

We demonstrate our commitment to professionalism by being members of the Institute of Scientific and Technical Communicators, the UK professional body for technical authors and related professions (visit www.istc.org.uk)

Code Critique Competition 92

Set and collated by Roger Orr. A book prize is awarded for the best entry.



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.

Note: If you would rather not have your critique visible online, please inform me. (We will remove email addresses!)

Last issue's code

I'm trying to migrate my skill set from C to C++ so thought I'd get started with a simple program to fill in a set of strings and print them. But I'm getting a compilation problem on the call to **`std::copy`** that makes no sense to me, although I thought I'd copied it from some working code on the Internet. Can you help me get it to compile?

Can you help this programmer to get past this presenting problem and help them to identify any other issues with their code? The code is in Listing 1.

Listing 1

```
#include <iostream>
#include <iterator>
#include <set>

// compare *contents* not raw pointers
bool string_less(char const *, char const *);

template <class T, class U>
void test(std::set<T, U> s, T p)
{
  if (s.insert(p).second)
    std::cout << "Added " << p << std::endl;
  else
    std::cout << p << " already present";
}

int main()
{
  std::set<char const *,
    decltype(&string_less)> s(string_less);

  test(s, "A");
  test(s, "B");
  test(s, "AB");

  std::copy(s.begin, s.end,
    std::ostream_iterator<
      char const *>(std::cout , " "));
  return 0;
}

bool string_less(const char *s, const char *t)
{
  while (*s == *t)
  {
    s++;
    t++;
  }
  return(*s < *t);
}
```

Critiques

Jim Segrave <jes@j-e-s.net>

There are a few problems with this code:

The **`test()`** function is passed the set by value, rather than by reference, so the result of inserting a new string is lost when the function returns.

Every invocation of **`test()`** is done with a copy of the empty set as it was first constructed in **`main()`**. Changing this to take the set by reference:

```
  void test(std::set<T, U>& s, T p)
                          ^
```

will cause the set **`s`** in **`main()`** to be updated as each string is inserted.

A minor point – there's no **`std::endl`** on the print when the string passed to **`test()`** is already present, so the following invocation of **`test()`** appends its output on the same line.

In **`main()`**, the invocation of **`copy()`** should be passed the return values of the member functions **`begin()`** and **`end()`**. The member name, standing alone is a pointer to the function, not an invocation of the function to get the iterators for begin and end. clang++ prints a reasonably clear pair of error messages:

```
x.cpp:22:15: error: reference to non-static member
function must be called; did you mean to call it
with no arguments?
  std::copy(s.begin, s.end,
            ~~^~~~~
                   ()
x.cpp:22:24: error: reference to non-static member
function must be called; did you mean to call it
with no arguments?
  std::copy(s.begin, s.end,
                     ~~^~~
```

clang++ then generates a working executable by treating it as though the parentheses had been present. It is arguable whether it should be this helpful or should treat this as a fatal error and not produce an executable.

g++ produces an accurate but rather opaque pair of error messages. Within them is the information you need, but it can be hard to see what it's trying to tell you is wrong:

```
/usr/lib/gcc/x86_64-pc-linux-gnu/4.8.3/include/g++-
v4/bits/stl_algobase.h:450:5: note: _OI
std::copy(_II, _II, _OI) [with _II =
std::_Rb_tree_const_iterator<const char*>
(std::set<const char*, bool (*)(const char*, const
char*)>::*)()const noexcept (true); _OI =
std::ostream_iterator<const char*>]
     copy(_II __first, _II __last, _OI __result)
     ^
/usr/lib/gcc/x86_64-pc-linux-gnu/4.8.3/include/g++-
v4/bits/stl_algobase.h:450:5: note:   no known
conversion for argument 1 from '<unresolved
```

ROGER ORR

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

```
overloaded function type>' to
'std::_Rb_tree_const_iterator<const char*>
(std::set<const char*, bool (*)(const char*, const
char*)>::*)()const noexcept (true)'
```

Which does tell you that it can't convert the pointer to function **end** into an iterator. The compile fails.

When this is fixed and the print succeeds, there is no **std::endl**, so the list of set members will spill onto the command line prompt at program exit.

Finally, the **string_less()** function has a severe problem:

```
bool string_less(const char *s, const char *t)
{
  while (*s == *t)
  {
    s++;
    t++;
  }
  return(*s < *t);
}
```

What happens when the strings **s** and **t** are identical? The **while** loop carries on past the terminating nul character and your program is off into undefined behaviour. After fixing the parameter to **test()** to be a reference, try duplicating the first invocation of **test(s, "A")**. On my machine this led to an immediate SIG_SEGV and core dump.

The fix is simple – before checking if ***s == *t**, check that ***s** is not zero:

```
while(*s && (*s == *t))
```

And to close, not a critique, but whenever I see something like:

```
test(s, "A");
test(s, "AB");
tets(s, "B");
```

I think to myself that there ought to be a better way to do this so that adding or removing tests is less work.

If we use an array of pointers to **const** char, we can iterate through the array running the test on each **char ***:

```
const char * test_vec[] = {"A", "A", "AB",
 "B", "", "ABC", "", "foo", "\xe2\x82\xac"};
for(auto sp: test_vec) {
  test(s, sp);
}
```

Now adding or removing test cases involves only adding or removing a quoted string in **test_vec**'s initialiser list.

My resulting version of this program:

```
#include <iostream>
#include <iterator>
#include <set>

// compare *contents* not raw pointers
bool string_less(char const *, char const *);

template <class T, class U>
void test(std::set<T, U>& s, T p)
{
  if (s.insert(p).second)
    std::cout << "Added " << p << std::endl;
  else
    std::cout << p << " already present"
      << std::endl;
}

int main()
{
  std::set<char const *,
    decltype(&string_less)> s(string_less);
  const char * test_vec[] = {"A", "A", "AB",
   "B", "", "ABC", "", "foo", "\xe2\x82\xac"};
  for(auto sp: test_vec) {
    test(s, sp);
  }
  std::copy(s.begin(), s.end(),
            std::ostream_iterator<
              char const *>(std::cout , " "));
  std::cout << std::endl;
  return 0;
}

bool string_less(const char *s, const char *t)
{
  while (*s && (*s == *t)) {
    s++;
    t++;
  }
  return(*s < *t);
}
```

Tom Björkholm <gr@tombjorkholm.se>

When reading the code of Code Critique 91 there are five problems in the code that immediately come to mind: 1. The function **test()** inserts into a local copy of the set 2. There is not much use having **test()** as a template here 3. The set stores pointers not objects. 4. The parenthesis are missing on **s.begin()** and **s.end()** 5. **string_less()** comparison will fail for identical strings

Let's look into them in order.

1. The function test() inserts into a local copy of the set

As the function test is defined

```
void test(std::set<T, U> s, T p)
```

the arguments are passed by value. I.e. inside the function test the variables **s** and **p** hold copies of the values in the calling function. Adding a value to the copy of the set inside test, does not affect the set in main. The set must be passed by reference **void test(std::set<T, U> & s, T p)**

2. There is not much use having test() as a template here

The function **test** is only called with one type. Templates are used if you want to express code so that it can be used with many types. As there is only one type here, you can just as well write it as normal function:

```
void test(std::set<const char *,
  decltype(&string_less)> & s, const char * p)
```

Especially for a novice C++ programmer it is easier to understand the C++ compiler errors in a normal function than in a template function.

We will get back to this function as changes below will introduce more changes to it.

3. The set stores pointers not objects

The C++ standard containers (like **set**) are intended to store values. Here the **set** store pointer (i.e. pointers to characters interpreted as C style strings). This is not good. If the pointed to objects (characters) goes out of scope, we will be left with dangling pointers (pointing to the memory location where the objects once was).

In this particular case the code does not crash (when using the clang compiler on OS X), because the C style strings happen to be compile time constants that remain at their memory location.

To fix this use the C++ **std::string** to store the strings (real values), instead of just pointers. This is easy.

Just change the first line of **main()** to

```
std::set<std::string> s;
```

Add the line

```
#include <string>
```

change the **test** function to

```
void test(std::set<std::string> & s,
  const std::string & p)
```

and change the `ostream_iterator` to

```
std::ostream_iterator<std::string>
```

You will see the complete code together with other fixes below.

4. The parenthesis are missing on `s.begin()` and `s.end()`

The algorithm `std::copy` takes iterators as arguments. The member functions `begin()` and `end()` of the set return an iterator to the beginning and to the end (one past the last element) of the set. Thus, you have to call `begin()` and `end()` to get the iterators to pass to `std::copy`. When you leave out the parenthesis you pass the function pointers to copy instead of the iterators. Write that line as

```
std::copy(s.begin(), s.end(),
  std::ostream_iterator<std::string>(std::cout,
  " "));
```

5. `string_less()` comparison will fail for identical strings

If you have followed my advice to let the set store string objects instead of character pointers, then the function `string_less` will not be needed at all. If you for some reason want to have a `string_less` for C style strings, then the loop must also handle the case of identical strings. Add a test for terminating null byte.

```
while ((*s == *t) && (*s != 0))
```

However, I do recommend that the set should store `std::string` instead of pointers. Using `std::set<std::string>` the complete code will be:

```
#include <iostream>
#include <iterator>
#include <set>
#include <string>
void test(std::set<std::string> & s,
  const std::string & p)
{
    if (s.insert(p).second) {
        std::cout << "Added " << p << std::endl;
    } else {
        std::cout << p << " already present"
          << std::endl;
    }
}
int main()
{
  std::set<std::string> s;
  test(s, "A");
  test(s, "B");
  test(s, "AB");
  test(s, "A");
  std::copy(s.begin(), s.end(),
    std::ostream_iterator<std::string>(
      std::cout, " "));
  return 0;
}
```

Or if you want to have `test()` as a template function (so that you can brag that you have written a template function yourself ;-)

```
template <typename T , typename U> void test(
  T & s, const U & p) {
  if (s.insert(p).second) {
    std::cout << "Added " << p << std::endl;
  } else {
    std::cout << p << " already present"
      << std::endl;
  }
}
```

James Holland <James.Holland@babcockinternational.com>

I think either the student made a mistake when copying (by retyping) the code or the Internet code was of very poor quality. Either way, the problem with the `std::copy` function lies in the first two parameters. The names are correct but they should be member functions of `s` not data members. This is simply corrected by adding an empty parameter list to the names as shown below.

```
std::copy(s.begin(), s.end(),
  std::ostream_iterator<char const *>
    (std::cout, " "));
```

One can spend ages staring at the code in the hope of finding this type of error. As the student suggested, the error message was not all that helpful. At least now the code compiles without error and runs. There are, however, some remaining problems.

One thing I found disconcerting is the definition of `string_less()`. If two identical strings are compared, the `string_less()` will attempt to access memory beyond the end of the strings. When attempting to confirm this, I was surprised to find that it was never called. So I gave up, for the time being, worrying about correcting `string_less()` and started to track down why it was never called. The problem lies with the `test()` function or, more specifically, with its first parameter. The set is being passed into `test()` by value. In other words, a copy of the set is being made and it is the copy into which `test()` inserts the string. After `test()` displays whether or not the string was inserted, the copy of the set is destroyed, leaving the original set without the string being inserted. In fact the original set always remains empty. This has two consequences. Firstly, `test()` will always report that the string is being added. This is because, as the original set never has a string inserted, the copy will always be empty on entry to `test()`. Secondly, and more importantly, it explains why `string_less()` never gets called. When inserting a string into an empty set, there is no need to compare it with existing strings, as there are none, and so `string_less()` is not called.

To correct the problem with `test()`, the set should be passed by reference, not by value. This will result in the set (and now there is only one set) being inserted with strings as expected. The signature of `test()` is now as shown below.

```
void test(std::set<T, U> & s, T p)
```

This brings me back to my concerns I had regarding the behaviour of `string_less()`. Correctly writing this function from first principles is a bit tricky and so I won't attempt it here. Fortunately, there is a standard function that almost does what we want, namely `strcmp()`. This function compares two strings and returns a negative integer if the first string comes before the second string, zero if the two strings are equal and a positive number if the first string comes after the second string. I am sure the student knows that, being a C programmer. All that needs to be done is to determine whether the value returned from `strcmp()` is less than zero. `string_less()` can, therefore, be rewritten as shown below.

```
bool string_less(const char *s, const char *t)
{
  return strcmp(s, t) < 0;
}
```

The `test()` function is still not quite right as the statement that informs us that the string is already present should print a 'new line' after printing the text. Once this is corrected, the program is in a state where it will work as expected.

The code is, however, quite involved and not all that intuitive. The main reason for this is that text is represented by C-style strings. This causes difficulties, especially when storing them in a container, such as an `std::set`, as is done here. It is far better to use C++'s `std::string` for representing text. The main advantage is that `std::string` has the `<` operator built in and so `std::string`s can be easily compared. This removed the need to write a bespoke comparison function as was the case in the original code.

As the student is just beginning to learn C++, it is best to start with a simple example. I feel that defining `test()` as a function template only serves to complicate matters at this stage. Templates can be tackled at a later date. It would appear from the code example that the student is interested in the use and storage of strings. This is a good place to start and the example can be modified to demonstrate this. All that remains to do is to remove the template clause from **test()**, to change template parameter of the **ostream_iterator** object passed to the copy function from **char const *** to **const std::string**, and to add the **#include** directive for **std::string**. The resultant program is listed below.

```
#include <string>
#include <iostream>
#include <iterator>
#include <set>
void test(std::set<std::string> & s,
  std::string p)
{
  if (s.insert(p).second)
    std::cout << "Added " << p << std::endl;
  else
    std::cout << p << " already present"
      << std::endl;
}
int main()
{
  std::set<std::string> s;
  test(s, "A");
  test(s, "B");
  test(s, "AB");
  test(s, "AB");
  std::copy(s.begin(), s.end(),
    std::ostream_iterator<std::string>
      (std::cout, " "));
  return 0;
}
```

Getting code examples from questionable websites is probably not the best way to learn C++. One site that is a good starting point is isocpp.org from where excellent books, for novices and for those with more experience, are suggested. Everyone should join ACCU, of course, for news of conferences, book reviews and much more.

Alex Paterson <alex@tolon.co.uk>

Quick solution: fixing the code

Ok, moving quickly on this one, because solving the coding errors isn't the real issue here:

The call to **s.begin** should be **s.begin()** in the **std::copy** line. Ditto for **s.end**.

The implementation of **string_less** does not check for the end of the string. To be correct it should check for null, assuming that we are dealing with null-terminated strings. E.g.

```
bool string_less(const char *s, const char *t)
{
  while ((*s == *t) && (*s != 0) && (*t != 0))
  {
    s++;
    t++;
  }
  return(*s < *t);
}
```

However, there is already a method that does this for us, **strcmp**, so we can reduce **string_less** to:

```
bool string_less(const char *s, const char *t)
  { return strcmp(s,t) < 0; }
```

Finally, the **test()** method should take its first parameter (the container) by reference (**&**) and the second parameter (the value) by either const-reference (**const&**) or rvalue (**&&**). As it currently stands, the method is taking a copy of the container and adding the string to the copy, which has no effect on the container declared in **main()**.

Long solution: fixing the design

Right, now let's get onto the real problem. I know this is probably only a piece of code for experimental purposes, but in the C++ world, we really should be taking advantage of RAII (Resource Acquisition Is Initialisation) and moving beyond dealing with naked pointers (a.k.a. raw or dumb pointers) – they should only be used as a last resort, say when we need to interface with an API for an external library or operating system call. Don't get me wrong, syntactically it's C++ alright, but dealing with naked pointers like this means that it is not quite in the true spirit of C++.

What's wrong with naked pointers?

The main issue with naked pointers is making sure that the memory is freed only after we're finished with it and not before; freeing memory before we're finished using it is an error and will result in undefined behaviour, whilst not freeing the memory at all leaks memory. Whilst this might sound trivial when considering memory allocation within a single function, it becomes more of a problem when memory allocation and deallocation is split across different functions, classes or libraries.

Moving away from pointers is a crucial concept to move from the low-level world of assembly and C into a the higher level abstraction of C++ and object-oriented programming in general. In the case of strings, the standard C++ library allows us to switch from **const char *** to **std::string** to automatically manage the dynamic allocation and deallocation of strings. Similarly, it also allows us to deal with strings in a more abstract sense, so we can iterate over the characters of a string until we reach the 'end' rather than previously checking for a 'null character'. Specifically to this case, we can take advantage of the fact that **operator<** is defined for the **std::string** class, so we don't need to write our own function.

Encapsulation

Encapsulation is a concept that is more prevalent in C++ than C, I guess mainly due to C++ being an object-oriented language. In the problem code extract, the **test** method is separated from the container, but really in an object-oriented domain, it should be tied to the collection type that it refers to, as shown below.

Whilst this adds many extra lines, it helps to provide encapsulation, where implementation detail is hidden and provides the key concepts of high cohesion and low coupling. The cohesion comes from having the test method and print output encapsulated in the **TestContainer** type. The low coupling comes from removing the container type information from the main method as well as the detail of how to print out the container to **std::cout**.

To aid readability (and therefore maintainability), the conditional in the test method could be replaced with a separate method, making the logic clear. I must admit that I wasn't aware that **set::insert** returned a **pair<>** that could be used to determine whether the value was added or not, so it is probably beneficial to highlight this in the code to aid other programmers who read it (and ourselves when we read it again months or years later).

Improved code, OO style

```
#include <iostream>
#include <iterator>
#include <set>
template<typename ValueType>
class TestContainer
{
public:
  TestContainer& insert(const ValueType& v)
  {
    if (WasInserted(m_set.insert(v)))
      std::cout << "Added " << v << std::endl;
    else
      std::cout << v << " already present"
        << std::endl;
    return *this;
  }
  template<typename StreamType>
  void print_to_stream(StreamType& s) const
  {
    std::copy(
      m_set.begin(),
      m_set.end(),
      std::ostream_iterator<std::string>
        (s, " "));
  }
private:
  //! Simply extracts the result of the set
  //! insertion, which indicates whether the
  //! value was added to the set or not.
  template<typename T>
  bool WasInserted(const T& insertResult)
  { return insertResult.second; }
  std::set<ValueType> m_set;
};
int main()
{
  TestContainer<std::string> s;
  s.insert("A").insert("B").insert("AB");
  s.print_to_stream(std::cout);
  return 0;
}
```

Improved code, functional style

Of course, a more pure functional approach would keep the container type and the methods related to it separate.

```
#include <iostream>
#include <iterator>
#include <set>

//! Return the second element from the pair<>
//! value that was returned from a call to
//! set::insert. This indicates whether the
//! value passed to insert was added or not.
template <typename T>
bool WasInserted(const T& t)
{ return t.second; }
template <class T, class V>
T test(const T& tOldContainer, const V& val)
{
  T tNewContainer(tOldContainer);
  if (WasInserted(tNewContainer.insert(val)))
    std::cout << "Added " << val << std::endl;
  else
    std::cout << val << " already present"
      << std::endl;
  return tNewContainer;
}

//! Copy the elements of a container to a
//! stream. One use is to print the elements
//! of a container to std::out.
template<typename StreamType,
  typename ContainerType>
void print_to_stream(StreamType& s,
  const ContainerType& c)
{
  std::copy(
    begin(c),
  end(c),
  std::ostream_iterator<std::string>
    (s, " "));
}
int main()
{
  std::set<std::string> s;
  s = test(s, "A");
  s = test(s, "B");
  s = test(s, "AB");
  print_to_stream(std::cout, s);
  return 0;
}
```

Commentary

This problem hinges on the difference between a pointer to a member function and the value resulting from calling a member function. As Jim found, clang++ tries to be helpful and guesses what was meant. While this is correct, in this case, I share Jim's concern that this may not be helpful in general.

Of course, there were other problems with the code. Passing the set into the test function by value not by reference was wrong here – but the program is correct *code* in both cases and it is hard for automated tools to detect this sort of semantic issue. One of the slight concerns over the use of `auto` in recent C++ code is the way it can hide whether the actual type is a value or a reference in cases where this matters a great deal.

What no-one really challenged about the program is that it isn't actually a useful test. The program writes output and returns 0 – whether or not the bug is fixed. A more useful test would report success or failure, so it is obvious without needing to read the test in detail whether or not the code is behaving as expected.

James did point out that the test did not exercise the `string_less` function in the original code, and his solution does try testing `"AB"` twice – but without testable success criteria it is not obvious what output from the program is actually correct.

I would encourage the programmer to decide what it means to pass the test here and to ensure the code unambiguously reports success only if this occurs.

The final niggle I have with the code is the use of `std::copy` to print the set. The `ostream_iterator` will place a copy of the delimiter after each element output, including the last one, and in some environments this trailing space can cause problems. This becomes a bigger problem when the delimiter is visible, for example trying to use a comma.

Unfortunately C++ doesn't yet have a standard way to fix this although there are some active proposals for delimited iterators so watch this space!

The winner of CC91

All four critiques identified the major problem and suggested how to fix it and all also noted the problem with `string_less` on equal strings.

Two people suggested implementing `string_less` with `strcmp` (although no-one pointed out this may also be more efficient than the hand-rolled string code, especially for larger strings, as library writers are likely to deliver an optimised implementation).

Several people recommended using `std::string` instead of character pointers and I would generally follow this recommendation myself too. Alex gave a good summary of why this is generally better.

Jim improved the test generation by making it data driven, which makes it very easy to change. It is all too easy to write test code using copy and paste resulting in a lot of unnecessary duplication.

In addition to the critique itself, James also pointed the programmer to some further useful sources of information; so I have awarded him the prize for this critique by a short head.

Code Critique 92

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

> I'm trying to use the new(ish) `unordered_map.` and my simple test program works with some compilers and options but not with others. I can't see why it wouldn't work consistently – are some of the compilers just broken? I do get a compiler warning about conversion from string constant to cstring – is that what's wrong?

Can you help this programmer to get past this presenting problem and help them to identify any other issues with their code? The code is in Listing 2.

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

Standards Report

Mark Radford reports on the latest C++ Standardisation proceedings.

Hello and welcome to my latest standards report.

In this report I'm rather short of events to report on. There have been no more full ISO C++ meetings, and the next one will be after I have to ship this report (sorry, I don't have any information at the moment on what's happening in the C standards process). There will be a BSI C++ Panel meeting on Monday 9th February, but this time I can't attend and I'm not likely to see the minutes in time to write anything about it before I have to ship this report. There has, however, been an ISO C++ meeting dedicated to discussing the Concepts TS; this took place in Skillman, HJ, USA on the 26th, 27th and 28th of January this year. I wasn't there, but one of my BSI C++ Panel colleagues was [1], so I have information on the proceedings.

Also a couple of C++ topics – namely, uniform call syntax and the overloading of operator dot (which I discussed briefly in my last report) – have come up recently. If either/both of these are achieved it would have an impact on the way C++ code is designed. Therefore both topics are important and merit some detailed coverage.

Skillman

The purpose of the meeting was to continue to review the Concepts document, and to determine the document's readiness for PDTS publication. The meeting also agreed on the categorisation of issues found. While there were more than two categories, the fundamental point was whether or not an issue needed fixing before the PDTS could ship. Happily, during the process it was found that none of the issues were serious enough to prevent shipping!

Unfortunately, extreme weather conditions on the first day of the meeting dictated that it needed to be adjourned at lunch time. For the same reason it was decided not to meet face-to-face the following day, but to have a teleconference instead. Andrew Sutton, the Concepts document author, was asked to address the issues that had been found in time for the following day, so the changes could be reviewed by the meeting.

When the meeting reconvened, it began by going through the updated version of the paper. There were some additional comments, including issues raised (while the meeting was taking place) on the BSI C++ Panel reflector. However none of the new issues were serious enough to delay shipping. Therefore, the outcome was to move the document (with some editorial fixes) to PDTS.

Of course, the big question now is whether or not Concepts can make it onto C++17. The problem is that not everyone is convinced we have the right design. No one has a problem with the current Concepts design going into a TS, but that's a very different thing from the International Standard (IS). While a TS can be a stepping stone to a feature going into the IS, the TS is also a proving ground i.e. something compiler writers can implement so real world experience can be obtained before a feature is committed to the IS.

MARK RADFORD

Mark Radford has been developing software for twenty-five years, and has been a member of the BSI C++ Panel for fourteen of them. His interests are mainly in C++, C# and Python. He can be contacted at mark@twonine.co.uk

Code Critique Competition #92 (continued)

Listing 2

```
#include <iostream>
#include <string>
#include <unordered_map>
typedef char *cstring;
typedef std::unordered_map<cstring, cstring>
AddressMap;
// Hold email addresses
class Addresses
{
public:
  // add addr for name
  void add(cstring name, cstring addr)
  {
    addresses[name] = addr;
  }
  // get addr from name or null if not found
  cstring get(cstring name)
  {
    if (addresses.count(name))
    {
      return addresses[name];
    }
    return 0;
  }
```

Listing 2 (cont'd)

```
private:
  AddressMap addresses;
};

int main()
{
  Addresses addr;

  addr.add("roger",
    "rogero@howzatt.demon.co.uk");
  addr.add("chris",
    "chris@gmail.com");

  cstring  myadd = addr.get("roger");
  if (myadd == 0)
  {
    std::cout << "Didn't find details"
      << std::endl;
    return 1;
  }
  std::cout << "Found " << myadd << std::endl;
}
```

Talking of real world experience, note that there is a Concepts branch of GCC; there was hope that the Concepts branch could be merged into GCC 5, but unfortunately this is now not going to happen as progress on the branch was not quite rapid enough. I don't have the details to hand, but anyone interested in playing with this implementation should have little difficulty in finding the branch, which is publicly available.

Uniform Call Syntax

I see this topic is once again under discussion, having been discussed on and off for over a decade (many of the discussions being informal). A discussion on the Evolution Working Group (EWG) reflector drew my attention to it, that discussion having been sparked by two papers in the pre-Urbana mailing: 'Unified Call Syntax' (N4165) is by Herb Sutter [2], and 'Call syntax: x.f(y) vs. f(x,y)' (N4174) is by Bjarne Stroustrup [3].

There are several good motives for doing this. One example is greater flexibility for generic programming: given a template parameter **`T`**, when invoking an operation involving **`T`** the generic code has to commit itself to assuming one syntax or the other i.e. member function or free-standing function.

Historically, when this topic has been discussed, there has been an implicit assumption that the free-standing call syntax would potentially call a member function. A simple example is: **`f(x)`** could potentially call **`x.f()`**. However, in 'Unified Call Syntax', Herb Sutter looks at the problem the other way, suggesting that **`x.f()`** could also potentially call **`f(x)`**. Note that, in particular, he says if the call can be resolved by a member function, then no further lookup should take place. This avoids any impact on existing code. Meanwhile, in 'Call syntax: x.f(y) vs. f(x,y)', Bjarne Stroustrup does not take any one viewpoint. Instead he presents a discussion of all the approaches to unifying the call syntax; this does not just involve syntax, as there are also various approaches to how the semantics would work: for example, if the **`f(x)`** syntax is preferred, should member functions be looked up only if no non-member match is found, or should they be included as equals for overload purposes? Note that the latter option is a silent breaking change, so I hope its inclusion extends to simply stating the complete set of options for comparison. I get nervous when I see such a silent breaking change even mentioned!

Anyway, I've said enough about those two papers in this report. Both are publicly available, so anyone who wants to know more can download them. I'll move onto 'Call Syntax Questions' by Bjarne Stroustrup, a paper which he circulated on the EWG reflector and which is not currently publicly available. Currently, I don't know whether or not it will be included in a future mailing. Actually, I suspect it won't be: looking at the introductory remarks, there is an indication that feedback on this paper will be used to revise the original (N4174) paper. As I write this I see the new (Feb 2015) mailing has just been published [4], but I can't see any new papers on this topic, or any updated editions of the two exiting papers.

'Call Syntax Questions' makes some compelling arguments against the **`x.f()`** syntax. For example: it feels "too object-oriented" may sound like an objection based on taste, but it starts to look like requiring **`x.f()`** is in rather bad taste when you consider that **`x.sqrt()`** looks rather strange, and **`2.sqrt()`** just looks plain wrong! This is C++ of course, and what looks right/wrong in C++ isn't necessarily the same as in other languages. Further, moving towards member function syntax encourages making functions members, thus increasing coupling. Finally, what about lambda functions passed as template arguments? They must be called using the **`f(x)`** notation. To quote the paper: "Not all opposition to **`x.f(y)`** is emotional and aesthetic".

Operator Dot

In my last report I talked briefly about "Operator Dot" by Bjarne Stroustrup and Gabriel Dos Reis (N4173). This is the latest in a series of proposals to make "operator dot" overloadable (the paper mentions several of its predecessors). Clearly, the overloading of operator dot has its applications: smart references (by analogy with smart pointers) are an obvious one, and the facility to write proxy classes more easily is another. This proposal has been very well received and there is general agreement that the work should continue. However there are some concerns.

One of the concerns is the possibility of introducing silent changes to templates. For example, consider the following fragment:

```
struct X
{
  X& operator=(int i);
  int i;
};

struct T
{
  X theX;
  T& operator=(int i);
};

T t;
t = 42;
```

What you have to remember here is that **`t = 42`** actually says **`t.operator = (42)`**, which means that if **`T`** is modified as follows (i.e. with the added operator dot):

```
struct T
{
  X theX;
  T& operator=(int i);
  X& operator.() { return theX; }
};
```

the actual meaning is **`t.operator.().operator = (42)`**, which in turn means the assignment is really to **`x`** and not to **`t`**. If **`T`** were used as a template argument this would bring about a silent, and probably surprising, change to the code's semantics [5].

Looking at this now, I'm wondering how much of a problem it really is. I'm thinking that overloading operator dot is really a design feature of a class, because it's a smart reference or some kind of handle i.e. it's not really the sort of thing likely to be added later on. Also, C++ is full of features that potentially cause pitfalls, and programmers should always think about the possible consequences of changing code. I know the fact that pitfalls are already in the language isn't a reason to add more. However, overloading operator dot does have benefits, and the standards committee must always weigh up the benefits afforded by a new feature versus the problems it (potentially) causes. Anyway, the jury is still out on operator dot, but I think it's one to watch with interest.

Acknowledgements/References

[1] Thanks to Dinka Ranns for the excellent reports (posted to the BSI C++ Panel reflector) from Skillman, on which I have drawn in my report.

[2] 'Unified Call Syntax' (N4165) is by Herb Sutter can be found at: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4165.pdf

[3] 'Call syntax: x.f(y) vs. f(x,y)' (N4174) by Bjarne Stroustrup can be found at: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4174.pdf

[4] The latest mailing can be found at: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/#mailing2015-02

[5] Thanks to Roger Orr for helping me understand this.

ACCU Information

Membership news and committee reports

accu

View from the Chair

Alan Lenton
chair @accu.org

Time for another edition of *CVu*. Considering that this esteemed publication only comes out every two months, demands for my contribution seem to arrive much more frequently than I would have expected!

So what's been going on since I last put fingers to keyboard?

Well, we had a long committee meeting, at which the main topic was the future of ACCU. The good news is that the committee does believe that ACCU has a future! However, it will take time and effort to rebuild the membership up to its former levels. There was much discussion, and eventually the committee took a position on three specific issues.

First, the committee decided to recommend to the Annual General Meeting (AGM) that the membership dues remain at the same level as this year for the next year. As I understand it, we are doing well enough to meet our current obligations, although not well enough to start holding committee meetings in the Bahamas...

Secondly, it was the committee's view that the magazines, *CVu* and *Overload*, should remain printed publications. There were a number of suggestions sent in suggesting that the publications, or perhaps one of them, should become e-publications. However, it's my experience that a large proportion of the membership are firmly attached to print publications. That's perhaps not surprising in view of the problems in using e-book readers to read technical publications, especially ones that have fixed font code and diagrams.

It may be that in a few years time, e-books will have the bugs worked out of them and have become the preferred way of consuming 'printed' matter. At that stage we can re-open the matter, and, perhaps take a different decision. In any case, I personally would definitely want to see a decision from the whole membership before embarking on a change of that magnitude.

As an aside I'd like to thank those of you who took the trouble to write in with suggestions on this and other issues, the ideas were very helpful.

However, given the extent to which the cost of publishing and distribution dominate ACCU's finances, the committee did feel that it should look into whether we should have a single 48 page magazine six times a years, rather than the existing two 24 page magazines. Doing so would cut the cost of distribution (though not by half). The committee didn't take a position on this, and the magazine editors would undoubtedly need to have a say in this decision before it's taken. I will be interested to hear opinions on this matter at the Annual General Meeting, and at Conference this year (more on that later).

The third thing the committee discussed was how to rebuild the membership, not to mention keeping the ACCU relevant to the existing membership! To cut a long story short, it is the committee's belief that the way to do this is by nurturing and building up strong local groups. That will be what the committee will be proposing at the AGM. This will involve all sorts of issues from funding through to building up speakers lists and giving advice on how to set up regular local meetings. We were fortunate to have Nigel Lester from the Oxford local group in attendance at the committee meeting, and his contribution to the discussion was extremely valuable.

I mentioned conference earlier on. We ('we' being the ACCU) have a 90 minute slot at the conference on Friday afternoon. Tentatively, I've organized it into two 45 minute sessions; one to discuss local groups – hopefully with Nigel on hand for high grade advice, and one to discuss writing for our magazine, hopefully with our editors in attendance. I know that there are lunchtime and evening Birds of a Feather meetings most days, but this will mean we can have a decent discussion without everyone munching food, or rushing off to get Lakos-ed! And, you can still have BoFs as well, should you so wish...

Well, I think that's about all there is for this issue – hopefully I'll meet some of you at the ACCU Conference this year.

Advertise in C Vu & Overload

80% of readers make purchasing decisions, or recommend products for their organisations.

Reasonable rates. Flexible options. Discounts available to corporate members.

Contact ads@accu.org for info.

Learn to write better code

Take steps to improve your skills

Release your talents

ACCU PROFESSIONALISM IN PROGRAMMING **JOIN : IN**