



[clivetong.wordpress.com](http://clivetong.wordpress.com)



# Contents

<b>2007</b>		<b>19</b>
December	.....	19
Common Lisp - thanks for the good times! (2007-12-13 10:47)	.....	19
A note on laziness (2007-12-13 11:46)	.....	21
A little extra (2007-12-13 13:22)	.....	23
Let's tie the knot (2007-12-13 21:12)	.....	24
Let's start at the beginning (2007-12-17 10:41)	.....	25
<b>2008</b>		<b>27</b>
January	.....	27
Windows Presentation Foundation Unleashed (2008-01-02 14:12)	.....	27
What's my name again? (2008-01-04 09:51)	.....	28
Meta Maths: The Quest For Omega by Gregory Chaitin (2008-01-07 11:07)	.....	39
Beautiful code Edited by Andy Oram and Greg Wilson (2008-01-13 14:46)	.....	40
The Haskell School Of Expression by Paul Hudak (2008-01-14 12:53)	.....	43
I'm not complex, I'm linear (2008-01-15 09:50)	.....	44
Please, let me continue! (2008-01-16 11:21)	.....	46
Yes please, I'd like a game (2008-01-21 13:53)	.....	52
How did he do that? (2008-01-22 14:49)	.....	53
Not just a normal variable. I'm special! (2008-01-23 11:04)	.....	54
February	.....	59
No Mr Postman. Visit the houses in this order! (2008-02-10 16:47)	.....	59
Pacman... march on! (2008-02-10 16:53)	.....	60
Put that down... it's sharp (2008-02-14 18:41)	.....	60
That's an interesting bit of information (2008-02-20 12:19)	.....	61
March	.....	62
It won't twist that way! (2008-03-04 08:11)	.....	62
It's a bit like lego (2008-03-09 14:57)	.....	63
It's a generational thing (2008-03-09 15:13)	.....	64
That argument is not complete! (2008-03-25 13:10)	.....	64
You're nothing like the prototype (2008-03-27 08:02)	.....	65
Testing ... testing...1...2...3 (2008-03-27 08:15)	.....	68
April	.....	72
Keep this a secret! (2008-04-07 21:24)	.....	72
Quickly now! (2008-04-15 07:29)	.....	79
Fail OrElse.... (2008-04-20 20:31)	.....	85
What does that denote? (2008-04-21 07:10)	.....	99
May	.....	100
Just get on with it will you! (2008-05-01 07:50)	.....	100

Abstraction isn't always good for you (2008-05-01 08:15) . . . . .	102
That's life (2008-05-14 16:05) . . . . .	104
It's very lively isn't it! (2008-05-28 17:36) . . . . .	105
June . . . . .	115
It's all about the people (2008-06-10 08:55) . . . . .	115
No need to apologise (2008-06-10 09:17) . . . . .	116
You're a bit remote today (2008-06-10 13:18) . . . . .	117
I'm an actor! (2008-06-11 07:26) . . . . .	118
July . . . . .	122
What difference does it make? (2008-07-06 17:05) . . . . .	122
You're a little too abstract for me (2008-07-07 12:18) . . . . .	131
It's a little surprising when you first see it (2008-07-07 14:49) . . . . .	133
It's not a question of "or" - it can be "and" (2008-07-07 18:47) . . . . .	137
Cut! (2008-07-16 18:53) . . . . .	145
August . . . . .	146
Voyage to the bottom of the sea (2008-08-12 05:56) . . . . .	146
Let's share some info (2008-08-14 09:18) . . . . .	147
Help is at hand (2008-08-27 16:25) . . . . .	148
September . . . . .	148
Flash... no..oh. (2008-09-08 18:29) . . . . .	148
Play it again Sam. (2008-09-08 18:45) . . . . .	151
Stand back - I'm a F#'ing scientist (2008-09-08 19:07) . . . . .	152
It's a bit of an enigma (2008-09-08 19:16) . . . . .	153
Listen to this (2008-09-08 20:33) . . . . .	154
The good, the bad and the ugly (2008-09-21 16:41) . . . . .	155
That's a very impressive CV (2008-09-21 16:46) . . . . .	156
November . . . . .	156
It's dark out there (2008-11-29 18:32) . . . . .	156
Ponder it? It almost blew my mind (2008-11-29 19:18) . . . . .	157
Style only gets you so far (2008-11-29 19:32) . . . . .	158
Software East (2008-11-29 19:39) . . . . .	160
There's only one way I can react to that (2008-11-29 19:42) . . . . .	161
December . . . . .	162
F# can even take out the garbage (2008-12-09 10:26) . . . . .	162
Let's chat about F# (2008-12-09 10:34) . . . . .	173
Install, damn you! (2008-12-09 10:52) . . . . .	174
Stop bleeding! (2008-12-14 08:32) . . . . .	175
Let's have a chat (2008-12-15 07:39) . . . . .	176
I'll have to check if you are my type (2008-12-15 07:43) . . . . .	176
<b>2009</b>	<b>181</b>
January . . . . .	181
F# I'll sieve my way through that composition (2009-01-04 11:28) . . . . .	181
That'll get a reaction (2009-01-04 11:45) . . . . .	187
Great functional datastructures 1 (2009-01-04 11:52) . . . . .	188
Task finished (2009-01-04 12:44) . . . . .	188
Let's catch up (2009-01-04 12:52) . . . . .	190
Keep it secret! (2009-01-04 13:00) . . . . .	190
Faster... faster... (2009-01-21 22:07) . . . . .	191
Even if you're lazy, you can do a lot at the same time (2009-01-27 21:16) . . . . .	202

February . . . . .	204
You're so vain.... (2009-02-01 15:04)	204
I've got a message for you! (2009-02-01 15:47)	205
A great read. (2009-02-02 19:05)	208
It's not knitting (2009-02-06 21:13)	209
Virtually brilliant - part one (2009-02-08 22:14)	213
I'll parse that in a sec (2009-02-08 22:24)	224
It's a joint effort (2009-02-14 10:06)	224
Basically you can join things together (2009-02-27 22:25)	231
March . . . . .	232
Ajax without the 'j' (2009-03-05 18:22)	232
Just like busses (2009-03-05 22:11)	233
Some low level surgery on F# (2009-03-10 20:03)	233
You will keep count won't you (2009-03-11 20:51)	241
That might be true (2009-03-13 20:34)	247
It might be secure (2009-03-13 20:37)	248
Prove it (2009-03-16 10:07)	248
April . . . . .	251
Bring out your dead (2009-04-14 17:16)	251
Caught up in the web (2009-04-22 17:12)	252
Come at me with the banana (2009-04-23 07:27)	253
There's something in my eye (2009-04-24 16:37)	253
Greengrocers know best! (2009-04-25 21:46)	254
Take your time (2009-04-25 21:56)	255
A couple of good papers (2009-04-26 20:01)	257
May . . . . .	258
Next please (2009-05-10 03:53)	258
Javascript is Bak (2009-05-10 22:55)	258
I'll return your call later (2009-05-20 21:20)	259
Play it again (2009-05-30 22:21)	267
It's a matter of trust (2009-05-30 22:35)	268
June . . . . .	269
What category would you place that in? (2009-06-11 19:00)	269
Divide (2009-06-11 19:25)	276
NxtGen Fest 09 (2009-06-15 11:24)	276
What's next? (2009-06-24 20:44)	278
It's one thing after another (2009-06-24 20:53)	278
What difference does it make? (2009-06-24 20:58)	279
Don't get in a flap (2009-06-24 21:02)	280
July . . . . .	280
I can see right through you (2009-07-04 21:28)	280
Thanks for clearing that up (2009-07-04 21:38)	286
Some more reactions (2009-07-11 08:28)	286
If you re-implement, you may get more than you bargained for (2009-07-13 21:53)	287
The release event has arrived (2009-07-24 06:10)	290
So that's what you do all day (2009-07-24 06:12)	290
August . . . . .	290
As easy as Pi (2009-08-02 15:53)	290
A nett gain of knowledge (2009-08-02 16:00)	292
How good is your translation? (2009-08-02 16:08)	293

Say that again... (2009-08-02 16:14) . . . . .	296
I wish I were that clever (2009-08-07 16:53) . . . . .	297
Money, money, money. (2009-08-07 17:03) . . . . .	298
Refactor this. (2009-08-07 17:21) . . . . .	298
I have achieved closure (2009-08-26 17:09) . . . . .	299
You're deeper than you look (2009-08-26 17:11) . . . . .	300
If you don't succeed, retry. (2009-08-26 17:14) . . . . .	300
September . . . . .	301
Put it together, then break it apart (2009-09-02 18:26) . . . . .	301
I've got a good memory for transactions (2009-09-08 19:51) . . . . .	302
It's not what you do, but the way that you do it (2009-09-20 15:25) . . . . .	303
You should have seen through that (2009-09-26 16:17) . . . . .	304
I'm not too lazy to watch them! (2009-09-26 16:41) . . . . .	304
So that's how it works! (2009-09-28 08:31) . . . . .	305
That's what its good for (2009-09-28 08:32) . . . . .	305
Different, yet the same (2009-09-28 08:33) . . . . .	306
That was quicker than I expected (2009-09-30 16:58) . . . . .	306
October . . . . .	309
Assembly required! (2009-10-03 07:43) . . . . .	309
LINQ into the data stream (2009-10-03 07:56) . . . . .	311
Go on, suggest something (2009-10-05 22:15) . . . . .	313
Some trees can only have certain kinds of branches (2009-10-18 17:22) . . . . .	316
It's alive, but not as we know it (2009-10-18 17:31) . . . . .	316
Keep the LINQ short (2009-10-23 20:01) . . . . .	326
Faster, faster (2009-10-23 20:03) . . . . .	328
November . . . . .	329
That's the way you do it (2009-11-03 20:05) . . . . .	329
That's something you can't do without (2009-11-12 20:32) . . . . .	332
You can depend on this (2009-11-12 20:35) . . . . .	332
How many dwarves? (2009-11-12 20:36) . . . . .	333
What a lot of developers! (2009-11-27 20:57) . . . . .	334
Oh, I say! (2009-11-29 10:45) . . . . .	335
Google that (2009-11-29 10:50) . . . . .	335
Burning bright (2009-11-29 10:58) . . . . .	335
Negotiate only once (2009-11-30 20:00) . . . . .	336
December . . . . .	336
The eagle has landed (2009-12-14 19:37) . . . . .	336
Attack is the best defense (2009-12-14 19:40) . . . . .	337
Can't go under it and can't go over it (2009-12-28 22:09) . . . . .	338
Bring your own boxes (2009-12-28 22:23) . . . . .	339
<b>2010</b>	<b>341</b>
January . . . . .	341
A journey through the maze of twisty passages (2010-01-03 12:14) . . . . .	341
Sorry, we're closed (over) (2010-01-04 16:45) . . . . .	356
How tight is the binding? (2010-01-05 21:51) . . . . .	358
Call my Agent will you? (2010-01-07 21:01) . . . . .	361
What do you base that on? (2010-01-09 14:00) . . . . .	369
Let's compose some arguments (2010-01-10 09:27) . . . . .	373
What's next in the sequence? (2010-01-10 21:09) . . . . .	375

What's the name of my space? (2010-01-13 21:04) . . . . .	380
Get Real! (2010-01-13 21:06) . . . . .	386
What sort of structure is that? (2010-01-16 09:34) . . . . .	386
Crash bang! (2010-01-16 09:34) . . . . .	388
I thought I'd just done that (2010-01-17 21:39) . . . . .	389
Simple is not always easy (2010-01-21 19:55) . . . . .	398
Where is my mind? (2010-01-24 15:28) . . . . .	403
Laziness is good for you (2010-01-24 15:29) . . . . .	403
Change of heart (2010-01-24 21:10) . . . . .	406
I'll be back (2010-01-24 22:01) . . . . .	406
De-structure the structure (2010-01-25 22:32) . . . . .	410
Back to the future (to get its value) (2010-01-26 21:38) . . . . .	412
Macro mayhem (2010-01-28 07:11) . . . . .	413
Off you go (2010-01-30 14:46) . . . . .	417
<b>February . . . . .</b>	<b>421</b>
There are no bugs here! (2010-02-07 22:38) . . . . .	421
Exceptionally good (2010-02-07 22:39) . . . . .	422
This and that (2010-02-07 22:39) . . . . .	423
There's more than one way to do that (2010-02-09 22:14) . . . . .	429
Say hello to FeeFee (2010-02-11 22:42) . . . . .	434
That's very convoluted (2010-02-13 10:10) . . . . .	438
Short and Sweet (2010-02-14 17:37) . . . . .	438
That's not what you said last time (2010-02-14 17:40) . . . . .	439
It's not Pandora's (2010-02-20 22:35) . . . . .	439
JSON told me the answer (2010-02-20 23:03) . . . . .	439
How did you do that (2010-02-22 18:18) . . . . .	440
I don't want another argument (2010-02-22 18:19) . . . . .	441
I thought I'd tested that! (2010-02-25 20:31) . . . . .	443
Take your PIC (2010-02-25 20:31) . . . . .	445
Watch out for the apple! (2010-02-28 14:57) . . . . .	447
A little lopsided (2010-02-28 14:57) . . . . .	447
Thanks for your contribution (2010-02-28 22:18) . . . . .	448
<b>March . . . . .</b>	<b>449</b>
Its important to have a good memory (2010-03-06 12:20) . . . . .	449
Model it and pass it on! (2010-03-08 16:03) . . . . .	449
Bring out your dead (2010-03-20 21:54) . . . . .	450
That's an interesting development (2010-03-21 20:31) . . . . .	451
It's hard to interoperate (2010-03-21 20:32) . . . . .	453
The domain of the thread (2010-03-25 20:58) . . . . .	453
<b>April . . . . .</b>	<b>456</b>
Left, Right, Left, Right... (2010-04-02 18:52) . . . . .	456
Tell me about it (2010-04-02 18:52) . . . . .	456
On reflection, it's better to call directly (2010-04-07 18:02) . . . . .	457
That's a letter, not a number (2010-04-11 08:20) . . . . .	458
Pack your bags (2010-04-11 08:21) . . . . .	458
Watch these (2010-04-11 21:23) . . . . .	460
Pack your bags 2 (2010-04-11 21:24) . . . . .	460
Don't get into a fight with him (2010-04-25 14:23) . . . . .	462
Down, down, deeper and down. (2010-04-25 14:23) . . . . .	462
Procrastination is (in fact) the thief of space (2010-04-25 14:23) . . . . .	462

What material is that made from? (2010-04-28 08:46) . . . . .	465
May . . . . .	466
Is that structurally sound? (2010-05-20 18:52) . . . . .	466
Blimey (2010-05-25 17:47) . . . . .	467
One thing at a time please! (2010-05-25 17:48) . . . . .	468
That's a bit complex (2010-05-25 17:48) . . . . .	469
You spin me around and around (2010-05-25 17:53) . . . . .	471
Put Something In The Way (2010-05-28 19:41) . . . . .	472
That's some beautiful architecture (2010-05-29 08:58) . . . . .	472
Knock me down and I'll keep coming back (2010-05-29 21:09) . . . . .	474
June . . . . .	474
I'm not bothered! (2010-06-11 19:25) . . . . .	474
That's very general (2010-06-18 18:26) . . . . .	475
You're talking about my generation (2010-06-18 18:47) . . . . .	476
One step at a time (2010-06-23 20:51) . . . . .	477
July . . . . .	477
Er, mind your language (2010-07-03 07:47) . . . . .	477
Calm - don't react! (2010-07-12 22:05) . . . . .	480
I said stay calm (2010-07-13 20:34) . . . . .	483
Don't believe everything you read (2010-07-18 20:10) . . . . .	483
Back to basics (2010-07-18 20:11) . . . . .	484
August . . . . .	484
Chip away at the problem (2010-08-01 15:24) . . . . .	484
Keep your ancestors hidden (2010-08-08 05:59) . . . . .	485
They've got a load of class (2010-08-08 05:59) . . . . .	486
Marvin, marvellous (2010-08-18 21:06) . . . . .	486
I really did read it! (2010-08-22 18:18) . . . . .	488
I know I'm ponderously slow (2010-08-22 18:18) . . . . .	488
Time to be a little more dynamic (2010-08-23 16:57) . . . . .	489
Get rid of it quickly (2010-08-30 17:18) . . . . .	495
That's one aspect of the problem (2010-08-30 17:22) . . . . .	498
September . . . . .	500
Get down! (2010-09-02 19:29) . . . . .	500
Steady! (2010-09-05 13:36) . . . . .	503
Break it down (2010-09-05 13:37) . . . . .	503
Visibility is bad (2010-09-18 07:26) . . . . .	504
2000 years and you haven't reached five? (2010-09-18 07:27) . . . . .	504
Ouch that's sharp (2010-09-23 13:01) . . . . .	505
Get that through the eye of a needle (2010-09-23 13:13) . . . . .	505
It's a Joy (2010-09-30 20:08) . . . . .	507
October . . . . .	507
Another time, another place (2010-10-16 02:41) . . . . .	507
I wish I knew that (2010-10-16 02:43) . . . . .	508
Hoisted by his own bootstraps (2010-10-16 02:46) . . . . .	508
Taking self-reference to the extreme (2010-10-22 20:24) . . . . .	510
All together now! (2010-10-31 13:01) . . . . .	510
November . . . . .	512
What language do ghosts use? (2010-11-06 10:03) . . . . .	512
That's a library Hercules would have been proud of (2010-11-14 15:34) . . . . .	513
Give me some time (2010-11-14 15:40) . . . . .	514

That's the way you do it (2010-11-23 18:58) . . . . .	517
It's all connected you know (2010-11-23 19:03) . . . . .	517
My evaluation is good (2010-11-27 21:28) . . . . .	517
Check to see if that's my type of thing (2010-11-27 21:28) . . . . .	519
<b>December . . . . .</b>	<b>519</b>
That's a lot more than 6 items (2010-12-04 14:08) . . . . .	519
I can live within those constraints (2010-12-04 14:41) . . . . .	520
Bless you! (2010-12-19 18:06) . . . . .	523
Thanks for the purity (2010-12-19 18:08) . . . . .	524
Very enthusiastic (2010-12-19 18:10) . . . . .	524
Pause a second would you? (2010-12-27 20:21) . . . . .	525
This is the first iteration (2010-12-29 20:53) . . . . .	526
On your Next action, please follow these guidelines (2010-12-29 20:55) . . . . .	526
That's not the value you showed me before (2010-12-29 20:57) . . . . .	527
<b>2011</b>	<b>529</b>
<b>January . . . . .</b>	<b>529</b>
Not just William (2011-01-03 12:46) . . . . .	529
Phone me! (2011-01-03 12:48) . . . . .	529
One at a time please! (2011-01-10 22:49) . . . . .	530
I did know that (2011-01-25 21:33) . . . . .	531
Share! (2011-01-25 22:20) . . . . .	531
<b>February . . . . .</b>	<b>532</b>
How do you build a football? (2011-02-18 20:15) . . . . .	532
If it's broken, fix it. (2011-02-28 20:06) . . . . .	533
Can you say that a different way? (2011-02-28 20:08) . . . . .	533
No it's not an assembly language (2011-02-28 20:10) . . . . .	534
<b>March . . . . .</b>	<b>535</b>
Yes, but can I rely on that? (2011-03-05 12:04) . . . . .	535
What a conference, what a night! (2011-03-12 17:41) . . . . .	535
Nobody said it was easy, nobody said it would be that fast (2011-03-19 09:57) . . . . .	537
Skills do matter (2011-03-19 10:04) . . . . .	539
<b>April . . . . .</b>	<b>540</b>
Get back, he's got a book (2011-04-08 20:01) . . . . .	540
Call me when you've done the second version (2011-04-23 07:57) . . . . .	541
No more heroes any more (2011-04-25 16:25) . . . . .	541
Check (2011-04-25 16:25) . . . . .	542
It certainly brings out the artist in you (2011-04-25 17:05) . . . . .	542
<b>May . . . . .</b>	<b>543</b>
Where is the string that Theseus laid? (2011-05-07 19:16) . . . . .	543
What's on your mind? (2011-05-07 19:16) . . . . .	543
No assembly required (2011-05-16 23:30) . . . . .	543
Crash bang (2011-05-22 09:34) . . . . .	544
You looked better from the other side (2011-05-22 09:35) . . . . .	544
Mix it up (2011-05-22 09:49) . . . . .	544
<b>June . . . . .</b>	<b>546</b>
A tale of recursion (2011-06-06 17:21) . . . . .	546
Thanks for the memory (2011-06-06 20:25) . . . . .	548
That's not the kind of thing I provide (2011-06-07 20:25) . . . . .	550
That had slipped my memory (2011-06-11 06:04) . . . . .	550

Watch out for the dragons (2011-06-11 06:09) . . . . .	551
Before you know it you're up to version 4 (2011-06-11 06:21) . . . . .	551
All sparks die out in the end (2011-06-14 18:56) . . . . .	552
My generation would have done it differently (2011-06-19 07:32) . . . . .	552
I won't make a hash of that (2011-06-19 21:03) . . . . .	552
<b>July . . . . .</b>	<b>554</b>
Please keep your focus! (2011-07-06 17:31) . . . . .	554
You can catch lots in your .NET (2011-07-09 13:23) . . . . .	555
Clojure becomes more portable (2011-07-24 06:48) . . . . .	556
Please mind the gap to the platform (2011-07-26 14:41) . . . . .	557
Camera! Action! (2011-07-31 07:12) . . . . .	558
<b>August . . . . .</b>	<b>559</b>
Pieces of eight (2011-08-09 16:46) . . . . .	559
I'll need to reflect on that (2011-08-10 16:51) . . . . .	560
No live animals here (2011-08-11 16:53) . . . . .	561
This C is Kilim me (2011-08-12 16:55) . . . . .	561
You say tomato, I say tomato (2011-08-13 17:04) . . . . .	562
V1, V2, V3, V4, ... (2011-08-14 17:06) . . . . .	562
What do you attribute that to? (2011-08-17 08:00) . . . . .	562
Follow the trail to the source (2011-08-23 08:00) . . . . .	563
<b>September . . . . .</b>	<b>563</b>
Behind you! (2011-09-04 08:16) . . . . .	563
You made me... now tell me what to do (2011-09-05 17:54) . . . . .	564
Down to the MIL (2011-09-05 21:50) . . . . .	565
What type of event were you expecting? (2011-09-07 22:10) . . . . .	570
Ask your Dad! (2011-09-10 18:29) . . . . .	573
COM is coming home (2011-09-21 15:51) . . . . .	574
It's all about the Type of the data (2011-09-22 19:50) . . . . .	579
<b>October . . . . .</b>	<b>580</b>
Anything you can do, I can do too. (2011-10-03 10:43) . . . . .	580
What a Functionally Perfect Day! (2011-10-19 08:53) . . . . .	581
Catch it if you can (2011-10-20 08:56) . . . . .	583
There's no debugger for this situation (2011-10-21 08:57) . . . . .	584
Tell Roslyn I've been waiting for her (2011-10-22 20:35) . . . . .	584
An Englishman's home is his castle (2011-10-23 23:16) . . . . .	588
Something for the weekend? (2011-10-30 22:17) . . . . .	589
<b>November . . . . .</b>	<b>590</b>
Small is beautiful...and fast... and obfuscated... and (2011-11-23 21:00) . . . . .	590
That's all I got from my inheritance? (2011-11-24 07:00) . . . . .	592
<b>December . . . . .</b>	<b>597</b>
I'll take a bite out of the existing code (2011-12-04 08:41) . . . . .	597
That was a lightening quick introduction! (2011-12-11 22:03) . . . . .	597
Please don't call, I'm not ready (2011-12-12 09:11) . . . . .	598
<b>2012</b>	<b>601</b>
<b>January . . . . .</b>	<b>601</b>
Someone needs to check the plumbing (for leaks) (2012-01-01 16:22) . . . . .	601
Bridge the gap! (2012-01-04 16:44) . . . . .	604
Let's see how you run (2012-01-07 09:50) . . . . .	604
Crystal balls out (2012-01-10 10:00) . . . . .	609

Exceptional behaviour- I think not! (2012-01-12 13:25) . . . . .	609
Lose the fat-it's slowing you down (2012-01-19 17:44) . . . . .	610
To the future and beyond! (2012-01-19 17:44) . . . . .	611
You've got loads of things in that container (2012-01-22 12:48) . . . . .	611
I'm still standing (2012-01-29 11:40) . . . . .	613
It's not a game you know (2012-01-30 05:00) . . . . .	613
February . . . . .	617
Watch out for the ceiling! (2012-02-14 20:43) . . . . .	617
To the future, today! (2012-02-19 08:33) . . . . .	621
Keep on searching to get your reward (2012-02-20 06:34) . . . . .	621
It's all Greek to me (2012-02-20 08:36) . . . . .	622
Is there an MD in the house? (2012-02-22 09:20) . . . . .	623
The more the merrier! (2012-02-24 09:18) . . . . .	624
March . . . . .	624
Never the twain (2012-03-01 20:17) . . . . .	624
It's all in the details (2012-03-04 10:52) . . . . .	624
Get to the heart of the matter (2012-03-04 15:09) . . . . .	626
Sorry, I gave you the wrong result last time (2012-03-25 15:24) . . . . .	630
I didn't mean to cause any disruption (2012-03-26 07:27) . . . . .	631
Money for nothing and your JITs for free (2012-03-27 07:32) . . . . .	631
April . . . . .	632
Let's do this scientifically (2012-04-01 19:21) . . . . .	632
You're so static (2012-04-02 19:22) . . . . .	633
Not your prototypical podcast (2012-04-03 19:23) . . . . .	633
A little Reflection can solve most problems (2012-04-04 19:25) . . . . .	634
In the place through which we wander (2012-04-23 07:00) . . . . .	635
There's nothing to see here (2012-04-25 07:00) . . . . .	636
Let's get it going (2012-04-27 07:00) . . . . .	637
May . . . . .	637
Entomology for beginners (2012-05-04 10:42) . . . . .	637
Surely performant isn't a word (2012-05-08 06:47) . . . . .	638
The sooner the better (2012-05-13 11:50) . . . . .	639
That's magic, that is (2012-05-13 16:53) . . . . .	640
You're doing it upside down (2012-05-14 06:24) . . . . .	641
Thanks for the memory (2012-05-30 06:09) . . . . .	642
June . . . . .	643
Phone me! (2012-06-14 05:00) . . . . .	643
Two for the price of one (2012-06-15 05:48) . . . . .	646
All together now (2012-06-16 05:46) . . . . .	647
Please be discrete (2012-06-17 05:04) . . . . .	647
Sweet seventeen (2012-06-18 05:08) . . . . .	648
There can only be one winner (2012-06-20 05:09) . . . . .	651
Back you go! (2012-06-22 05:11) . . . . .	651
July . . . . .	652
Are you still there? (2012-07-30 05:24) . . . . .	652
August . . . . .	654
Did that get a reaction? (2012-08-08 09:18) . . . . .	654
You've got that backwards (2012-08-25 06:16) . . . . .	654
Put down the tools (2012-08-28 05:10) . . . . .	655
September . . . . .	658

I can see CLRly now the reign has gone (2012-09-03 05:32) . . . . .	658
Does it compute? (2012-09-11 05:18) . . . . .	659
I'll get back to you sometime in the future (2012-09-13 05:20) . . . . .	660
October . . . . .	660
It's certainly a type of script (2012-10-02 17:09) . . . . .	660
It's more about async than you might think (2012-10-03 06:10) . . . . .	660
Another one bites the dust (2012-10-15 06:01) . . . . .	661
Abort, abort ... no way! (2012-10-16 06:12) . . . . .	661
Use the source Luke (2012-10-17 06:34) . . . . .	662
Very professional! (2012-10-18 05:45) . . . . .	663
When I go, I'm going to take it with me (2012-10-19 06:07) . . . . .	667
It's the future (but not as we know it) (2012-10-21 06:47) . . . . .	670
It's certainly extensible (2012-10-22 06:49) . . . . .	670
Safe for parallelism (2012-10-31 07:10) . . . . .	671
November . . . . .	672
Things are a'changing (2012-11-01 07:11) . . . . .	672
That's one heck of a virtual machine (2012-11-02 07:23) . . . . .	672
I'm busy! (2012-11-03 07:02) . . . . .	673
That does require some interpretation (2012-11-05 07:35) . . . . .	673
I'd rather not do all of that again (2012-11-14 07:22) . . . . .	674
December . . . . .	674
That's the right script for the play (2012-12-11 07:28) . . . . .	674
Single page apps gone mad (2012-12-12 07:31) . . . . .	678
Bind it tight (2012-12-17 07:09) . . . . .	680
I came to find out about the barrier, and found RCU (2012-12-19 07:10) . . . . .	682
So many architectures, so little time (2012-12-21 07:11) . . . . .	682
<b>2013</b>	<b>685</b>
January . . . . .	685
I brought the glue for the model (2013-01-14 07:23) . . . . .	685
Give us a job (2013-01-16 07:25) . . . . .	686
Something for Hercules to do later (2013-01-18 07:28) . . . . .	686
Get taller (2013-01-23 07:31) . . . . .	687
I'd like a new kernel please (2013-01-25 07:32) . . . . .	688
It's hard to get across the border (2013-01-28 07:40) . . . . .	689
I want to see what's behind the window (2013-01-30 06:37) . . . . .	689
February . . . . .	690
No white pyjamas here (2013-02-19 07:28) . . . . .	690
Just a few things to make it more efficient (2013-02-21 07:30) . . . . .	691
Press the meta key (2013-02-22 07:08) . . . . .	691
Smoke and mirrors - well, reflection anyway (2013-02-25 07:34) . . . . .	694
Unbounded storage - how much? (2013-02-25 08:36) . . . . .	694
March . . . . .	695
If the skies are grey, read something else (2013-03-04 07:01) . . . . .	695
That's not what you said earlier (2013-03-21 07:36) . . . . .	696
That's a very sharp solution (2013-03-25 07:10) . . . . .	696
April . . . . .	697
I remember when multi-threaded was only on a single core (2013-04-08 06:16) . . . . .	697
Ever onward (2013-04-10 06:54) . . . . .	698
Send the signal (2013-04-12 06:28) . . . . .	698

Short and sweet (2013-04-22 06:09) . . . . .	699
Follow that up (2013-04-23 06:11) . . . . .	701
May . . . . .	702
Bits 'n' pieces (2013-05-13 05:47) . . . . .	702
Windbg and the case of the unfreed garbage (2013-05-30 06:39) . . . . .	703
MLWorks—yes it does! (2013-05-31 06:40) . . . . .	707
June . . . . .	708
Stop darting about (2013-06-03 06:40) . . . . .	708
That's too complex for me (2013-06-05 05:57) . . . . .	709
No change here (2013-06-06 05:11) . . . . .	710
Where I come from (2013-06-10 05:33) . . . . .	712
Stunned—I could hardly get going again (2013-06-13 05:23) . . . . .	713
Here's your script (2013-06-20 05:46) . . . . .	714
July . . . . .	715
Time for a rest (2013-07-12 05:12) . . . . .	715
Where worlds collide (2013-07-15 05:38) . . . . .	716
Go...go...go... but only at the right time (2013-07-24 05:40) . . . . .	717
Haswell, will end well? (2013-07-26 05:41) . . . . .	721
More, more, more (2013-07-29 05:44) . . . . .	721
Don't store this for too long (2013-07-31 05:47) . . . . .	723
August . . . . .	725
I say container, you say no brainer (2013-08-02 05:48) . . . . .	725
Almost written at the same time (2013-08-05 05:36) . . . . .	725
That's not my tactic (2013-08-06 05:37) . . . . .	726
It's as sharp as it has always been (2013-08-08 05:41) . . . . .	726
What type of theory? (2013-08-09 05:40) . . . . .	727
You need to be clear with your instructions (2013-08-12 05:42) . . . . .	728
Blast from the past (2013-08-14 05:42) . . . . .	729
That's some art (2013-08-16 05:43) . . . . .	729
Exceptionally good post (2013-08-19 05:56) . . . . .	729
Dead or Alive? (2013-08-21 05:56) . . . . .	730
How clear is the proof? (2013-08-23 05:56) . . . . .	731
September . . . . .	732
Surface Detail does Matter (2013-09-06 06:16) . . . . .	732
I'll use multiple cores, but not for long (2013-09-10 06:35) . . . . .	733
I'm flagging (2013-09-30 06:22) . . . . .	735
October . . . . .	736
When you leave, I leave too (2013-10-02 05:13) . . . . .	736
Something for the weekend (2013-10-04 05:15) . . . . .	738
It's not what you do, it's the way that you use it (2013-10-10 05:06) . . . . .	738
I didn't know that! (2013-10-14 06:12) . . . . .	739
Debug the debugger (2013-10-21 05:21) . . . . .	740
To the cloud (2013-10-25 06:18) . . . . .	742
Not essential, but useful anyway (2013-10-28 06:19) . . . . .	743
Get inside (2013-10-30 05:22) . . . . .	744
November . . . . .	745
Keep it simple (2013-11-01 05:22) . . . . .	745
That's not what I expected (2013-11-03 10:20) . . . . .	745
I'm a little blue today (2013-11-04 10:22) . . . . .	746
Puzzling (2013-11-18 06:51) . . . . .	746

This language has more than one dimension (2013-11-25 06:04) . . . . .	747
That's no barrier to my understanding (2013-11-27 07:07) . . . . .	749
December . . . . .	749
It all seems very functional (2013-12-02 06:30) . . . . .	749
Meshing ideas (2013-12-09 06:45) . . . . .	750
<b>2014</b>	<b>755</b>
January . . . . .	755
Sharp after all these years (2014-01-06 06:51) . . . . .	755
You don't have to be a compiler to manipulate a syntax tree (2014-01-13 06:52)	756
One at a time (2014-01-16 06:56) . . . . .	757
It's all about the script (2014-01-20 06:21) . . . . .	758
Choose wisely (2014-01-20 06:57) . . . . .	759
You GIT (2014-01-23 06:59) . . . . .	760
It will be my legacy (2014-01-28 06:01) . . . . .	760
Use the (Windows) source Luke (2014-01-30 06:24) . . . . .	761
February . . . . .	762
Cast the fun (2014-02-04 06:58) . . . . .	762
Plug table into light (2014-02-18 07:22) . . . . .	763
Should I do them concurrently or in parallel? (2014-02-24 07:06)	766
To infinity from far away (2014-02-27 07:07) . . . . .	768
Where next? (2014-02-28 07:08) . . . . .	769
March . . . . .	769
It's off to war we go (2014-03-31 06:17) . . . . .	769
April . . . . .	769
That's for sure (2014-04-04 05:29) . . . . .	769
Some odds and ends (2014-04-28 05:52) . . . . .	771
May . . . . .	772
Hello source, meet filter (2014-05-07 05:31) . . . . .	772
It's certainly an oK runtime (2014-05-15 05:04) . . . . .	772
A nugget of information (2014-05-19 05:39) . . . . .	772
June . . . . .	774
You need a load of extra functionality to support that (2014-06-02 05:18) . . . . .	774
July . . . . .	776
Deep, but friendly (2014-07-08 05:49) . . . . .	776
Evolution is the key (2014-07-10 05:57) . . . . .	777
I've told you several times, but all at the same time (2014-07-15 05:06) . . . . .	778
Odds but not ends (2014-07-17 06:04) . . . . .	779
It's not the parsing, but a question of semantics (2014-07-22 06:10) . . . . .	780
To the bottom of the sea (2014-07-24 06:19) . . . . .	782
August . . . . .	782
Device drivers step by step (2014-08-05 05:32) . . . . .	782
See more sharp things (2014-08-19 06:30) . . . . .	784
Stop interrupting me (2014-08-21 06:31) . . . . .	786
A marvellous miscellany (2014-08-26 07:30) . . . . .	787
September . . . . .	788
Sometimes your worker role needs to run server GC (2014-09-02 07:18) . . . . .	788
What does better mean for overloads? (2014-09-03 06:21) . . . . .	791
A good read on concurrency in general (and Java in particular) (2014-09-04 06:33)	792
Making your C# more effective (2014-09-09 06:43) . . . . .	793

csc generates code as well you know (2014-09-19 06:53) . . . . .	794
The best explanation of cache coherency I've seen (2014-09-30 06:48) . . . . .	796
October . . . . .	796
Some interesting bits and pieces (2014-10-02 06:51) . . . . .	796
TPL Dataflow still has its uses (2014-10-14 06:27) . . . . .	797
If you use R# to code generate, make sure to regenerate if things change (2014-10-21 06:42) . . . . .	798
December . . . . .	801
Homomorphic encryption could make it easier to use cloud computing power (in the future) (2014-12-08 09:13) . . . . .	801
Distributed computing is another world! (2014-12-11 07:42) . . . . .	801
The source (or a variant) is often available (2014-12-12 08:20) . . . . .	802
It's the season to watch loads of videos (2014-12-22 06:46) . . . . .	803
<b>2015</b>	<b>805</b>
January . . . . .	805
Some useful distributed algorithms (2015-01-12 06:44) . . . . .	805
Writing high-performance .NET code can be hard (2015-01-14 06:25) . . . . .	806
Category theory seems to pop up everywhere (2015-01-16 06:21) . . . . .	807
Understanding the CLR via its assembly code (2015-01-19 06:39) . . . . .	808
JavaScript Ninja Secrets (2015-01-21 06:43) . . . . .	810
February . . . . .	811
You can have your own CLR too (2015-02-16 08:04) . . . . .	811
April . . . . .	812
Time Reborn (2015-04-14 06:59) . . . . .	812
F# Deep Dives (2015-04-14 07:46) . . . . .	812
So how do you patch a deployed Nuget package (2015-04-16 07:06) . . . . .	813
May . . . . .	814
Effective Modern C++ (2015-05-06 07:08) . . . . .	814
Some low level locking papers (2015-05-19 07:23) . . . . .	814
June . . . . .	815
So what happened at //build (2015-06-03 07:14) . . . . .	815
/dev/summer had some good talks as usual (2015-06-30 07:00) . . . . .	818
July . . . . .	819
Advanced Topics In Types and Programming Languages (2015-07-09 06:47) . . . . .	819
C# await inside catch and finally leads to some interesting semantics (2015-07-22 06:30) . . . . .	820
August . . . . .	822
Some UWP and a little ClojureScript (2015-08-03 06:17) . . . . .	822
CLR Load contexts can make things confusing (2015-08-05 06:17) . . . . .	823
Modern C++ Design: magic with templates (2015-08-07 06:05) . . . . .	826
Adaptive Code Via C#: Agile coding with design patterns and SOLID principles (2015-08-31 07:17) . . . . .	827
October . . . . .	828
Some interesting .NET videos (2015-10-19 08:01) . . . . .	828
November . . . . .	829
Some good reads on Angular 1.x (2015-11-24 07:31) . . . . .	829
December . . . . .	831
Sometimes it's not good to build on other libraries (2015-12-24 08:03) . . . . .	831

<b>2016</b>		<b>833</b>
February	.....	833
Actors have become popular again (2016-02-02 07:00)	.....	833
Another good DevWinter (2016-02-04 07:07)	.....	834
Your .NET applications can start being more native (2016-02-08 07:30)	.....	836
What do I need to know about Bitcoin? (2016-02-22 07:27)	.....	843
Just how serious is a bug (2016-02-29 07:36)	.....	846
March	.....	847
Vagrant, up and running (2016-03-07 07:42)	.....	847
May	.....	848
Switching to Angular 2 (2016-05-09 07:32)	.....	848
November	.....	850
At last you can prove your understanding of the C# specification (2016-11-14 07:01)	.....	850
<b>2017</b>		<b>853</b>
January	.....	853
A dump has so many uses (2017-01-02 06:57)	.....	853
'Tis the season for loads of Haskell (2017-01-04 07:20)	.....	855
October	.....	858
Some books since last time (2017-10-28 18:57)	.....	858
November	.....	860
Bitcoin - where does it go next? (2017-11-27 07:07)	.....	860
How do you nest your virtualization? (2017-11-29 07:07)	.....	861
December	.....	862
Some what types of reflection are there? (2017-12-12 07:03)	.....	862
<b>2018</b>		<b>863</b>
March	.....	863
Some recent C# learnings (2018-03-06 08:40)	.....	863
Not sure I love it, but I do understand it better (2018-03-14 06:47)	.....	865
April	.....	866
C# fun with structs (2018-04-03 05:47)	.....	866
Dependent Types (2018-04-05 05:47)	.....	870
Modern JavaScript for the win (2018-04-10 05:47)	.....	871
Progressive Web Applications (2018-04-16 05:57)	.....	874
Stack allocated closures make it into C# (2018-04-17 05:47)	.....	876
Reactive extensions in action (2018-04-18 05:06)	.....	880
Let's get started with Docker (2018-04-19 05:14)	.....	883
June	.....	889
What are micro-tasks in the browser all about? (2018-06-18 06:27)	.....	889
August	.....	889
Kubernetes is the winner (2018-08-07 20:59)	.....	889
Designing system for scalability (2018-08-10 06:07)	.....	890
October	.....	891
Reliable, Scalable and Maintainable Systems (2018-10-02 18:46)	.....	891
November	.....	892
Some more reading (2018-11-12 06:45)	.....	892
Of CORS, it's easy to test (2018-11-14 06:44)	.....	893
I'm hooked (2018-11-26 06:27)	.....	894

<b>2019</b>		<b>897</b>
January	.....	897
Some odds and ends (2019-01-14 06:21)	.....	897
February	.....	899
And some more books (2019-02-03 07:43)	.....	899
March	.....	900
More things for using K8s for real (2019-03-21 07:45)	.....	900
April	.....	901
More miscellaneous books (2019-04-29 06:54)	.....	901
<b>2020</b>		<b>903</b>
March	.....	903
It's been a while (2020-03-16 06:45)	.....	903
August	.....	904
Some Big Data reading (2020-08-23 15:11)	.....	904
September	.....	906
System Design Interviews (2020-09-01 06:51)	.....	906
OAuth 2 in Action (2020-09-02 07:21)	.....	907
Haskell's Type System is really amazing (2020-09-03 07:04)	.....	908
And a final batch of books (2020-09-07 07:03)	.....	909
October	.....	911
That's very unlikely (2020-10-25 15:28)	.....	911
<b>2021</b>		<b>913</b>
January	.....	913
Streaming Systems (2021-01-10 07:49)	.....	913
Data Science from scratch (2021-01-11 07:52)	.....	914
That's the way to do it (2021-01-12 07:55)	.....	915
Links aplenty (2021-01-13 07:08)	.....	916
Effective STL (2021-01-14 07:28)	.....	917
MapReduce for the win (2021-01-25 06:23)	.....	918
February	.....	919
Getting up to speed on K8s (2021-02-15 06:45)	.....	919
April	.....	921
Some more books (2021-04-05 18:34)	.....	921
And a little gRPC (2021-04-18 07:10)	.....	922
And yet more links (2021-04-19 07:26)	.....	923
May	.....	924
Processing unbounded data with Sql like languages (2021-05-24 05:52)	.....	924
June	.....	925
Let's Flink about it some more (2021-06-21 06:03)	.....	925
Stream Processing with Apache Flink (2021-06-28 06:29)	.....	926
A few more books (2021-06-30 06:23)	.....	927
September	.....	929
Develop C# as if it were a dynamic language? (2021-09-13 07:20)	.....	929
And a summer of reading (2021-09-15 06:49)	.....	929
October	.....	931
Some Kafka and some algorithms (2021-10-18 06:02)	.....	931
November	.....	933
And some holiday reading (2021-11-07 07:47)	.....	933

<b>2022</b>		<b>937</b>
January . . . . .		937
Static and Abstract? (2022-01-03 07:10) . . . . .		937
February . . . . .		938
Dynamic and profile guided (2022-02-27 05:57) . . . . .		938
Python for the win (2022-02-28 06:54) . . . . .		938
March . . . . .		940
Getting to grips with Quantum Computing (2022-03-14 06:55) . . . . .		940
Microsoft Refresh (2022-03-20 06:37) . . . . .		941
Secure your containers (2022-03-21 06:51) . . . . .		941
What are the chances of that? (2022-03-23 06:55) . . . . .		942
Ceph in detail (2022-03-28 06:21) . . . . .		943
April . . . . .		943
Software Architecture: It's hard! (2022-04-04 06:27) . . . . .		943
May . . . . .		944
Another set of books (2022-05-16 06:25) . . . . .		944
And yet more (2022-05-18 06:17) . . . . .		945
Distributed system distilled (2022-05-23 06:36) . . . . .		947
July . . . . .		948
Some C# odds and ends (2022-07-04 06:38) . . . . .		948
Inside Facebook, and API security (2022-07-11 06:20) . . . . .		949
October . . . . .		950
And some more links (2022-10-24 07:13) . . . . .		950
<b>2023</b>		<b>953</b>
July . . . . .		953
What is it if it isn't SQL? (2023-07-03 07:25) . . . . .		953
September . . . . .		953
Hello World (again) (2023-09-04 07:09) . . . . .		953

# 2007

## 1.1 December

### 1.1.1 Common Lisp - thanks for the good times! (2007-12-13 10:47)

Well I've been meaning to start blogging for a while now. It's mainly because everyone else is doing it, and although my contribution won't be great, I can use it as a place to note down interesting articles and observations that I find. However, I had kept putting it off, because the postings are never quite good enough. But I've now realised that my writing style can only improve with practice, so here we go.

I was hoping to start by doing a series of quick observations on what I like about Common Lisp and why I think it is so great. My motivation for this is that, sadly, I'm doing less and less Common Lisp in my every day programming, and in the near future I'll be moving to a company that doesn't do any Lisp at all, so I'd like to take the opportunity to detail some of the good things about the language.

Common Lisp and me, we go back a long way.

I first heard about Lisp in the early 1981-1982 in one of the PC magazines of the time, PC world I think it was, and then bought a copy to run on the BBC micro. The language was interesting as it was all about manipulating symbols and lists and seemed closely tied to interesting things like AI.

I took a year industrial placement as part of a sponsorship from the computer company [1]ICL, and during this time I started learning C in my spare time. My first fun project in this new

language, was to write a simple Lisp interpreter. This was nothing fancy. The reader was implemented in C and only dealt with lists and atoms that were symbols. The memory consisted of a few thousand cells, where a cell was a C union type that was either a symbol (with a name of up to 20 characters) or a cons cell, and a simple mark-and-sweep garbage collector that could free up some space. The neat thing was that I only had to implement a simple evaluator in C for EVAL/APPLY, and expose a few C implemented primitives into the Lisp language, CAR/CDR/CONS/IF/ATOM/EQ/SETQ/LAMBDA and I could write some quite interesting programs.

The idea of a language with a small number of built in features that could be built up into something much more powerful really appealed to me.

After university, where I studied for a degree in mathematics and then did a postgraduate diploma in [2]Computer Science, I joined [3]Harlequin Ltd , to work mainly on the LispWorks Common Lisp programming environment though with a year here and there spent working on an ML compiler and programming environment. When Harlequin got taken over by Global Graphics in 1999, I moved to Scientia Ltd, a software company who's scheduling software was written in Common Lisp, and where I implemented exclusively in Common Lisp until about 2002, when C # and the CLR started becoming a focus.

Common Lisp has paid my bills for nearly 20 years and I've very grateful for that. As a language I think it's very much the quiet one in the corner, the language with lots of clever ideas (such as first class functions, objects, internal domain specific languages via macros, reflection), a language that doesn't push itself forward, but which contains a great deal of very deep ideas.

1. [http://en.wikipedia.org/wiki/International\\_Computers\\_Ltd](http://en.wikipedia.org/wiki/International_Computers_Ltd)
2. [http://en.wikipedia.org/wiki/Cambridge\\_Diploma\\_in\\_Computer\\_Science](http://en.wikipedia.org/wiki/Cambridge_Diploma_in_Computer_Science)
3. [http://en.wikipedia.org/wiki/Harlequin\\_\(software\\_company\)](http://en.wikipedia.org/wiki/Harlequin_(software_company))

### 1.1.2 A note on laziness (2007-12-13 11:46)

During my placement time at ICL, I got to work with a number of [1]functional programming languages. During my initial industrial placement year I got interested in [2]Lispkit Lisp, and during later summer and xmas vacations I got to work on the Alvey Flagship Project, a project with the goal of running functional programming languages efficiently by evaluating parts of the computation in parallel using multiple off the shelf processors. This introduced me to the functional programming language [3]Hope. The implementation of such languages was really interesting to me and the [4]Simon Peyton Jones book (at the time) was something I found very interesting.

One beautiful thing about these languages was that they had subsets that were lazily evaluated. [5]LispKit Lisp was lazily evaluated and by using lazy lists people had even considered the possibility of writing an operating system in the functional language. I guess that rather than an operating system, we'd call it a platform these days, as there was no intention of going as low as the device driver level. [It seems to me that this work is being taken up these days in projects like the [6]Singularity operating system that Microsoft Research are writing] The idea was that events could be represented to the kernel via streams (lazy lists) with lists being fused together using non-deterministic stream merging operations to allow a single function to process multiple sources without blocking.

I've recently been looking at [7]F #. This language has lazy streams built in to it, and it was a joy to try to implement some of the old lazy functional programming language algorithms that I had seen 20 years ago, in other languages like KRC and SASL.

Here's the [8]Sieve of Eratosthenes, implemented in F #. To me, the beauty is that I can write something that returns a collection of items of unbounded size, and the system will only do as much work as is necessary to calculate items that I need in future computations. I don't have to worry about the size of the Sieve. In other languages, the Sieve is often implemented as an array, so if we need N primes, we have to work out how big to make it to ensure that we will have N primes remaining after the algorithm. This kind of thing just complicates the code, making it a lot harder to see if the code is returning the correct results.

```
> // Check for m not being a multiple on n  
- let notmultiple n m = m - n * (m / n) <> 0I ;;
```

```
val notmultiple : Math.BigInt -> bigint -> bool
```

```
> // Generate a lazy of list of all positive integers
- let rec integersFrom n = LazyList.cons n (LazyList.delayed (fun _ -> integersFrom (n+1)));;
```

```
val integersFrom : bigint -> bigint LazyList
```

```
> LazyList.to _array (LazyList.take 30 (integersFrom 1));;
val it : bigint array
= [|1I; 2I; 3I; 4I; 5I; 6I; 7I; 8I; 9I; 10I; 11I; 12I; 13I; 14I; 15I; 16I; 17I;
18I; 19I; 20I; 21I; 22I; 23I; 24I; 25I; 26I; 27I; 28I; 29I; 30I|]
```

```
> let rec sieve ll =
- match LazyList.get ll with
- | None _ -> LazyList.empty ()
- | Some (hd, rest) -> LazyList.cons hd (LazyList.delayed (fun _ -> (sieve (LazyList.filter (not-
multiple hd) rest))));;
```

```
val sieve : bigint LazyList -> bigint LazyList
```

```
> // Generate the stream of primes
- let primes = sieve (integersFrom 2);;
```

```
val primes : bigint LazyList
```

```

> // Let's have a look at the first 20
- LazyList.to_array (LazyList.take 20 primes);;
val it : bigint array
= [|2I; 3I; 5I; 7I; 11I; 13I; 17I; 19I; 23I; 29I; 31I; 37I; 41I; 43I; 47I; 53I; 59I; 61I; 67I; 71I|]

```

It was a joy to find that F # also supports unbounded integers, though by default, like C # and lots of languages from the C family, it defaults to silently wrapping on overflow. In this age, where the emphasis is more on correctness and productivity than speed, the few instructions needed to check for overflow seem a small price to me.

```

> 2000000000 * 4;;
val it : int = -589934592
> 2000000000I * 4I;;
val it : Math.BigInt = 8000000000I
>

```

1. [http://en.wikipedia.org/wiki/Functional\\_programming\\_languages](http://en.wikipedia.org/wiki/Functional_programming_languages)
  2. [http://en.wikipedia.org/wiki/Lispkit\\_Lisp](http://en.wikipedia.org/wiki/Lispkit_Lisp)
  3. [http://en.wikipedia.org/wiki/Hope\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Hope_(programming_language))
  4. <http://research.microsoft.com/~simonpj/papers/slpj-book-1987/index.htm>
  5. [http://www.amazon.co.uk/Functional-Programming-Implementation-Prentice-Hall-international/dp/0133315797/ref=sr\\_11\\_1?ie=UTF8&qid=1197537909&sr=11-1](http://www.amazon.co.uk/Functional-Programming-Implementation-Prentice-Hall-international/dp/0133315797/ref=sr_11_1?ie=UTF8&qid=1197537909&sr=11-1)
  6. <http://research.microsoft.com/os/singularity/>
  7. <http://research.microsoft.com/fsharp/fsharp.aspx>
  8. [http://en.wikipedia.org/wiki/Sieve\\_of\\_Eratosthenes](http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes)
- 

### **1.1.3 A little extra (2007-12-13 13:22)**

And just for completeness, the other classic stream example.

```

> // Define a function to combine two successive elements in a list
- let rec nextFib (x : bigint LazyList) =
- LazyList.consf (LazyList.hd x + LazyList.hd (LazyList.tl x)) (fun _ -> nextFib (LazyList.tl x));;

val nextFib : bigint LazyList -> Math.BigInt LazyList

> // Now apply to the list with first two elements 1
- let rec foo _ =
- LazyList.cons 1I (LazyList.consf 1I (fun _ -> nextFib (foo ())));;

val foo : unit -> bigint LazyList

> // This is the Fibonacci sequence
- let fibonacci = foo();;

val fibonacci : bigint LazyList

> LazyList.to _array (LazyList.take 30 fibonacci);;
val it : bigint array
= [|1I; 1I; 2I; 3I; 5I; 8I; 13I; 21I; 34I; 55I; 89I; 144I; 233I; 377I; 610I;
987I; 1597I; 2584I; 4181I; 6765I; 10946I; 17711I; 28657I; 46368I; 75025I;
121393I; 196418I; 317811I; 514229I; 832040I|]
>
```

---

#### 1.1.4 Let's tie the knot (2007-12-13 21:12)

The performance of the last example was dreadful; you can see this by considering how many LazyList objects get allocated. A better technique to use is called "tying the knot". This uses a mutable reference cell to feed the sequence back in to itself.

In F# again, using the LazyList data type.

```
// Add the consecutive pairs lazily to get the next element
```

```
> let rec nextFib (x : bigint LazyList) =  
- LazyList.consf (LazyList.hd x + LazyList.hd (LazyList.tl x)) (fun _ -> nextFib (LazyList.tl x));;
```

```
val nextFib : bigint LazyList -> Math.BigInt LazyList
```

```
> let fiblist : bigint LazyList ref = ref (LazyList.empty () )  
- in  
- let fibonacci = LazyList.cons 1I (LazyList.consf 1I (fun _ -> nextFib !fiblist))  
- in  
- fiblist := fibonacci;  
- fibonacci ;;  
val it : bigint LazyList = seq [1I; 1I; 2I; 3I; ...]
```

```
> let fibs = it;;
```

```
val fibs : bigint LazyList
```

```
> LazyList.to _array (LazyList.take 20 fibs);;  
val it : bigint array  
= [|1I; 1I; 2I; 3I; 5I; 8I; 13I; 21I; 34I; 55I; 89I; 144I; 233I; 377I; 610I;  
987I; 1597I; 2584I; 4181I; 6765I|]
```

---

### 1.1.5 Let's start at the beginning (2007-12-17 10:41)

I was hoping to use this blog to note down computing books that I have read together with a quick review. I read this book some time ago, but it was interesting nevertheless and deserves a mention.

## [1]Beginning SQL Server 2005 Programming

This was a good overview of SQL Server 2005. It seemed to cover most things, though not in the depth I would have liked. There is a follow up book (Professional SQL Server 2005 Programming) which probably goes into more depth and which I will read soon. I liked the emphasis on using management studio and learned about some new things like the existence of indexed (materialised) views. The only oddity was that it mentions TRY/CATCH blocks, claiming that they will be discussed later in the presentation, but I don't believe they are.

On a kind of related note, this recent [2]channel9 video explained some of the aspects of the Entity framework very well.

1. [http://www.amazon.co.uk/BEGINNING-SERVER-SMALL-PROGRAMMING-PROGRAMMER/dp/0764584332/ref=sr\\_1\\_1?ie=UTF8&search-term=beginning+sql+server+2005&qid=1197885276&sr=1-1](http://www.amazon.co.uk/BEGINNING-SERVER-SMALL-PROGRAMMING-PROGRAMMER/dp/0764584332/ref=sr_1_1?ie=UTF8&search-term=beginning+sql+server+2005&qid=1197885276&sr=1-1)
  2. <http://channel9.msdn.com>ShowPost.aspx?PostID=366232#366232>
-

# 2008

## 2.1 January

### 2.1.1 Windows Presentation Foundation Unleashed (2008-01-02 14:12)

Over the Xmas break I read [1]Windows Presentation Foundation Unleashed by Adam Nathan.

An excellent read with many useful tips. Well organised and very clear. Many times when I thought up a question, it was answered almost straightaway a paragraph or two later in the text. "Digging deeper" sidebars often made good points and illuminated parts of the text. Many useful examples, such as simple example demonstrating visual studio type pinning of a number of panes, showing how three layers of grid layout could work well together. Very good coverage of the XAML to objects translation in chapter two, and a clear explanation of the dependency properties model and the flow of events in chapter three.

WPF looks like a great way of writing GUIs in a declarative style, using triggers to handle the interaction between elements, though with the ability to drop into code-behind when the need arises. I need to spent time rereading and working through all of the examples to get more to grips with it.

Thinking back, I don't remember any mention of drag and drop which might be required in some rich client applications.

1. <http://www.amazon.co.uk/Windows-Presentation-Foundation-Unleashed-WPF/dp/0672328917>

---

### 2.1.2 What's my name again? (2008-01-04 09:51)

Naming of functions is interesting in that it differs widely between languages.

C, for example, gets rid of most names as they are local to the unit of compilation. For example, if I take the following small program, and look at the result of the compilation using dumpbin to look within the COFF format executable:

```
void DoPrinting()
{
    printf("Hello World!\n");
}
```

```
int main(int argc, char* argv[])
{
    DoPrinting();
    return 0;
}
```

The following shows that the name is not preserved  
> dumpbin /all play.exe | grep DoPrinting

If I annotate the DoPrinting function with `_declspec(dllexport)`, the name is preserved as an export to allow later linking:

```
> dumpbin /exports play.exe | grep DoPrinting  
1 0 0000100A [1]?DoPrinting@@YAXXZ
```

Moving on to C #, this language offers reflection which allows the dynamic runtime discovery of methods and other types of object.

```
using System.Reflection;  
using System;  
namespace Fred  
{  
    class Foo  
    {  
        static void Main()  
        {  
            Type thisType = typeof(Foo);  
            MethodInfo printMethod =  
                thisType.GetMethod("doPrinting",  
                    BindingFlags.Static | BindingFlags.InvokeMethod | BindingFlags.NonPublic);  
            printMethod.Invoke(null, null);  
        }  
  
        static void doPrinting()  
        {  
            System.Console.WriteLine("Hello world");  
        }  
    }  
}
```

In the IL metadata, we can find the name `doPrinting` associated with the name `Fred.Foo`, the name `Foo` in the namespace named `Fred`.

```
.method private hidebysig static void doPrinting() cil managed
{
// Code size 13 (0xd)
.maxstack 8
IL_0000: nop
IL_0001: ldstr "Hello world"
IL_0006: call void [mscorlib]System.Console::WriteLine(string)
IL_000b: nop
IL_000c: ret
} // end of method Foo::doPrinting
```

Common Lisp pushes the naming to the extreme representing a global name by an object called a symbol.

```
(defun main()
  (doPrinting))
```

```
(defun doPrinting()
  (format t "Hello World"))
```

Looking inside the image, we find the symbol doPrinting. We can use the in-built function describe to have a look at the attributes of a symbol.

```
CL-USER 3 > (describe 'doprinting)
```

```
DOPRINTING is a SYMBOL
NAME "DOPRINTING"
VALUE #<unbound value>
FUNCTION #<Function DOPRINTING 217846E2>
PLIST NIL
PACKAGE #<The COMMON-LISP-USER package, 9/16 internal, 0/4 external>
```

A couple of comments here before we get down to what was displayed. First, we are not looking at any kind of object file here - in the C case we were looking at an executable and in the C # case we were looking at an assembly; both of these are units of deployment. In the Lisp case, we have compiled the two function definitions and loaded them in to the running image. When we interact with the REPL (read-eval-print-loop) by typing "dopriniting", it is the Lisp reader, which converts the characters into a symbol object. Second, by default, any name that is read by the reader is upcased for lookup. Hence, doPrinting is converted into the symbol with the name DOPRINTING. I can force the case to be preserved by using escape characters, |, around the name to prevent the reader from doing anything to the case. The printer by default puts the escape characters back in when it prints the symbol, so that the same symbol will be read.

```
CL-USER 5 > '|doPrinting|
|doPrinting|
```

One last observation. Notice the ' before the name in 'doPrinting - the doPrinting symbol is said to be quoted. ' is handled specially by the reader, which reads the next item and emits an expansion that causes it not to be evaluated. If a symbol is not quoted, then it is going to be evaluated. It does this by parsing 'doPrinting into (QUOTE DoPrinting), where QUOTE is a special form that prevents its first argument being evaluated.

Looking at the result of the DESCRIBE call, we can see that it is as if the symbol type were defined as a standard structure

```
CL-USER 6 > (defstruct my-symbol name value function package plist)
MY-SYMBOL
```

```
CL-USER 7 > (describe (make-my-symbol))
```

```
#S(MY-SYMBOL :NAME NIL :VALUE NIL :FUNCTION NIL :PACKAGE NIL :PLIST NIL) is a MY-SYMBOL
NAME NIL
VALUE NIL
FUNCTION NIL
PACKAGE NIL
PLIST NIL
```

Common Lisp offers a make-symbol function for making the symbol, though this has a required parameter for the symbol name.

```
CL-USER 3 > (make-symbol "Fred")
#:|Fred|
```

There are accessors symbol-name, symbol-value, symbol-function, symbol-package and symbol-plist, for getting the values from the various slots of the symbol structure. symbol-value and symbol-function access the symbol's value and function definitions. Whereas in Scheme the function and value share a slot, in Common Lisp a symbol can have a function and value associated with it.

```
CL-USER 5 > (setf (symbol-function 'doPrinting) #'(lambda () (format t "Hello World")))
#<anonymous interpreted function 217225F2>
```

```
CL-USER 6 > (setf (symbol-value 'doPrinting) 20)
20
```

```
CL-USER 7 > (symbol-function 'doPrinting)
#<anonymous interpreted function 217225F2>
```

```
CL-USER 8 > (symbol-value 'doPrinting)
20
```

```
CL-USER 9 > (symbol-name 'doPrinting)
"DOPRINTING"
```

```
CL-USER 10 > (symbol-package 'doPrinting)
#<The COMMON-LISP-USER package, 9/16 internal, 0/4 external>
```

Symbols are grouped together into packages. Packages allow access to multiple symbols with the same name, typically by the reader. When some Lisp code is being read, there is always the notion of the current package which is maintained in the special variable \*package\*. When a symbol name is read, the name is looked up in the package which is the current value of

\*package\* (assuming the symbol has no explicit package prefix). A symbol name is prefixed by a package by separating the package name and symbol name by a single or double colon ie fred::bar denotes the symbol bar in the package fred, whereas fred denotes the symbol fred in the package \*package\*. The reader will create the symbol if it is not available, using the equivalent of the function intern

Packages are first-class objects. They can be created using the make-package function.

```
CL-USER 14 > (make-package "FRR")
#<The FRR package, 0/16 internal, 0/16 external>
```

```
CL-USER 15 > (make-symbol "xxx")
#:|xxx|
```

```
CL-USER 16 > (intern * **)
FRR::|xxx|
NIL
```

```
CL-USER 17 > (find-symbol "xxx" "FRR")
FRR::|xxx|
:INTERNAL
```

In the above example, \*/\*\*/\*\* are maintained by the REPL. \* refers to the value from the last evaluation, \*\* to the value from the previous evaluation and \*\*\* may be used for the value

before that. The last call, that to find-symbol shows something rather interesting. A symbol can be internal or external in a package. When accessed via the package:name form only external symbols are taken into account, whereas using a double colon like package::name causes internal and external symbols to be taken into account.

What is the point of external symbols?

Packages can inherit from other packages, making external symbols from the inherited package available in the package that does the inheriting.

Packages can be imported and exported from packages using import and export, but a much neater technique is to use the defpackage macro to create the package and set up its exports at the same time.

```
(defpackage :Fred  
  (:export "A" "B")  
  (:intern "C"))
```

```
(defpackage :George  
  (:use :fred)  
  (:export "A"))
```

Note that if the use list is not explicitly given, it defaults to a list of common packages.

This leads to the following behaviour by the reader.

```
CL-USER 2 > 'fred::a  
FRED:A
```

```
CL-USER 3 > 'fred::c  
FRED::C
```

```
CL-USER 5 > 'george:a  
FRED:A
```

```
CL-USER 6 > 'george::b  
FRED:B
```

The first defpackage expands in Procyon lisp to

```
((lambda (package)
  (dolist (pro::x '("A" "B"))
    (export (intern (string pro::x) package) package))
  (intern "C" package)
  package)
  (or (find-package ""Fred") (make-package ""Fred"))))
```

A typical technique is to use a list of defpackage forms early in the build to set up the package structure for externals of all packages, with the internal symbols set up later as the reader interns symbols as part of the compilation process. External symbols should only ever be set up in defpackage forms ie never use the EXPORT function to manually change the package structure.

Symbols can be uninterned from packages to remove this correspondence.

```
CL-USER 1 > (defpackage "A"  
  (:export "HELLO"))  
 #<The A package, 0/16 internal, 1/16 external>
```

```
CL-USER 2 > (defpackage "B"  
  (:use "A"))  
 #<The B package, 0/16 internal, 0/16 external>
```

```
CL-USER 3 > (eq 'a:hello 'b::hello)  
 T
```

```
CL-USER 4 > (defvar *a* 'a:hello)  
 A*
```

```
CL-USER 5 > *a*
A:HELLO
```

```
CL-USER 6 > (unintern *a* 'a)
T
```

```
CL-USER 7 > *a*
#:HELLO
```

The #: prefix means that we are printing an uninterned symbol. This syntax is recognised by the reader which will also create uninterned symbols.

```
CL-USER 8 > (eq ' #:foo ' #:foo)
NIL
```

```
CL-USER 9 > (eq 'foo 'foo)
T
```

As a final example, let's have a look at a simple file that we might compile.

```
(in-package :cl-user)
```

```
(defun doPrinting()
  (format t "Hello World"))
```

The key point is that when the compiler processes the `in-package` it has the effect of setting the `*package*` variable to the result of `(find-package "CL-USER")` at compile-time. This then causes the reader to intern the string "DOPRINTING" in the CL-USER package when it is read, so that the result of the defun expands into code much like:

```
(setf (symbol-function 'CL-USER::DOPRINTING
  #'(lambda () (format t "Hello World"))))
```

The mechanism of symbols opens a number of possibilities including the ability to patch the image by redefining symbol-functions at later stages. More on that some other time.

1. <mailto:?DoPrinting@@YAXXZ>

---

### 2.1.3 Meta Maths: The Quest For Omega by Gregory Chaitin (2008-01-07 11:07)

[1]Meta Maths: The Quest For Omega explains some of the relationships between the halting problem, Godel's incompleteness theorem and the measurement of complexity where this is defined as the size of the smallest program that encodes the information.

The book covers how Chaitin became interested in this area (as a means for understanding Gödel's incompleteness theorem) and covers some of the history of such ideas, relating them to philosophy and logic.

This book is intended as a general overview and missed out some of the more technical details which I think I would have found interesting, but was a good read.

[2]Chaitin's website contains some of the related material, such as Lisp programs for calculating some of the digits of omega (the halting probability).

1. [http://www.amazon.co.uk/gp/product/product-description/184354525X/sr=1-1/qid=1199696514/ref=dp\\_proddesc\\_0?ie=UTF8&n=266239&s=books&qid=1199696514&sr=1-1](http://www.amazon.co.uk/gp/product/product-description/184354525X/sr=1-1/qid=1199696514/ref=dp_proddesc_0?ie=UTF8&n=266239&s=books&qid=1199696514&sr=1-1)
  2. <http://www.cs.auckland.ac.nz/~chaitin/>
- 

### **2.1.4 Beautiful code Edited by Andy Oram and Greg Wilson (2008-01-13 14:46)**

[1]Beautiful Code is a spectacularly good series of 33 essays on various problems in computing that were solved, in the opinion of the individual authors, in a beautiful manner. The subject areas of the essays go from bit twiddling implementations that count the number of one bits in a machine word in an efficient manner to verifying XML, from parsing languages to audio enabling emacs. All of the essays were interesting and offered some good insights. Better still, all author royalties for the book are being donated to [2]Amnesty International. To give a flavour of the book, I thought I'd list my 5 favourites.

Beautiful Concurrency by [3]Simon Peyton Jones

This essay was a brief introduction to [4]software transactional memory, a technique of currency control which doesn't require the programmer to deal with locks at the program level. The essay covers an implementation using the [5]Haskell programming language. STM presents the programmer with abstractions allowing him to mark blocks of code as atomic causing reads and writes by the block to be executed as if they were running isolated from other process, and then offers other abstractions for combining such blocks. Atomic blocks can be retried and combined using a choice operator. The retry can be efficiently implemented as the system knows what shared data was read by the transaction and can decide to not run the transaction again until this memory is touched. Such blocks can be implemented very nicely in Haskell using monads to make the nature of the atomic block visible via the type system, and to prevent side effects being mixed into atomic operations that may need to be retried.

Simon will be doing a talk for the Cambridge [6]BCS SPA group in a couple of months, and wrote [7]The implementation of functional programming languages which had a great effect on me when I read it in the mid-80s.

The quest for accelerated population count by Henry S Warren Jr.

This essay concerns the problem of counting the number of binary 1 bits in a given word. It covers a number of very clever algorithms for doing the calculation.

Python's Dictionary Implementation: Being all things to all people by Andrew Kuchling

The notion of a dictionary is central to the implementation of Python. This essay covered the design of the implementation used in CPython, outlining the history and tradeoffs involved in the implementation.

## Distributed Programming With MAPREDUCE by Jeffrey Dean and Sanjay Ghemawat

One technique used by Google for writing programs so that they can be run in parallel across massive numbers of commodity machines is to use the MAPREDUCE model. The processing is broken into two phases. The first phase, called MAP, processes a subset of the data into a set of key/value pairs. The second phase, called REDUCE, takes output for the MAP phase and converts into the final result. By splitting the input data, large number of MAP processs can work in parallel, and lots of REDUCE processes can take these results and combine them to produce a final answer. Many problems can be coerced into this framework.

## A Spoonful of sewage by Bryan Cantrill

An essay on how one bug in the turnstile implementation inside Solaris was tracked down and fixed, which points out some of the interesting tradeoffs that can happen inside the operating system kernel design.

1. [http://www.amazon.co.uk/Beautiful-Code-Theory-Practice-OReilly/dp/0596510047/ref=w1\\_it\\_dp?ie=UTF8&coliid=I273PKU0VM5ZZG&colid=2123C82LZYSII](http://www.amazon.co.uk/Beautiful-Code-Theory-Practice-OReilly/dp/0596510047/ref=w1_it_dp?ie=UTF8&coliid=I273PKU0VM5ZZG&colid=2123C82LZYSII)
2. <http://www.amnesty.org/>
3. <http://research.microsoft.com/~simonpj/>
4. <http://research.microsoft.com/~simonpj/papers/stm/>
5. <http://www.haskell.org/>
6. <http://www.bcs-spa.org/cgi-bin/view/SPA/SPACambridgeEveningMeetingProgramme>
7. <http://research.microsoft.com/~simonpj/Papers/slpj-book-1987/index.htm>

## **2.1.5 The Haskell School Of Expression by Paul Hudak (2008-01-14 12:53)**

[1]The Haskell School of Expression is an introduction to the Haskell programming language.

This was by far the best programming language tutorial that I have ever read.

Using an application for drawing and calculating data about shapes, the book shows us the power of functional programming building through a number of chapters into an application that handles reactive animation. On the way we learn about the power of the abstractions that functional programming languages offer, as well as features such as [2]Monads, streams and [3]type classes. The reactive animation example shows beautifully how a [4]domain specific language can be embedded into Haskell.

For example, you can run a simple paddleball game by loading Reactivate.lhs into Hugs and typing:

```
> test (paddleball 1.0)
```

The ball has the following definition high level definition which is relativly easy to read:

```
> pball vel =  
> let xlabel = vel `stepAccum` xbounce ->> negate  
> xpos = integral xlabel  
> xbounce = when (xpos >* 2 ||* xpos <* -2)  
> ylabel = vel `stepAccum` ybounce ->> negate  
> ypos = integral ylabel  
> ybounce = when (ypos >* 1.5
```

```
> ||* ypos 'between' (-2.0,-1.5) & &*
> fst mouse 'between' (xpos-0.25,xpos+0.25)
> in paint yellow (translate (xpos, ypos) (ell 0.2 0.2))
```

Additional examples of generating Midi files and simulating a robot on the screen demonstrate how easy it is to build on top of the previously implemented modules. Many questions at the end of most chapters help illuminate the text. Fantastic!

The programming environment used, HUGS, can be downloaded from [5]this page which also has an installer for installing all of the [6]examples from the book.

1. [http://www.amazon.co.uk/Haskell-School-Expression-Functional-Programming/dp/0521644089/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1200313168&sr=1-1](http://www.amazon.co.uk/Haskell-School-Expression-Functional-Programming/dp/0521644089/ref=sr_1_1?ie=UTF8&s=books&qid=1200313168&sr=1-1)
  2. <http://citeseer.ist.psu.edu/wadler95monads.html>
  3. [http://en.wikipedia.org/wiki/Type\\_class](http://en.wikipedia.org/wiki/Type_class)
  4. [http://en.wikipedia.org/wiki/Domain-specific\\_programming\\_language](http://en.wikipedia.org/wiki/Domain-specific_programming_language)
  5. <http://cvs.haskell.org/Hugs/pages/downloading-Nov2002.htm>
  6. <http://cvs.haskell.org/Hugs/pages/downloading-Nov2002.htm>
- 

## 2.1.6 I'm not complex, I'm linear (2008-01-15 09:50)

My son got some homework from school connected to common multiples of two integers. For a while I couldn't remember what this meant - until I remembered the least common multiple. This made me remember the rather clever proof using lcm to deduce that the partial sums of the harmonic series

$$1 + 1/2 + 1/3 + \dots + 1/n$$

aren't integer after the first term.

On the way to work I started thinking through some linear algorithms that surprised me when I first heard about them.

[1]The Majority Vote algorithm discovered and proved by Boyer and Moore in 1980. Given a sequence of votes, if we know that there is a candidate with a majority, we can find it in a single linear scan. They proved the algorithm using the [2]Boyer-Moore theorem prover, a theorem prover that was designed to prove things inductively across a computational logic based on Lisp.

Finding loops in a path in linear time using [3]tortoise and hare algorithm, that can be used to solve [4]problems like this.

[5]Finding the median of a collection of numbers in worst case linear time.

[6]Searching for a substring match in super-linear time.

1. <http://www.cs.utexas.edu/users/moore/best-ideas/mjrty/index.html>
2. <http://www.cs.utexas.edu/users/moore/best-ideas/nqthm/index.html>
3. [http://en.wikipedia.org/wiki/Cycle\\_detection](http://en.wikipedia.org/wiki/Cycle_detection)
4. [http://domino.research.ibm.com/Comm/wwwr\\_ponder.nsf/Challenges/January2004.html](http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/Challenges/January2004.html)
5. [http://en.wikipedia.org/wiki/Selection\\_algorithm](http://en.wikipedia.org/wiki/Selection_algorithm)
6. <http://www.cs.utexas.edu/~moore/best-ideas/string-searching/>

---

## **2.1.7 Please, let me continue! (2008-01-16 11:21)**

I got interested again in continuations when we got a stack overflow in an algorithm at a customer site. The algorithm used recursion to neatly express what it was doing, but it was being applied to a dataset that caused too much recursion causing a stack overflow. For such recursive algorithms, it's hard to predict when this might be a problem. [1]Compiling into continuations gets around this rather neatly by translating stack based recursion into heap based allocation.

I've recently been interested in the [2]actor model of object communication and this is tricky to implement well on top of stacks, but continuations solve this too.

Lastly, continuations are also interesting to me as they are related to the [3]Curry-Howard isomorphism. The derivation of functional programmes from proofs is something that interests me. For example see the excellent [4]Adventures in Classical-Land.

The concept of a continuation is fairly straightforward. It is a function which we call with the answer to our part of the computation. Normally, we'd simply return the result, but in the CPS style we call someone else with the result.

For example, the following Lisp code,

```
(fact (+ x 1))
```

would be translated into

```
(funcall  
(lambda (argument)  
(fact' argument k))  
(+ x 1))
```

where k is the continuation that is going to take our result.

Basically, all non-primitive functions are translated into functions that take an argument and a continuation, and all applications are translated into a series calls that evaluate the argument and then pass the result into a continuation that applies the function passing the result on to the continuation that we defer to.

I wanted to demonstrate the recursion of the factorial function, so a simple CPS translater that handles only single argument functions is:

```
(defun process-code (code)  
(if (atom code)  
'(lambda (k)  
(funcall k ,code))  
(let ((operation (first code)))  
(ecase operation  
(fact
```

```

'(lambda (k)
(funcall ,(process-code (second code)))
(lambda (x)
(fact x k))))))
((* - =)
(let ((x (second code))
(y (third code)))
'(lambda (k)
(funcall ,(process-code x)
(lambda (x-val)
(funcall ,(process-code y)
(lambda (y-val)
(funcall k ,(operation x-val y-val))))))))
((if)
(let ((condition (second code))
(then (third code))
(else (fourth code)))
'(lambda (k)
(funcall ,(process-code condition)
(lambda (condition)
(if condition
(funcall ,(process-code then) k)
(funcall ,(process-code else) k)))))))

```

```

(defmacro defun-as-cps (func (argument) body)
(with-unique-names (continuation)
'(defun ,func ,(argument ,continuation)
(funcall ,(process-code body)
,continuation))))

```

```

(defun-as-cps fact (x)
(if (= x 0)
1
(* x (fact (- x 1)))))


```

We can call the CPS version of factorial using:

```
CL-USER 111 > (fact 10000 #'(lambda (res) res))
```

Calling the standard recursive version gives a stack overflow well before that point.

There is another spin off from the translation. Each use of funcall in the above code is essentially the application of a single unit of work in the computation. By replacing funcall with a function we control, call-next-step, we can constraint how and when the steps of the current computation get executed.

If we define a simple work queue:

```
(defvar *work-queue* nil)
```

```
(defun execute ()  
(when *work-queue*  
(apply 'funcall (pop *work-queue*))))
```

```
(defun call-next-step (f k)  
(push-end (list f k) *work-queue*))
```

And replace funcall in the translator with call-next-step,

```
(defun process-code2 (code)
(if (atom code)
'(lambda (k)
(call-next-step k ,code))
(let ((operation (first code)))
(ecase operation
(fact2
'(lambda (k)
(call-next-step ,(process-code2 (second code)))
(lambda (x)
(fact2 x k))))))
((* - =)
(let ((x (second code))
(y (third code)))
'(lambda (k)
(call-next-step ,(process-code2 x)
(lambda (x-val)
(call-next-step ,(process-code2 y)
(lambda (y-val)
(call-next-step k ,(operation x-val y-val))))))))
(if)
(let ((condition (second code))
(then (third code))
(else (fourth code)))
'(lambda (k)
(call-next-step ,(process-code2 condition)
(lambda (condition)
(if condition
(call-next-step ,(process-code2 then) k)
(call-next-step ,(process-code2 else) k))))))))
```

```
(defmacro defun-as-cps2 (func (argument) body)
(with-unique-names (continuation)
'(defun ,func (,argument ,continuation)
(call-next-step ,(process-code2 body)
,continuation))))
```

```
(defun-as-cps2 fact2 (x)
(if (= x 0)
```

```
1
(* x (fact2 (- x 1))))
```

We can now have computations that run steps in parallel.

```
CL-USER 1 > (fact2 3 #'(lambda (res) (format t "Finished with result ~A ~%" res)))
```

```
CL-USER 2 > (fact2 20 #'(lambda (res) (format t "Finished with result ~A ~%" res)))
```

```
CL-USER 3 > (fact2 4 #'(lambda (res) (format t "Finished with result ~A ~%" res)))
```

```
CL-USER 4 > (execute)
```

```
CL-USER 5 > (execute)
```

```
CL-USER 6 > (execute)
```

```
CL-USER 7 > (dotimes (i 1000) (execute))
Finsihed with result 6
```

Finsihed with result 24

Finsihed with result 2432902008176640000

NIL

The other cool part of this translation is that the various steps of the computation are represented by closures on the heap which can be re-executed. This has been used as a means of handling the interaction pattern of web servers. For example, see [5]<http://bc.tech.coop/blog/040404.html>

1. [http://en.wikipedia.org/wiki/Continuation-passing\\_style](http://en.wikipedia.org/wiki/Continuation-passing_style)
  2. <http://lampwww.epfl.ch/~odersky/papers/jmlc06.pdf>
  3. <http://en.wikipedia.org/wiki/Curry-Howard>
  4. <http://www.haskell.org/sitewiki/images/1/14/TMR-Issue6.pdf>
  5. <http://bc.tech.coop/blog/040404.html>
- 

## 2.1.8 Yes please, I'd like a game (2008-01-21 13:53)

Ever since we bought a Nintendo DS, my son has been keen on writing a game for it. As a means of getting into writing such games, I set myself a target of writing something simple which would:

1. Demonstrate some sprites and how to move them around
2. Demonstrate tile backgrounds
3. Display text in the lower DS pane
4. Enable state to be saved

I've now achieved that goal. It's taken a while as I found it hard to find good tutorials on writing DS code on the internet. There are many partial tutorials and lots of full blown games, but very little in between. The attached example consists of bits of code taken from the devKitPro examples and elsewhere. We display three rotating squares with the middle square

(in the initial position and z-order) under the control of the direction keys. A counter value is displayed in the lower window. Pressing "A" saves the counter to the (R4) memory card and pressing "B" restores the saved value.

The code works on top of the devKitPro development environment that can be downloaded [1]here. To build it, take the code from [2]here, unzip it and in the msys window type make. This will produce the files CTPlay2.nds (the game) and ctplay.txt (the file storing the state) which need to be transferred onto an R4 card's micro SD card which is the only thing I have tried it on for real. During development I tested on the DeSmuMe simulator, though the saving doesn't work on the version of the simulator that I have, though it does work on the full DS.

Total cost to play with this kind of thing is just 25 UK pounds for the R4 chip and some micro SD cards. Now I just need to improve my C programming skills and get a clone version of PacMan written.

1. [http://sourceforge.net/project/showfiles.php?group\\_id=114505&package\\_id=160396](http://sourceforge.net/project/showfiles.php?group_id=114505&package_id=160396)
  2. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/NDSexamples/CTPlay2.zip>
- 

### **2.1.9 How did he do that? (2008-01-22 14:49)**

I just finished the 800 pages that is [1]Joe Celko's SQL for Smarties: Advanced SQL Programming

This book is basically a massive set of examples of SQL solutions to a variety of problems. The text is divided into sections covering various areas like how NULLs can be handled and the use of temporary tables, with some repetition between the chapters. Some of the solutions are ingenious. This was a good introduction to some of the more advanced uses of SQL.

An excellent book.

1. [http://www.amazon.co.uk/gp/product/product-description/0123693799/sr=1-2/qid=1201012845/ref=dp\\_proddesc\\_0?ie=UTF8&n=266239&s=books&qid=1201012845&sr=1-2](http://www.amazon.co.uk/gp/product/product-description/0123693799/sr=1-2/qid=1201012845/ref=dp_proddesc_0?ie=UTF8&n=266239&s=books&qid=1201012845&sr=1-2)

---

### 2.1.10 Not just a normal variable. I'm special! (2008-01-23 11:04)

In my list of Common Lisp notes, I wanted to get on to handler-case and handler-bind, but I think before I do that I ought to try to explain a little about special variables.

Normal variables in Lisp are lexically scoped. When you bind them in a let expression or as arguments to a function, the variable can only be accessed within the lexical scope.

```
(progn
  (let ((x 20))
    ... I can reference x anywhere here ...
  )
  ... But I can't access it here ...
)
```

The extent of the variable however, may be larger than when the scope is active, so in the following example, the location x is shared by the two closures that can access it for reading and writing.

```
CL-USER 1 > (let ((x 10))
  (cons #'(lambda () x)
```

```
#'(lambda (y) (setq x y)))  
( #<anonymous interpreted function 217DC1C2> . #<anonymous interpreted function  
217DB012>)
```

```
CL-USER 2 > (funcall (car *))  
10
```

```
CL-USER 3 > (funcall (cdr **) 20)  
20
```

```
CL-USER 4 > (funcall (car ***))  
20
```

Special variables allow the same location to be accessed outside the lexical scope of any binding.

```
(defun outer ()  
(let ((x 10))  
(declare (special x))  
(inner)))
```

```
(defun inner ()  
(declare (special x))  
(print x))
```

```
CL-USER 7 > (outer)
```

10  
10

Note that special bindings nest, so that in the following the first call of (inner) accesses the nested binding of x, which is restored for the second call to (inner).

```
(defun outer2 ()  
(let ((x 10))  
(declare (special x))  
(let ((x 20))  
(declare (special x))  
(inner))  
(inner)))
```

CL-USER 10 > (outer2)

20  
10  
10

The same variable can be used as special and non-special in the same form, so that in the following example, (inner) references the special bound x which is bound by the first let but not the second.

```
(defun outer2 ()  
(let ((x 10))  
(declare (special x))
```

```
(let ((x 20))
  (inner)
  (inner)))
```

```
CL-USER 11 > (outer2)
```

```
10
10
10
```

To avoid the need to keep adding the special declarations, a variable can be declared special globally by

```
CL-USER 12 > (proclaim '(special y))
NIL
```

or more usually by using defvar or defparameter to define a variable which has the side-effect of making it special.

By convention, special variables are usually named with leading and trailing \*, ie \*x\* would have been a more conventional name to use in the above examples.

In a multi-threaded Common Lisp, a binding of a special variable is thread local, making them very much like thread local variables that you find in other languages. There is one big difference though. In other languages, a thread local is implemented as something like:

```
try
{
    ... store old value of the thread local variable location ...
    ... put new value into the location ...
    ... do something ...
}
finally
{
    ... restore the old value into the thread location ...
}
```

This kind of code doesn't behave well in the presence of asynchronous exceptions, like the ThreadAbort exception of C# which could happen when the finally is entered but before the restore happens, leading to the unwinding of the binding not happening.

With the Common Lisp equivalent,

```
(let ((*x* 10))
  ... do something ...)
)
```

the Lisp system guarantees that the unwinding happens.

How are specials normally implemented? The implementations I have worked on used a technique called shallow binding. The symbol-value slot of the symbol is used to hold the current value. When a special bindings is made, the old value is pushed on to the stack and chained into a linked list of current bindings. When a non-local return (such as a throw) happens, the bindings list can be used by the runtime system to unwind special bindings whose stack frames are no longer active. This technique is good in that accessing the current value just means accessing the value slot of the symbol so it is fast. However, this simple technique only works on uniprocessor implementations and requires that whenever the Lisp thread changes, the bindings for the current thread need to be undone and the bindings for the new thread need to be switched in.

The way that special binding interacts with the contours of the stack gives special variables many uses. Hopefully, I'll get on to them soon.

---

## 2.2 February

### 2.2.1 No Mr Postman. Visit the houses in this order! (2008-02-10 16:47)

[1]Nets, Puzzles and Postmen: An explanation of Mathematical connections by Peter M Higgins

I was browsing in Borders bookshop waiting for my wife when I came across this book. I got hooked after the first chapter, bought it and have been reading it over the past couple of weeks. This book is a great introduction to graph theory and its applications, introducing theorems like the handshaking lemma and algorithms like the spanning tree algorithms of Djikstra and Prim and the Ford-Fulkerson min cut max flow theorem. It then goes on to show how they can be used to solve a number of practical problems. Proofs of the harder theorems are placed in a final chapter that can be read by the interested reader. It's an easy read, but very interesting.

A few other interesting books of this type that I have read recently are: [2]Knots: Mathematics with a twist, [3]Four colors suffice: How the map problem was solved and [4]Symmetry and the monster: One of the greatest quests of mathematics.

1. [http://www.amazon.co.uk/Nets-Puzzles-Postmen-Exploration-Mathematical/dp/0199218420/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1202661075&sr=1-1](http://www.amazon.co.uk/Nets-Puzzles-Postmen-Exploration-Mathematical/dp/0199218420/ref=sr_1_1?ie=UTF8&s=books&qid=1202661075&sr=1-1)
  2. [http://www.amazon.co.uk/Knots-Mathematics-Twist-B-Sossinsky/dp/0674009444/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1202661847&sr=1-1](http://www.amazon.co.uk/Knots-Mathematics-Twist-B-Sossinsky/dp/0674009444/ref=sr_1_1?ie=UTF8&s=books&qid=1202661847&sr=1-1)
  3. [http://www.amazon.co.uk/Four-Colors-Suffice-Problem-Solved/dp/0691115338/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1202661944&sr=1-1](http://www.amazon.co.uk/Four-Colors-Suffice-Problem-Solved/dp/0691115338/ref=sr_1_1?ie=UTF8&s=books&qid=1202661944&sr=1-1)
  4. [http://www.amazon.co.uk/Symmetry-Monster-Greatest-Quests-Mathematics/dp/0192807234/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1202661992&sr=1-1](http://www.amazon.co.uk/Symmetry-Monster-Greatest-Quests-Mathematics/dp/0192807234/ref=sr_1_1?ie=UTF8&s=books&qid=1202661992&sr=1-1)
- 

### **2.2.2 Pacman... march on! (2008-02-10 16:53)**

[1]The next version of a PacMan type game is available here. I realise that it isn't that good, but at least it shows some of the things you can do with the Nintendo DS.

1. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/NDSexamples/main-1.zip>
- 

### **2.2.3 Put that down... it's sharp (2008-02-14 18:41)**

I've just finished reading [1]Expert F # by Don Syme and two other authors. This was a fantastic book, though it's a little odd in that it appears to have two purposes.

First it introduces the syntax of F # and highlights the advantages of the functional style of programming, and second it shows how many useful applications can be built using F # with an emphasis on integrating well with the object oriented .NET framework. There are examples of applications that use Windows Forms to display algebraic expressions and applications that get data from databases. To me, the applications were interesting, but I'd have liked a lot more depth on the features of the language - workflows, active patterns and how F # modules appear to other .NET languages were not covered in as much detail as I would have liked.

I've done a lot of programming in ML over the years, so the functional syntax of F # seemed familiar, though once the object oriented features of the language were introduced, the syntax of the type definitions started to look a little complicated to me. I also haven't made up my mind whether I like the #light syntax, where whitespace is used to reflect the scope of the various constructs.

In summary though, a very good book, and a language with a great deal of potential.

1. [http://www.amazon.co.uk/Expert-F-Experts-Voice-Net/dp/1590598504/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1203013756&sr=1-1](http://www.amazon.co.uk/Expert-F-Experts-Voice-Net/dp/1590598504/ref=sr_1_1?ie=UTF8&s=books&qid=1203013756&sr=1-1)
- 

#### **2.2.4 That's an interesting bit of information (2008-02-20 12:19)**

[1]Programming the universe: A quantum computer scientist takes on the cosmos by Seth Lloyd

I thought this was an interesting book. It talks about information content and its relationship to the other laws of physics using information to explain thought experiments like Maxwell's demon. The book argues that the universe can be considered as a quantum computer. I enjoyed the book, especially the discussion of quantum computers and what they can achieve, though the book lacked depth on such issues. However, Scott Aaronson has a good article in the latest [2][www.sciam.com](http://www.sciam.com) Scientific American that discusses the types of algorithms that

quantum computers will be able to perform, dispelling the notions that all NP problems will fall at their feet. On his blog, he also has a simple explanation of the way that [3]Shor's factorization algorithm can be implemented on such a computer.

1. [http://www.amazon.co.uk/Programming-Universe-Quantum-Computer-Scientist/dp/0099455374/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1203494104&sr=1-1](http://www.amazon.co.uk/Programming-Universe-Quantum-Computer-Scientist/dp/0099455374/ref=sr_1_1?ie=UTF8&s=books&qid=1203494104&sr=1-1)
  2. <http://www.sciam.com/>
  3. <http://www.scottaaronson.com/blog/?p=208>
- 

## 2.3 March

### 2.3.1 It won't twist that way! (2008-03-04 08:11)

[1]Poincare's Prize: The Hundred-year Quest to Solve One of Math's Greatest Puzzles by George Szpiro

This was a great read, covering the people involved over the last hundred years in the quest for a proof of the Poincare conjecture. The maths was covered in more detail than some of the other books on this conjecture, giving enough overview to make the reader aware of that person's contribution and how their contribution fell short. Topology has always been an interest of mine, and I loved this book and will read it again soon.

The same author wrote [2]Kepler's conjecture: How some of the greatest minds in history helped solve one of the oldest math problems in the world which I also think is a very good book.

1. [http://www.amazon.co.uk/Poincares-Prize-Hundred-year-Greatest-Puzzles/dp/0525950249/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1204617762&sr=1-1](http://www.amazon.co.uk/Poincares-Prize-Hundred-year-Greatest-Puzzles/dp/0525950249/ref=sr_1_1?ie=UTF8&s=books&qid=1204617762&sr=1-1)
  2. [http://www.amazon.co.uk/Keplers-Conjecture-Greatest-History-Problems/dp/0471086010/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1204617973&sr=1-1](http://www.amazon.co.uk/Keplers-Conjecture-Greatest-History-Problems/dp/0471086010/ref=sr_1_1?ie=UTF8&s=books&qid=1204617973&sr=1-1)
- 

### 2.3.2 It's a bit like lego (2008-03-09 14:57)

I've just finished reading [1]Component Software: Beyond Object-Oriented Programming by Clemens Szyperski.

I read this book for the first time many years ago after a recommendation from my then-manager. It is a great discussion of component technology, going from the low level details such as the patterns of events delivery that can happen when multiple components are involved (such as nested and recursive) and the subsequent problems of re-entry into the object model, to higher level details such as a discussion of the viability of component marketplaces. The technologies that it covers, CORBA, COM and Java Beans are perhaps a little dated now, but there is enough information in the book to keep it relevant and interesting. Having done a lot of work on CORBA and COM in the past, it was interesting to re-read details about the similarities and differences between them.

The book was recently mentioned in an episode of [2]Software Engineering Radio on component technology, which covered material necessary for the following episode on [3]Singularity, the Microsoft Research operating system which implements the related notion of software isolated processes. essentially components with manifests that detail the requirements which communicate via channels. This operating system is rather interesting in that it is based on managed code, together with communication channels that can be statically verified. This gives the operating system the ability to avoid the need for hardware enforced virtual memory protection, an overhead that they have measured at 30 % for some scenarios.

1. [http://www.amazon.co.uk/Component-Software-Beyond-Object-Oriented-Programming/dp/0201745720/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1205073740&sr=1-1](http://www.amazon.co.uk/Component-Software-Beyond-Object-Oriented-Programming/dp/0201745720/ref=sr_1_1?ie=UTF8&s=books&qid=1205073740&sr=1-1)
  2. <http://www.se-radio.net/podcast/2008-02/episode-87-software-components>
  3. <http://www.se-radio.net/podcast/2008-03/episode-88-singularity-research-os-galen-hunt>
- 

### **2.3.3 It's a generational thing (2008-03-09 15:13)**

Mark Dalgarno has recently published details about this year's Code Generation conference, [1]CG2008, which will be held here in Cambridge. I attended last year's [2]conference and it was fantastic. Mark recorded a couple of audios of two of the sessions from last year which can be accessed from the CG2007 page to give a flavour of the conference. It was a good introduction to DSL and the uses people had for UML, though most the speakers seemed rather against UML, preferring DSLs instead. Microsoft were even there showing some of their DSL tools which are now part of Visual Studio. This year's conference looks even more interesting with some good speakers.

It will be interesting to see what Microsoft are planning in this area. At the recent Lang.NET symposium, Don Box gave a quick talk about raising the abstraction level without going into [3]details about what his team are really doing.

1. <http://www.codegeneration.net/conference/index.php>
  2. <http://www.codegeneration.net/cg2007/index.php>
  3. <http://langnetsymposium.com/talks.asp>
- 

Mark Dalgarno (2008-03-11 17:33:19)

Thanks for the nice words about this year's and last year's events. We had a much higher number of people wanting to speak at CG2008 as opposed to CG2007 and this is one indicator that interest in the event (and in code generation tools and technologies) is growing. Traffic to the conference site is good and the bookings have started coming in. You forgot to say that booking is now open and that an early-bird booking period (with savings up to 20 %) runs until 31st March 2008 :-) Book now before places run out! Visit the Code Generation 2008 web site for more information...

### **2.3.4 That argument is not complete! (2008-03-25 13:10)**

I've just finished reading [1]Gödel's Theorem: An Incomplete Guide to Its Use and Abuse by Torkel Franzén.

This was a really good read and at less than two hundred pages it isn't too long. However, some of the points that the author makes take some work on the part of the reader to fully comprehend. This book draws out many common misconceptions about Godel's theorem and its supposed applications, placing the theorem within a wider philosophical setting. It covers a number of interesting issues and confusions.

- How first-order logic is complete while still being a target of the incompleteness theorem.
- How certain theorems being undecidable would mean that they were true.
- How Godel's theorem is only about the arithmetic portion of a theory
- How Penrose's arguments about human consciousness use interpretations of the incompleteness results that aren't really valid.

A fantastic book!

1. [http://www.amazon.co.uk/Godels-Theorem-Incomplete-Guide-Abuse/dp/1568812388/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1206348178&sr=1-1](http://www.amazon.co.uk/Godels-Theorem-Incomplete-Guide-Abuse/dp/1568812388/ref=sr_1_1?ie=UTF8&s=books&qid=1206348178&sr=1-1)

---

### **2.3.5 You're nothing like the prototype (2008-03-27 08:02)**

I've just been doing a little more reading about javascript. The language has always impressed me with its simplicity, the rather neat way that everything is explained using properties on a object together with a prototype pointer that points to another object on which the properties could be looked up. This so-called "differential inheritance" was investigated by the [1]SELF language in the late 80's.

There is already an implementation of javascript for the Microsoft DLR (dynamic language runtime) which is embedded into the [2]DLR console standard Silverlight sample Using this it is easy to play with javascript example at a read-eval-print-loop without having to write a web page.

This is an example of the kind of inheritance that you can use in the language. We define a constructor function for a "thing". A thing has a name and a value which is set to 20.

```
function thing(name)
{
    this.name = name;
    this.a = 20;
}
```

We can make two new thing objects.

```
var xx = new thing("thing1");
var yy = new thing("thing2");
```

We can look at the properties of the thing bound to xx.

```
xx; -> object representation
xx.a; -> 20.0
xx.name; -> 'thing1'
xx.z -> undefined
```

When a constructor function is used, the resulting object has a prototype which is the same object as the prototype of the function object ie `prototype(xx)` is the same object as `prototype(thing)`. If we add a property to this prototype:

```
thing.prototype.z = 30
```

then since `xx` and `yy` have this as their prototype, if a lookup fails on `xx` or `yy`, the lookup continues on the prototype. Hence `xx` and `yy` now have a slot named `z`.

```
xx.z -> 30.0  
yy.z -> 30.0
```

Instances of new objects also have a prototype. This prototype will be the prototype of the prototype of function objects like the definition of `thing` above. (In javascript, a function is an object too)

```
var zz = new Object();  
zz.x -> Undefined
```

If we put a value on the prototype of `Object`, then it is shared by all of the objects we have created so far.

```
Object.prototype.pp = 60;
```

```
xx.p -> 60.0  
zz.p -> 60.0
```

Nice!

1. <http://research.sun.com/self/papers/papers.html>
  2. <http://dynamicsilverlight.net/see/dlrconsole/>
- 

### **2.3.6 Testing ... testing...1...2...3 (2008-03-27 08:15)**

A couple of weeks ago I attended the latest [1]BCS SPA cambridge talk which was by [2]Simon Peyton Jones on type driven testing in Haskell. This was a great talk and [3]it was recorded.

Most of the audience hadn't programmed in a functional languages before, so the first part of the talk was a brief introduction to the advantages of functional programming languages. Consistency of expression, clarity and understandability were the buzzwords here. The lack of side-effects make code much more understandable, and the availability of higher level datatypes often makes the algorithm easier to follow. However, the higher level of abstraction takes the

language further away from the machine and it can be difficult to get the compiler to generate efficient code. In the 80's people looked at using the lack of side effects to generate programs that could be executed in parallel on graph reduction machines; this effort generated a lot of important research results, but didn't succeed in the market place as commonly available machines obeyed Moore's laws and doubled in speed over regular time intervals, and the languages of the marketplace weren't really suitable for parallelising. It was interesting to hear Peyton-Jones state that investments in compiler technology are now making such languages competitive with more mainstream languages in many areas. Tricks like memoization also help too.

The next part of the talk was a brief introduction to Haskell. This is a very impressive language which is for the most part pure. Monads are used to isolate side effecting parts of the program into small islands of functions with types which make the presence of side effects obvious.

This part introduced type classes, the means for demanding that the types that a function can be applied to offer other support functions. In languages like ML, functions like + and \* are special in that there are two versions of the function, one for reals and one for ints, but any use of the these functions inside another must resolve to either the real or the int case. In Haskell, we can instead have a type class that reflects the presence of + and \* functions and then constraint the types of the resulting signature to be instances of the given type class.

In the following example, we define two type classes, Class1 and Class2 where Class2 extends class Class1.

```
module Play where
```

```
class Class1 a where
  foo :: a -> [a]
```

```
class (Class1 a) => Class2 a where  
bar :: a -> [a]
```

```
instance Class1 Int where  
foo x = [x]
```

```
instance Class2 Int where  
bar x = foo x
```

```
play x = bar x
```

```
play2 x = foo x
```

If we look at the types of play and play2, we can see that the function is fully polymorphic with the proviso that instance types support the Class1 or Class2 ie that the type implements foo and bar methods.

```
Play> :type play  
play :: Class2 a => a -> [a]
```

```
Play> :type play2
play2 :: Class1 a => a -> [a]
```

Because of the instance declarations, we have given suitable definitions for Int of foo and bar and can therefore call play on an integer.

```
Play> play (2 :: Int)
[2]
```

The type classes were used by the testing framework for generating the test cases to pass through the test functions. All very tidy indeed.

Peyton-Jones argued that functional languages were easier to test because the lack of side effects means that you don't have the problems of setting up the world before calling the function and then checking the state of the world afterwards. This is true for the pure parts of the code, though the same could be said of the pure parts of C# code that are being tested. Also, setting up the state of the world isn't always so hard as you'd typically use mock objects and dependency injection to get these partial models of the world used during the test case.

Functional languages do have a lot to offer. When I was doing a lot of ML programming, the weird thing was that most of the debugging was at the type level getting the program to type

check - once it type checked correctly then it would often run successfully. It will be interesting to see if F # can finally take these languages into the mainstream.

1. <http://www.bcs-spa.org/cgi-bin/view/SPA/TypeDrivenTestingInHaskell>
  2. <http://research.microsoft.com/~simonpj/>
  3. <http://www.foomongers.org.uk/videos/spj-typedriventestinginhaskell.html>
- 

## 2.4 April

### 2.4.1 Keep this a secret! (2008-04-07 21:24)

I've just finished reading [1]Cryptography Decrypted by H.X.Mel and Doris Baker. I've read a few texts on the the subject before, but this was certainly the most accessible book on the subject that I have ever read. It covered some of the history of code making and code breaking, and then moved on to symmetric and asymmetric cryptography followed by an introduction to IPSec and SSL. It explained the basic concepts well with many examples, pushing some of the details in the appendixes of the book.

One of the appendixes covered the math behind public key cryptography, and since I'd never tried implementing any of this stuff before, I thought I should see how hard it was to implement (in a very inefficient manner). My current favourite language for rapidly prototyping code on .NET is F#. The type checking together with the REPL (read-eval-print loop) make it easy to write code while exploring the problem space. In the end I spent about one hour getting some of the basic algorithms together.

This is a copy of the F # session:

First we define a function for checking if a number is prime, by testing for divisibility by all numbers up to the square root of the number under test. We use the bigint type, a type containing unbounded integer values. This is separate from the integer type and requires us to put I after the numeric forms to avoid type errors.

```
> #light
- let isPrime (n : bigint) =
- let lim = Math.BigInt.FromInt64(int64(sqrt(float(n)))))
- let rec sieve m =
- m > lim or (n % m <> 0I & & sieve (m+1I))
- sieve 2I;;
val isPrime : bigint -> bool
```

We can run a few quick tests on the function.

```
> isPrime 2I;;
val it : bool = true
> isPrime 3I;;
val it : bool = true
> isPrime 4I;;
val it : bool = false
> isPrime 5I;;
val it : bool = true
```

We'll need a list of small primes later for filtering, so we can quickly generate them using a sequence expression.

```
> let smallPrimesSequence = seq { 2I .. 100I } |> Seq.filter isPrime;;
```

```
val smallPrimesSequence : seq<bigint>
```

```
> for i in smallPrimesSequence do
```

```
- printfn " %A" i;;
```

```
2I
```

```
3I
```

```
..... lots of lines missed out
```

```
97I
```

```
val it : unit = ()
```

```
> let smallprimes = Seq.to _list smallPrimesSequence ;;
```

```
val smallprimes : bigint list
```

We define a function for raising one number to an integer power, and test it on a sequence that we recognise.

```
> let power n m =
```

```
- let rec accumulatedPower acc left raised =
```

```
- if left = 0I
```

```
- then acc
```

```
- else
```

```
- let acc' = if left % 2I = 1I then acc * raised else acc
```

```
- accumulatedPower acc' (left / 2I) (raised * raised)
```

```
- accumulatedPower 1I m n;;
```

```
val power : Math.BigInt -> bigint -> bigint
```

```
> for i in 1I .. 12I do
```

```
- printfn " %A" (power 2I i);;
```

```
2I
```

```
4I
```

```
8I
```

```
16I
```

```
32I
```

```
74
```

```
64I  
128I  
256I  
512I  
1024I  
2048I  
4096I  
val it : unit = ()
```

A good way to spot non-primes is to see if a number passes the Fermat test. For a prime  $p$ , and a number  $m$  less than  $p$ , it is a theorem of Fermat that  $m^{(p-1)} \bmod p = 1$ . We'll apply the Fermat test with the value  $m$  as 2,3,5,7. If the number passes this and isn't divisible by any of the numbers on our list of primes, we know there is a very good chance of it being prime.

```
> let possiblePrimeViaFermat (p : bigint) m =  
- (power m p) % p = m;;  
  
val possiblePrimeViaFermat : bigint -> Math.BigInt -> bool
```

```
> possiblePrimeViaFermat 127I 3I;;  
val it : bool = true  
> possiblePrimeViaFermat 105I 3I;;  
val it : bool = false  
> let testForPrimality (n : bigint) =  
- let rec filterUsingSmallPrimes candidates =  
- match candidates with  
- | [] -> true  
- | h::t -> (n % h <> 0I && filterUsingSmallPrimes t)  
- let fermatTest = possiblePrimeViaFermat n  
- filterUsingSmallPrimes smallprimes  
- & & fermatTest 2I && fermatTest 3I  
- & & fermatTest 4I && fermatTest 5I;;
```

```
val testForPrimality : bigint -> bool
```

```
> testForPrimality 220I;;
val it : bool = false
```

Given a starting value that we'll suppose has been generated randomly, we need to find the next prime.

```
> let rec nextPrime (n : bigint) =
- if testForPrimality n
- then n
- else nextPrime (n+1I);;
```

```
val nextPrime : bigint -> bigint
```

```
> nextPrime 101I;;
val it : bigint = 101I
```

Given two numbers, a modulus value  $n$  and a potential exponent in the public key algorithm,  $m$ , we need to determine if  $n$  and  $m$  are co-prime and we need to determine the multiplicative inverse of  $m$  in the group for that modulus. A modified version of Euclid's algorithm allows us to calculate both values at the same time.

```
> let rec hcf (n : bigint) m =
- let rec hcf' a b c d =
- if b = 0I
- then (a, if c < 0I then c + n * (1I + (-c / n)) else c)
- else
- let multiple = a / b
- hcf' b (a - b * multiple) d (c - d * multiple)
- if n < m then failwith "invalid parameters" else hcf' n m n 1I;;
```

```
val hcf : bigint -> bigint -> bigint * Math.BigInt
```

```
> hcf 7I 5I ;;
val it : bigint * Math.BigInt = (1I, 3I)
> hcf 8I 4I;;
val it : bigint * Math.BigInt = (4I, 1I)
```

The first return value shows that 7 and 5 are co-prime, and that  $3 * 5 = 1 \text{ mod } 7$ . The second example shows that 8 and 4 are not co-prime.

Given two primes,  $p$  and  $q$ , we need to find  $e$  and  $d$  where  $e * d = 1 \text{ mod } (p-1)(q-1)$  and  $e$  is co-prime to  $(p-1)(q-1)$ . Algorithms usually start out with  $e=3$  and only use another value if it doesn't meet the criteria.

```
> let getExponents p q =
- let decrementedPrimeProduct = (p - 1I)*(q - 1I)
- let rec testExponent e =
- let (h, d) = hcf decrementedPrimeProduct e
- if h = 1I then (e,d) else testExponent (e+1I)
- testExponent 3I;;
```

```
val getExponents : bigint -> Math.BigInt -> bigint * Math.BigInt
```

```
> getExponents 5I 11I;;
val it : bigint * Math.BigInt = (3I, 27I)
```

Lastly, we can define a function that takes the two primes and returns the modulus and the exponents for encrypting and decrypting.

```
> let getPublicPrivateKeys p q =
- let n = p * q
- let (e,d) = getExponents p q
- (n,e,d);;

val getPublicPrivateKeys :
  bigint -> Math.BigInt -> Math.BigInt * bigint * Math.BigInt
```

For the example in the book, they take primes 5 and 11 and get exponent 3 for encrypting and 27 for decrypting.

```
> getPublicPrivateKeys 5I 11I;;
val it : Math.BigInt * bigint * Math.BigInt = (55I, 3I, 27I)
```

We can define a test function for passing a message through the encoding process:

```
> let test keyData message =
- match keyData with
- | (modulus, e, d) ->
- let encoded = (power message e) % modulus
- let decoded = (power encoded d) % modulus
- (message, encoded, decoded);;
```

```

val test :
Math.BigInt * bigint * bigint -> Math.BigInt ->
Math.BigInt * Math.BigInt * Math.BigInt

> let myTest = test (getPublicPrivateKeys (nextPrime 170I) (nextPrime 300I));;

val myTest : (Math.BigInt -> Math.BigInt * Math.BigInt * Math.BigInt)

> myTest 29292I;;
val it : Math.BigInt * Math.BigInt * Math.BigInt = (29292I, 2290I, 29292I)

```

All in all, this was quite a fun experience. The algorithms are easy to write in F # without having to add a lot of fluff to get things to run, and it is easy to execute the functions from the command line to test out the code and explore the problem space.

---

1. [http://www.amazon.co.uk/Cryptography-Decrypted-H-X-Mel/dp/0201616475/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1207601274&sr=1-1](http://www.amazon.co.uk/Cryptography-Decrypted-H-X-Mel/dp/0201616475/ref=sr_1_1?ie=UTF8&s=books&qid=1207601274&sr=1-1)

## 2.4.2 Quickly now! (2008-04-15 07:29)

I've been doing some reading about functional datatypes, after seeing some interesting papers go past on [1]Lambda The Ultimate. One such papers was Chris Okasaki's [2]Purely Functional Random Access Lists which gives a rather clever implementation of a random access list with O(log(n)) update and lookup operations and O(1) list operations (cons, head, tail).

I like these algorithms. Often they are quite clever in their implementation and they get rid of the complex task of needing to lock data structures without sacrificing potential concurrency (or at least move it to a more central location where the multiple versions of the data structure are merged).

F# gives a nice blend of functional programming for the implementation whilst still allowing me to wrap the code into classes for exposing to other consumers.

```
#light
```

```
module FunctionalDatatypes
```

```
type 'a Tree =
Leaf of 'a
| Node of 'a * Tree<'a> * Tree<'a>;
```

```
let rec treeLookup size tree index =
match tree, index with
| Leaf x, 0 -> x
| Leaf x, _ -> failwith "Subscript error"
| Node(x, _, _), 0 -> x
| Node(x,t1,t2), i ->
let size' = i / 2
if i <= size'
```

```
then treeLookup size' t1 (i-1)
else treeLookup size' t2 (i-1-size');;
```

```
let rec treeUpdate size tree index value =
match tree, index with
| Leaf x, 0 -> Leaf value
| Leaf x, _ -> failwith "Out of bounds"
| Node(x,t1,t2), 0 -> Node(value,t1,t2)
| Node(x,t1,t2), _ ->
let size' = size / 2
if index <= size'
then Node(x, treeUpdate size' t1 (index-1) value, t2)
else Node(x, t1, treeUpdate size' t2 (index-1-size') value);;
```

```
let rec lookup array i =
match array with
| [] -> failwith "Subscript"
| ((size,t) :: rest) ->
if i < size
then treeLookup size t i
else lookup rest (i-size);;
```

```
let rec update array i value =
match array with
| [] -> failwith "Subscript"
| (size,t) :: rest ->
if i < size
then (size, treeUpdate size t i value) :: rest
else (size, t) :: update rest (i-size) value;;
```

```
let cons x array =
match array with
| ((size1, t1) ::(size2, t2) ::rest) as xs ->
```

```

if size1 = size2
then (1+size1+size2, Node(x,t1,t2)) ::rest
else (1, Leaf x) :: xs
| xs -> (1, Leaf x)::xs

```

```

let head array =
match array with
| (size, Leaf x) :: rest -> x
| (size, Node(x, _, _)) :: _ -> x
| _ -> failwith "No items in the array"

```

```

let tail array =
match array with
| (_, Leaf x):: rest -> rest
| (size, Node(x,t1,t2)) :: rest ->
let size' = size / 2
(size', t1) :: (size', t2) :: rest
| _ -> failwith "No items in the array"

```

```

type RandomAccessList<'a> (content) =
member x.cons(item : 'a) = RandomAccessList(cons item content)
member x.head = head content
member x.tail() = RandomAccessList(tail content)
member x.lookup(index) = lookup content index
member x.update(index, value) = RandomAccessList(update content index value)
member x.item with get(i) = lookup content i;;

```

```
let emptyList<'a> = new RandomAccessList<'a>([]);;
```

I intended to make most of the implementation unexposed, but when I started adding accessibility annotations I was warned by the compiler than these are not yet fully implemented.

Loading the code into F #, I can exercise it at the command line.

```
> #load "c:/profiles/clive.tong/desktop/FunctionalDatatypes.fs";;
 $\dots$ 
type 'a Tree =
| Leaf of 'a
| Node of 'a * 'a Tree * 'a Tree
val treeLookup : int -> 'a Tree -> int -> 'a
val treeUpdate : int -> 'a Tree -> int -> 'a -> 'a Tree
val lookup : (int * 'a Tree) list -> int -> 'a
val update : (int * 'a Tree) list -> int -> 'a -> (int * 'a Tree) list
val cons : 'a -> (int * 'a Tree) list -> (int * 'a Tree) list
val head : ('a * 'b Tree) list -> 'b
val tail : (int * 'a Tree) list -> (int * 'a Tree) list
type RandomAccessList<'a> =
class
end
with
new : content:(int * 'a Tree) list -> RandomAccessList<'a>
member cons : item:'a -> RandomAccessList<'a>
member Item : i:int -> 'a with get
member head : 'a
member lookup : index:int -> 'a
member tail : unit -> RandomAccessList<'a>
member update : index:int * value:'a -> RandomAccessList<'a>
end
val emptyList : RandomAccessList<'a>

> open FunctionalDatatypes;;
 $\dots$ 
> let xxx = emptyList;;
 $\dots$ 
val xxx : RandomAccessList<'a>
```

```
> xxx.cons(2);;
val it : RandomAccessList<int>
= FSI_0002.FunctionalDatatypes+RandomAccessList'1[System.Int32] {Item = ?;
  head = 2; }
> it.cons(40);;
val it : RandomAccessList<int>
= FSI_0002.FunctionalDatatypes+RandomAccessList'1[System.Int32] {Item = ?;
  head = 40; }
> let yyy = it;;
```

```
val yyy : RandomAccessList<int>
```

```
> yyy.[0];;
val it : int = 40
> yyy.[1];;
val it : int = 2
>
```

Better still, I can compile the code into an assembly and call into it from C #.

```
fsc.exe -a FunctionalDatatypes.fs
```

And then call it.

```
FunctionalDatatypes.RandomAccessList<int> list = FunctionalDatatypes.emptyList<int>();
list = list.cons(1);
list = list.cons(2);
```

```
Console.WriteLine(list.head);
list = list.tail();
Console.WriteLine(list.head);
```

I realise there is work to do to expose the class according to the design guidelines, but this is easiest example I have ever come across of exposing an implementation in one language for consumption in another.

1. <http://lambda-the-ultimate.org/>
  2. <http://citeseer.ist.psu.edu/27428.html>
- 

### **2.4.3 Fail OrElse.... (2008-04-20 20:31)**

A year or two ago, [1]software transactional memory started getting mentioned all over the place. This is a technique for wrapping application locks in a way that hides the underlying locking from application programmers. Over the weekend, I though I'd have a play around implementing some of this technique using F #.

We'll be using Dictionary objects to record the reads and writes that happen during the execution, so we need access to the .NET framework dictionary type.

```
#light;;
```

```
open System.Collections.Generic;;
```

A object behind the TransactionLog interface will manage a number of indexed locations of a fixed type ('a). We will have to be able to see the value of a location as viewed via the lens of the transaction (VisibleValue) and will be able to see if the transaction has read the value (TransactionHasRead). We will also be able to tell the transaction to commit (using Commit()). This will internally use an overloaded version of the method to do the work of committing.

```
type 'a TransactionLog() =
abstract VisibleValue : int -> 'a
abstract TransactionHasRead : int -> bool
abstract Commit : unit -> bool
abstract Commit : Dictionary<int, 'a> list * Dictionary<int, 'a> list -> bool
```

Our transactions will nest (so that we can later implement a way of combining transactions), but at the base level we'll have an array of memory that has software transactions running on top of it.

```
type 'a TransactionalMemory (memory : 'a[]) =
inherit TransactionLog<'a>()
override v.VisibleValue(index) = memory.[index]
override v.TransactionHasRead(index) = false
override v.Commit () = true
override v.Commit (reads, writes) =
lock memory
(fun _ ->
if List.for_all
(fun (dict : Dictionary<int, 'a>) ->
Seq.for_all (fun key -> memory.[key] = dict.[key]) dict.Keys)
reads
then List.iter
(fun (dict : Dictionary<int, 'a>) ->
Seq.iter (fun key -> memory.[key] <- dict.[key]) dict.Keys)
writes;
true
else false)
```

The Commit operation takes a list of values that have been read by the sequence of nested transactions, ordered by outermost transaction first, and a list of locations that have been written by the transactions. It locks the memory, checks that the values of the locations are still as they were during the first reads that happened as part of the transaction, and then replays all of the writes. If the values are not as they were read during the transaction, the commit fails.

Sitting on top of the transaction memory, we have a series of nested transactions.

```
type 'a NestedTransaction (innerTransaction : TransactionLog<'a>) =
inherit TransactionLog<'a>() as base
let reads = new System.Collections.Generic.Dictionary<int, 'a>()
let writes = new System.Collections.Generic.Dictionary<int, 'a>()
override v.VisibleValue(index) =
let success, value = writes.TryGetValue(index)
if success
then value
else innerTransaction.VisibleValue(index)
override v.TransactionHasRead(index) =
reads.ContainsKey(index) || innerTransaction.TransactionHasRead(index)
member v.Read(index) =
if v.TransactionHasRead(index)
then v.VisibleValue(index)
else
let visibleValue = innerTransaction.VisibleValue(index)
(reads.Add(index, visibleValue); visibleValue)
member v.Write(index, value) =
writes.[index] <- value
override v.Commit() =
v.Commit([reads], [writes])
override v.Commit(all_reads, all_writes) =
innerTransaction.Commit (reads::all_reads, writes::all_writes)
```

We can quickly try out the code:

```
> let test = [| 0; 0; 0; 0; |];;

val test : int array

> let transactionalBlock = new TransactionalMemory<int>(test);;

val transactionalBlock : int TransactionalMemory

> let transactionOverMemory = new NestedTransaction<int>(transactionalBlock);;

val transactionOverMemory : int NestedTransaction

> transactionOverMemory.Read(0);;
val it : int = 0
> transactionOverMemory.Write(0, 10);;
val it : unit = ()
> test;;
val it : int array = [|0; 0; 0; 0|]
> transactionOverMemory.Commit();;
val it : bool = true
> test;;
val it : int array = [|10; 0; 0; 0|]
```

However, if the value is changed before we run the transaction, it fails to commit.

```

> let test = [| 0; 0; 0; 0; |];;
val test : int array

> let transactionalBlock = new TransactionalMemory<int>(test);;

val transactionalBlock : int TransactionalMemory

> let transactionOverMemory = new NestedTransaction<int>(transactionalBlock);;
val transactionOverMemory : int NestedTransaction

> transactionOverMemory.Read(0);;
val it : int = 0
> transactionOverMemory.Write(0, 10);;
val it : unit = ()
> test.[0] <- 20;;
val it : unit = ()
> test;;
val it : int array = [|20; 0; 0; 0|]
> transactionOverMemory.Commit();;
val it : bool = false
> test;;
val it : int array = [|20; 0; 0; 0|]

```

That's all well and good, but what we really need is a more concise syntax for expressing transactions. F# has a way of doing this using computation expressions or workflows. These offer some of the facilities of monads.

```

type TransactionBuilder() =
    member v.Return(x) = fun transaction -> failwith "Transactions have no value"
    member v.Bind(p, rest) =
        fun transaction ->

```

```
match p transaction with
(result, currentTransaction) -> rest result currentTransaction
member v.Let(p, rest) = fun transaction -> rest p transaction
member v.Delay(f) = f()
member v.Zero () = fun transaction -> transaction
```

```
let transaction = new TransactionBuilder()
```

```
let ReadTVal index (v : 'a NestedTransaction) = (v.Read(index), v)
let WriteTVal index value (v : 'a NestedTransaction) = (v.Write(index,value), v)
```

We can now write out transactions in a more concise manner.

```
> let transaction1 =
- transaction {
- let! x = ReadTVal(0)
- let newValue = x + 1
- do! WriteTVal 0 (x+1)
- } ;;
```

```
val transaction1 : (int NestedTransaction -> int NestedTransaction)
```

And we can try this transaction out.

```
> let test = [| 0; 0; 0; 0; |];;
```

```
val test : int array
```

```
> let transactionalBlock = new TransactionalMemory<int>(test);;
```

```
val transactionalBlock : int TransactionalMemory
```

```
> let transactionOverMemory = new NestedTransaction<int>(transactionalBlock);;
```

```
val transactionOverMemory : int NestedTransaction
```

```
> let resultingTransaction = transactionOverMemory;;
```

```
val resultingTransaction : int NestedTransaction
```

```
> test;;
val it : int array = [|0; 0; 0; 0|]
> resultingTransaction.Commit();;
val it : bool = true
> test;;
val it : int array = [|1; 0; 0; 0|]
```

The workflow expression needs a little explaining, but I would like to leave that until another blog post on quasi-quoting in F#. Suffice it to say that the front end of the F# compiler will do transformation for certain expressions with syntax `expr { ... }`

We are getting there but we would like more. First it would be nice to allow transactions to be failed by calls inside the code. This will give us the ability to decide at runtime whether the transaction is allowed to proceed.

F# has a type named option that can be used to wrap values while offering an additional null value.

```
> Some 10;;
val it : int option = Some 10
> None;;
val it : 'a option = None
```

Instead of passing NestedTransaction values around, we'll pass NestedTransaction option values where the None case is used to represent an aborted transaction.

```
type TransactionBuilder() =
    member v.Return(x) = fun transaction -> failwith "Transactions have no value"
    member v.Bind(p, rest) =
        fun transaction ->
            match transaction with
            | None -> None
            | Some carriedTransaction ->
                match p carriedTransaction with
                | None -> None
                | Some (result, currentTransaction) -> rest result (Some currentTransaction)
    member v.Let(p, rest) =
        fun transaction ->
            match transaction with
            | None -> None
            | Some current -> rest p (Some current)
    member v.Delay(f) = f()
    member v.Zero () = fun transaction -> transaction
    member v.Combine (p,q) =
        fun transaction ->
            match transaction with
            | None -> None
```

```
| carriedTransaction ->
match p carriedTransaction with
None -> None
| resultTransaction -> q resultTransaction
```

```
let transaction = new TransactionBuilder()
```

We can now define fail.

```
let fail = fun transaction -> None
```

We have to redefine the read and write values code.

```
let ReadTVal index (v : 'a NestedTransaction) = Some (v.Read(index), v)
```

```
let WriteTVal index value (v : 'a NestedTransaction) = Some (v.Write(index,value), v)
```

We can now define a transaction that increments only if the value is greater than zero.

```
let transaction2 =
transaction { let! x = ReadTVal(0)
if x=0 then return! fail
let newValue = x + 1
do! WriteTVal 0 (x+1)
}
```

Next we can define something that runs a transaction and commits.

```
let runAndCommitTransaction action (memory : 'a array) =
let transactionalBlock = new TransactionalMemory<'a>(memory)
let transactionOverMemory = new NestedTransaction<'a>(transactionalBlock )
let finalTransaction = action (Some transactionOverMemory)
match finalTransaction with
None -> false
| Some (transaction : 'a NestedTransaction) -> transaction.Commit()
```

We can test it out as follows.

```
> let test = [| 0;1;2;3;4; |];;
```

```
val test : int array
```

```
> runAndCommitTransaction transaction2 test;;
val it : bool = false
> let test = [| 0;1;2;3;4; |];;
```

```

val test : int array

> test.[0] <- 10;;
val it : unit = ()
> runAndCommitTransaction transaction2 test;;
val it : bool = true
> test;;
val it : int array = [|11; 1; 2; 3; 4|]

```

These kinds of transactions are very powerful in that they are composable. We can define higher level operations to compose them.

The first, Then, takes two transactions and runs one before the other, failing if either fails.

```

let Then action1 action2 =
  fun transaction ->
    match transaction with
    None -> None
    | Some _ as currentTransaction ->
      match action1 currentTransaction with
      None -> None
      | result -> action2 result

```

The second, OrElse, takes two transactions. Its result is the result of the first transaction if that is successful, or the result of the second if the first transaction fails.

```

let OrElse action1 action2 =
  fun transaction ->

```

```

match transaction with
None -> None
| Some currentTransaction ->
let nestedTransaction = new NestedTransaction<int>(currentTransactio n)
match action1 (Some nestedTransaction) with
None ->
let nestedTransaction2 = new NestedTransaction<int>(currentTransaction)
action2 (Some nestedTransaction2)
| result -> result

```

We could now define two predicates for testing a transactional location for zero and a means for incrementing it.

```

let incrementLocation index =
transaction { let! current = ReadTVal index
let newValue = current + 1
do! WriteTVal index newValue
}

```

```

let checkLocationValue index value =
transaction { let! current = ReadTVal(index)
if current = 0 then return! fail
}

```

We can now define a composite transaction.

```

let testTransaction transaction =
OrElse (Then (checkLocationValue 0 0)
(incrementLocation 0))
(Then (checkLocationValue 1 0)

```

```
(incrementLocation 1))  
transaction;;
```

If the first branch of the OrElse is applicable, the composite transaction only depends on the first transactional location not changing value to enable a commit.

```
let test = [| 2; 0; 0; 0; |];;  
let transactionalBlock = new TransactionalMemory<int>(test);;  
let transactionOverMemory = new NestedTransaction<int>(transactionalBloc k)  
let result = match testTransaction (Some transactionOverMemory) with Some x -> x;;  
  
> result.TransactionHasRead 0;;  
val it : bool = true  
> result.TransactionHasRead 1;;  
val it : bool = false  
> result.Commit();;  
val it : bool = true  
> test;;  
val it : int array = [|3; 0; 0; 0|]
```

Likewise, if the second branch takes effect, then the only precondition is the second transactional location.

```
let test = [| 0; 2; 0; 0; |];;  
let transactionalBlock = new TransactionalMemory<int>(test);;  
let transactionOverMemory = new NestedTransaction<int>(transactionalBloc k)  
let result = match testTransaction (Some transactionOverMemory) with Some x -> x;;
```

```
> result.TransactionHasRead 0;;
val it : bool = false
> result.TransactionHasRead 1;;
val it : bool = true
> result.Commit();;
val it : bool = true
> test;;
val it : int array = [|0; 3; 0; 0|]
```

The transaction will fail if neither branch is applicable.

```
let test = [| 0; 0; 0; 0; |];
let transactionalBlock = new TransactionalMemory<int>(test);;
let transactionOverMemory = new NestedTransaction<int>(transactionalBloc k)
```

```
> testTransaction (Some transactionOverMemory) ;;
val it : int NestedTransaction option = None
```

And will still fail if the precondition is invalidated.

```
let test = [| 2; 0; 0; 0; |];
let transactionalBlock = new TransactionalMemory<int>(test);;
let transactionOverMemory = new NestedTransaction<int>(transactionalBloc k)
let result = match testTransaction (Some transactionOverMemory) with Some x -> x;;
test.[0] <- 20;;
```

```
> result.Commit();  
val it : bool = false
```

Transactional memory has given us a means of doing transactions without needing to do any locking at the user code level, whilst offering us the ability to define mechanisms for composing transactions in really powerful ways. Most articles go on to define, `Retry`, a primitive that causes the transaction to be repeated. Some implementations of this use the locations read set to avoid (in the context of functional transactions) re-executing the transaction until the values that the transaction previously read have changed. Essentially we could do this by having a third alternative to `Some/None` which passes back the read collection for the retrying collection.

Like all abstractions there are disadvantages to this kind of scheme. For the small number of lines of code it takes to implement it, it is a powerful abstraction though.

1. [http://en.wikipedia.org/wiki/Software\\_transactional\\_memory](http://en.wikipedia.org/wiki/Software_transactional_memory)

---

#### 2.4.4 What does that denote? (2008-04-21 07:10)

[1]Denotational semantics lecture notes by Professor Glynn Winskel

A great introduction to domains and continuous functions between them, together with constructions on domains and proof via Scott induction.

An operational and a denotational semantics for the functional language PCF is stated. The denotational model is shown to be sound, compositional and adequate.

1. <http://www.cl.cam.ac.uk/~gw104/dens.pdf>

---

## 2.5 May

### 2.5.1 Just get on with it will you! (2008-05-01 07:50)

Concurrency seems to be harder than it ought to be, even when you take that into account.

I was doing a code review a couple of days ago. The code in question had a main thread that launched a series of worker threads and a timer event that reported on their progress. The structure of the main thread (when simplified) looked like:

```
AutoResetEvent xx = new AutoResetEvent(false);
AutoResetEvent yy = new AutoResetEvent(false);
WaitHandle[] handles = new WaitHandle[] { xx, yy };
ThreadPool.QueueUserWorkItem(DoWork, xx);
ThreadPool.QueueUserWorkItem(DoWork2, yy);
WaitHandle.WaitAll(handles);
// Output state of workers using timed event
WaitHandle.WaitAll(handles);
```

The workers were structured as follows

```
AutoResetEvent eventToSignal = state as AutoResetEvent;  
eventToSignal.Set();  
// Do some processing  
  
eventToSignal.Set();
```

What I realised, when I looked at it, is that I didn't know at what point the auto reset event gets reset to a state when it can receive the next Set method call. If it isn't reset, then subsequent calls to Set will effectively be lost. Reading the documentation on [1]WaitAll didn't give much of a clue, though the documentation on [2]AutoResetEvent did hint at how the WaitAll and the AutoResetEvent interact. It is only when a thread is released that the events are reset. This is obvious when you think about it as there could be multiple WaitAll methods waiting for arrays of events that contain the same handle. You wouldn't want one of them to be picked arbitrarily as the WaitAll that resets the event and it would be hard to have the two WaitAll methods coordinating after resetting the event. Sure enough, changing one of the worker threads to sleep before calling Set for the first time, causes the main thread to hang on the second WaitAll as the first worker's second call to Set is lost.

This made me realise yet again how subtle some of this concurrency code can be. The example this was extracted from would work virtually all of the time as the work section of the threads would usually take several minutes. It would only be when the thread pool was short of worker threads, so that worker1 got time to run and finish before worker2 started, or when the system hit an extreme condition that slowed one of the threads down when the bug would become apparent. This kind of thing would be quite hard to find using black box testing without injecting delays into the thread code. Maybe it is down to people correctly reasoning about code behaviour, but then the documentation should make the behaviour of some of the concurrency primitives much more obvious so that people's mental model is more accurate.

1. <http://msdn.microsoft.com/en-us/library/z6w25xa6.aspx>

2. <http://msdn.microsoft.com/en-us/library/system.threading.autoresetevent.aspx>

---

## 2.5.2 Abstraction isn't always good for you (2008-05-01 08:15)

I was reminded of the paper by Andrew Kennedy [1]securing the .NET programming model by a conversation the other day about how interfaces are implemented on the CLR.

The CLR expects interface methods to be virtual but the C # language doesn't require this so it effectively virtualises them behind your back.

For example taking the code:

```
interface IFoo
{
    void Method1();
}

class Program //: IFoo
{
    public void Method1()
    {
    }

    static void Main(string[] args)
    {
        MethodInfo method = typeof(Program).GetMethod("Method1");
        Console.WriteLine("IsVirtual " + method.IsVirtual);
    }
}
```

If we compile and run it we see that Method1 is non-virtual as you'd expect. However, if we uncomment the interface declaration on the class, we see that the method suddenly becomes virtual and final.

The IL changes from

```
.method public hidebysig instance [2]void [3]Method1() cil managed
{
    .maxstack 8
    L_0000: ret
}
```

to

```
.method public hidebysig newslot virtual final instance [4]void [5]Method1()
cil managed
{
    .maxstack 8
    L_0000: ret
}
```

It's often stated that C # is the assembly language of the CLR, but examples like this raise the question about how valid it is to reason about the .NET framework and the CLR using C #.

1. <http://research.microsoft.com/~akenn/sec/appsem-tcs.pdf>
  2. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Void>
  3. [http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication8:1.0.0.0/ConsoleApplication8.Program/Method1\(\)](http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication8:1.0.0.0/ConsoleApplication8.Program/Method1())
  4. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Void>
  5. [http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication8:1.0.0.0/ConsoleApplication8.Program/Method1\(\)](http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication8:1.0.0.0/ConsoleApplication8.Program/Method1())
- 

### 2.5.3 That's life (2008-05-14 16:05)

Inspired by this month's [1]IBM Ponder, this is the company that I work at.

.....X.....X.....  
....X.....X..X.X.X.....X.....X..X.  
....XXXX...X.XXX....X...XX.X.XX.X.X.X..X..XX.....X.  
.....X....X..XX....XX..XXXX..X.X..X.XXXX.XX...  
....X..X..XX.XX.....XXX.XXX..X.....X.X.....  
....XXX....X..XX..XXX.X..X.....X..X...XX.X..X..X.  
....X.....XXX..X.....X..X.X.XXXXXX....XXXXX.....  
....X.X.X...X..X..XX...X.....X..X....X.XX...XX.X.  
.....X.....X.X.....

1. [http://domino.research.ibm.com/Comm/wwwr\\_ponder.nsf/Challenges/May2008.html](http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/Challenges/May2008.html)

---

## **2.5.4 It's very lively isn't it! (2008-05-28 17:36)**

I've got a confession to make... I've always had a soft spot for Smalltalk. It all started in the mid-90's when I read the book [1]Smalltalk-80: Bits of History, Words Of Advice This told the story of the porting of the Smalltalk virtual machine onto a number of different platforms, allowing them to run the base Smalltalk-80 image released by Xerox Parc. The idea of having a single image containing the whole smalltalk environment which could be ported to many different machines just by implementing a [2]small virtual machine was very impressive. At the time it was even more cutting edge because the machine supported windows and a mouse, with the windowing mainly achieved by the use of a single BitBlt operation acting on a bit mapped display which it used for doing all of the windowing operations.

The Smalltalk language itself impressed me greatly. The language is pretty simple with little syntax and is conceptually simple as every operation is regarded as a message send to an object which can be as simple as an integer or as complicated as compiler. The power comes from the libraries which come as part of the VM image.

A couple of months ago I was listening to the talks at the [3]2008 lang.NET symposium when one of the smalltalk names I [4]recognised cropped up. This was a presentation of the [5]Lively Kernel, a javascript implementation of the Morphic user interface that has been available for some time on Squeak smalltalk. The implementation is entirely in javascript, but uses [6]SVG for rendering the graphics elements. Events raised on the SVG elements are converted into events that are processed by the javascript code which can then use the browser's DOM to adjust the SVG elements.

Unfortunately, the system only runs on a few browsers which have good enough SVG support to make the system worth using. It is also, at the current time, rather slow.

In my experience there is only one good way to learn about a graphics system. This is to write the game of pong. This game requires animation type graphics (which one would like to make flicker free), event handling and timer events so it touches most parts of a window library.

Armed with build 2.0.0.14 of Firefox and the Venkman javascript debugger add-on I set off.

The first thing to note is that the lively kernel includes a version of prototype, a javascript library for aiding the construction of large code bases in javascript. In particular, the [7]prototype library allows javascript entities to be defined with a notion of inheritance.

```
Object.subclass('Animal',  
{
```

```
    initialize: function(name)  
{  
    this.name = name;  
},
```

```
    getName : function()  
{  
    return this.name;  
},
```

```
    makeNoise: function()  
{  
    alert("Generic animals can't make a noise");  
}  
});
```

```

Animal.subclass("Bear",
{
  initialize: function( $super, name)
  {
    $super(name);
  },
  makeNoise: function()
  {
    alert(this.getName() + ": growl");
  }
});
rupert = new Bear("Rupert");

```

In the above code fragment, the prototype library takes special notice of an initial parameter named \$super, and redefines the name within the scope of the function body to mean the function defined in the parent class. Hence, the initialize function on Bear calls into the initialize function on Animal, so that rupert.makeNoise will alert with the string "rupert: growl". We can use the same idea to define a subclass of the standard Morph class, which can play the game of pong.

The WindowMorph allows the area of the screen to have a title.

```
WindowMorph.subclass('PongMorph', {
```

We define a conventional initialize function to setup the state of the object. The pong game consists of a PanelMorph (a panel) on which we place the two bats and a ball.

```
initialize: function( $super, position) {
```

```
pongBoard = new PanelMorph(pt(300,300));  
$super(pongBoard, 'Pong', position);
```

```
playerx = 20;  
playery = 20;  
playerBat = Morph.makeLine([pt(20,20),pt(20,50)],4,C olor.blue);  
playerBatPosition = new Rectangle(20,20,4,30);  
pongBoard.addMorph(playerBat );
```

```
computerx = 280;  
computery = 20;  
computerBat = Morph.makeLine([pt(280,20),pt(280,50)],4 ,Color.blue);  
computerBatPosition = new Rectangle(280,20,4,30);  
pongBoard.addMorph(computerBat);
```

```
ballSpeed = 5;
ballx = 100;
bally = 100;
ballxdiff = ballSpeed;
ballydiff = ballSpeed;
ballPosition = new Rectangle(ballx,bally,20,20);
ball = Morph.makeCircle(pt(ballx,bally), 10, 1, Color.blue, Color.blue);
pongBoard.addMorph(ball);
```

```
boardExtent = pongBoard.getExtent();
return this;
},
```

The startSteppingScripts function starts a timer that will call the moveBall function ten times a second.

```
startSteppingScripts: function() {
this.startStepping(100, "moveBall");
},
```

The moveBall function checks if the ball is sufficiently close to a bat to cause it to rebound. If not, we can check for a bounce against the walls. Lastly we can check for whether the computer should move its bat, which it will do 70 % of the time in the direction of the ball.

```
moveBall: function()
{
var dist = ballPosition.dist(playerBatPosition);
if (dist == 0)
{
ballxdiff = -ballxdiff;
}
```

```

var dist = ballPosition.dist(computerBatPosition);
if (dist == 0)
{
ballxdiff = -ballxdiff;
}

if (ballx + ballxdiff <= 0 || ballx + ballPosition.width + ballxdiff >= boardExtent.x)
{
ballxdiff = -ballxdiff;
}
else
{
ballx += ballxdiff;
}
if (bally + ballydiff <= 0 || bally + ballPosition.height + ballydiff >= boardExtent.y)
{
ballydiff = -ballydiff;
}
else
{
bally += ballydiff;
}

ball.setPosition(pt(ballx, bally));
ballPosition.x = ballx;
ballPosition.y = bally;

if(Math.random() < 0.7)
{
if (bally > computery)
{
computery += ballSpeed;
computerBatPosition.y = computery;
computerBat.setPosition(pt(computerx, computery));
}
if (bally < computery)
{
computery -= ballSpeed;
computerBatPosition.y = computery;
computerBat.setPosition(pt(computerx, computery));
}
}
},

```

We need to say that the Morph handles the mouse down event

```
handlesMouseDown: function () {  
    return true;  
},
```

And when it gets it, we request the keyboard focus so that we can process keys corresponding to the movement of the user bat, in this case, the up and down arrow keys.

```
onMouseDown : function ( $super,evt) {  
    this.requestKeyboardFocus(evt.hand);  
    $super(evt);  
    return true;  
},
```

These functions allow us to accept the keyboard focus.

```
takesKeyboardFocus: function() {  
    return true;  
},
```

```
setHasKeyboardFocus: function(newSetting) {  
    return newSetting;  
},
```

If the up or down arrow keys are pressed, then we need we move the bat in the appropriate direction. If keyboard key is held down and repeats we can generate so many events that the timer events do not get a chance to fire, so we do a cycle of the timer action of moving the ball on processing each keyboard event.

```
onKeyDown: function (event) {
var key = event.getKeyCode() || event.charCode;
if (key == Event.KEY_UP)
{
if (playery - ballSpeed >= 0)
{
playery = playery - ballSpeed;
playerBat.setPosition(pt(playerx, playery));
}
}
if (key == Event.KEY_DOWN)
{
if (playery + playerBatPosition.height + ballSpeed <= boardExtent.y)
{
playery = playery + ballSpeed;
playerBat.setPosition(pt(playerx, playery));
}
}
playerBatPosition.x = playerx;
playerBatPosition.y = playery;
event.stop();
this.moveBall();
},
```

This prevents the system drawing a focus halo after the pong area is selected.

```
onFocus: function () { },  
});
```

Finally we need to make an instance of this class. I did this by putting the following code in the populateWorldWithExamples function in main.js in the lively kernel 0.8 distribution which I ran locally.

```
var pong = new PongMorph(pt(200,200).extent(300,600));  
world.addMorph(pong);  
pong.startSteppingScripts();
```

When we run it we get something like [8]<http://cid-3f21df299c355e7f.skydrive.live.com/self-.aspx/Pictures/pong.jpg>

What did I learn from the experience?

That currently the javascript/SVG interpreter combination is not yet fast enough to make develop a painless process. The javascript code is single threaded, so it is hard to do too much work inside timer callbacks without the whole GUI slowing down dramatically. The lively kernel library contains a lot of code for making it easy to do XMLHTTP calls so it would be possible to push some of this work onto the web server.

SVG allows the graphics to be quite spectacular.

There is an inbuilt model/view/controller setup with changed events being propagated from the model into the views to allow multiple views on the same data.

Included in the distribution there are a number of browsers which look a lot like the smalltalk object and method browsers. With these, and extra reflection facilities supplied by the prototype library, it is easy to see the javascript code for a given method on a given class. Unfortunately the methods are not sorted into protocols so it is really hard to see which ones fit together. I had a lot of problems dealing with the keyboard focus and still haven't got it working properly.

The use of javascript is neat, but the debugging experience isn't the same as using real smalltalk. I often had to resort to editing the lively kernel javascript source to include a debugger call to allow me to single step the code in Venkman. In Smalltalk I'd be able to embed a break into a method without the whole system stopping while I debugged but the single threaded nature of the browser stops this.

There are methods for serialising the Morph objects back to the web server from which they came. I couldn't find enough documentation on this to experiment much with this part of the system.

Writing and developing in javascript is certainly one way to go. Another is to do what other projects are doing and regard javascript as the target language for a compiler and framework written in another high level language. In other words to use javascript as the assembly language of the net rather than the implementation language for large projects. It will be interesting to see how the lively kernel goes in the future.

1. [http://www.amazon.co.uk/Smalltalk-80-Bits-History-Words-Advice/dp/0201116693/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1211826571&sr=1-1](http://www.amazon.co.uk/Smalltalk-80-Bits-History-Words-Advice/dp/0201116693/ref=sr_1_1?ie=UTF8&s=books&qid=1211826571&sr=1-1)
  2. <http://wiki.squeak.org/squeak/1602>
  3. <http://www.langnetsymposium.com/speakers.asp>
  4. <http://www.squeakland.org/community/biography/ingalls.html>
  5. <http://research.sun.com/projects/lively/>
  6. <http://www.w3.org/Graphics/SVG/>
  7. <http://www.prototypejs.org/>
  8. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Pictures/pong.jpg>
- 

## 2.6 June

### 2.6.1 It's all about the people (2008-06-10 08:55)

[1]Peopleware: Productive Projects and Teams

The project leader of my new job gave me this book to read in my first week, and I'm very glad he did. This book is full of lots of good observations about getting the best out of software developers. Some of the observations are obvious and some are a little counter-intuitive, but all are backed up by data and anecdotes from the writers' project management experience.

I enjoyed the observations on flow, that time when you work exclusively on some problem whilst paying no notice of the world around you, so that when you next look up at a clock several hours have passed. According to psychologists, it takes up to 15 minutes to enter this state of mind, so it is important that the work environment is distraction free to allow people to enter this state of mind where they work very effectively.

The book also talks about methods for getting a team to gel, to get them all to work to the same goal. Peer pressure within the team then encourages everyone to pull their weight. Some

of the ideas here reminded me of the things that I associate with agile project management. People become the centre of things: the team members set the next short term goals of the iteration, effectively getting buy in from the team, while short, say two week, iterations allow the estimates of the developers to be compared against the percentage completion figures (the velocity) allowing management to estimate how long things will really take. Dependencies that were not obvious on day one will surface as the project moves forward and can be moved into the next iteration if someone is being held up. The team gets a good feeling if the iteration achieves all of the goals for the iteration, thought missing a target is not demoralising as they get another chance in a very short space of time. This causes the team to work at a sustainable rate, avoiding the situations where a massive effort is made to hit a deadline which causes the team to burn out making them less effective.

- 
1. [http://www.amazon.co.uk/Peopleware-Productive-Projects-Tom-DeMarco/dp/0932633439/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1213082579&sr=1-1](http://www.amazon.co.uk/Peopleware-Productive-Projects-Tom-DeMarco/dp/0932633439/ref=sr_1_1?ie=UTF8&s=books&qid=1213082579&sr=1-1)

## 2.6.2 No need to apologise (2008-06-10 09:17)

[1]A Mathematician's Apology

I've heard people mention this book over the years and found it half-price in a Heffers closing down sale. I really enjoyed the mini-biography of Hardy by C.P.Snow which takes up close to half of the book. The notes and remarks in the section of the book by Hardy were interesting, being an insight into the creative process and the worth of devoting one's life to mathematics, but somehow the book just didn't grab me.

- 
1. [http://www.amazon.co.uk/Mathematicians-Apology-Canto/dp/0521427061/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1213088218&sr=1-1](http://www.amazon.co.uk/Mathematicians-Apology-Canto/dp/0521427061/ref=sr_1_1?ie=UTF8&s=books&qid=1213088218&sr=1-1)

### **2.6.3 You're a bit remote today (2008-06-10 13:18)**

[1]Advanced .NET Remoting

I had read the first version of this book several years ago, but the second edition was far more interesting. The book gives the high level view of remoting with tips on using the technology in real world scenarios, together with a second half of the book which plumbs into the low level details, showing how a client side call on a remoted object passes through the transparent proxy, gets the converted into an IMessage and then details how this passes through the stack of message sinks, formatters and channel sinks to arrive at the server. At the server it then goes into detail about how the call is dispatched and the results are marshalled back to the client.

There are many examples of how objects can be added to this pipeline to customise things. Examples include compressing the data stream, encrypting the data stream and responding to the mime type of the result, together with an example of developing a transport using SMTP. There is also a whole section on versioning which is a useful read for anyone who is going to deploy this technology for real.

My only quibble was the quote in the book "Both CORBA and DCOM have employed distributed reference counting". This is plain wrong. CORBA never had distributed reference counting according to the standard. Some of the language binding such as the C++ binding used local reference counting to clear up the client side stubs, but destruction of objects needed to be an explicit operation on the object itself or the object adaptor that was responsible for managing it.

Digging into the remoting architecture reminded me of the CORBA work that I did in the late 90's. I worked hard at implementing a Common Lisp ORB for the Lisp vendor for whom I worked at the time. We collaborated with [2]Franz on the [3]OMG Common Lisp Language mapping. The actual means of getting messages across to the server was very much like the process that happens inside .NET. A request package was generated that contained details of the target object together with the name of the operation to be invoked and values for the parameters. This request was submitted to another object which marshalled the request package and sent

it across tcp to the target server. When the result came back, the results were placed into the initial request package as a response. As reflection wasn't generally available, the link between the initial function call and the generation of the request package was via a stub that was generated by compiling an IDL definition to generate functions that would do the necessary construction. The sink chain had an equivalent in that interceptors could be set up to run either on the request/response packet or on the marshalled data stream before the client sent the data or after it received it. Likewise for the server. The mechanism for finding the target object was by looking down a tree of object adapters; an object reference essentially gave a search path through this tree. This was neat because the non-root adapters could be started lazily. The adapters were responsible for generating the servant - the object that would handle the request, and were responsible for deleting it, either under the control of an API call from a client or by some other process. There was nothing built-in to do the lease sponsoring of .NET or the distributed GC of DCOM.

The .NET scheme is certainly more flexible than the CORBA scheme in terms of the message passing, and it offers the possibility of working across more transport types. All very interesting.

1. [http://www.amazon.co.uk/Advanced-Remoting-Second-Szpuszta-Rammer/dp/1590594177/ref=sr\\_1\\_2?ie=UTF8&s=books&qid=1213095448&sr=1-2](http://www.amazon.co.uk/Advanced-Remoting-Second-Szpuszta-Rammer/dp/1590594177/ref=sr_1_2?ie=UTF8&s=books&qid=1213095448&sr=1-2)
  2. <http://www.franz.com/>
  3. [http://www.omg.org/technology/documents/formal/lisp\\_language\\_mapping.htm](http://www.omg.org/technology/documents/formal/lisp_language_mapping.htm)
- 

#### 2.6.4 I'm an actor! (2008-06-11 07:26)

I quite like the actor model of concurrency and was reminded of it by a recent edition of [1] the java posse where some concurrency experts gave their options on this and software transactional memory. In the actor model, objects communicate by messaging each other. In contrast to the shared state model of concurrency, there is no explicit locking as it is all hidden away inside the code which handles the mail boxes (where the queues could be manipulated using lock free techniques). F# offers a mailbox processor type for doing this kind of thing.

The typical example of this is the harmless counter which responds to messages telling it to increment itself and to pass its current value along a channel. In the following code we make

the counter two state. In the initial state it appears as above, but it can be told to change to a locked state where increment isn't allowed.

```
> #light;;
> open Microsoft.FSharp.Control.Mailboxes;;
```

Define the messages that the counter will understand.

```
> type Action= Read of IChannel<int> | Increment of int | Lock;;
```

```
type Action =
| Read of IChannel<int>
| Increment of int
| Lock
```

Define the counter. The first function handles the initial state, the second function handles the state where the counter will not allow an increment.

```
> let makeCounter initialCount =
- let count = ref initialCount
- let counter = MailboxProcessor.Start( fun inbox ->
- let rec increment =
- async { let! msg = inbox.Receive()
- match msg with
- | Read channel ->
- do channel.Post(!count)
- return! increment
```

```
- | Increment amount ->
- do count := !count + amount
- return! increment
- | Lock -> return! readOnly }
- and readOnly =
- async { let! msg = inbox.Receive()
- match msg with
- | Lock -> return! readOnly
- | Increment _ -> do failwith "Locked"
- | Read channel ->
- do channel.Post(!count)
- return! readOnly }
- increment)
- counter;;
```

return! increment

-----^ ^ ^ ^ ^sciicircum^ ^

stdin(12,26): warning: FS0040: This and other recursive references will be checked for initialization-soundness at runtime because you are defining a recursive object or function value, rather than a simple recursive function. This warning is often harmless, and may be suppressed by using #nowarn "40" or -no-warn 40

val makeCounter : int -> MailboxProcessor<Action>

> open System;;

Define an instance to act as a message sink which prints any value that is sent to it.

```
> let sink =
- { new IChannel<int>
- with Post (x) = printfn "Value is %d" x; };;
```

```
val sink : IChannel<int>
```

```
> let counter1 = makeCounter 5;;
```

```
val counter1 : MailboxProcessor<Action>
```

```
> counter1.Post (Read sink);;
Value is 5
val it : unit = ()
> counter1.Post (Increment 2);;
val it : unit = ()
> counter1.Post (Increment 2);;
val it : unit = ()
> counter1.Post (Lock);;
val it : unit = ()
> counter1.Post (Read sink);;
val it : unit = ()
Value is 9
```

The counter is now locked so when we do the following, an exception is thrown.

```
> counter1.Post (Increment 2);;
```

1. [http://www.javaposse.com/index.php?post\\_id=346291](http://www.javaposse.com/index.php?post_id=346291)

## 2.7 July

### 2.7.1 What difference does it make? (2008-07-06 17:05)

A while ago I came across the paper [1]A Formal Investigation Of Diff3. I'd never really thought before about how such three way merge algorithms work and since the diff3 algorithm uses an underlying algorithm for diff, which I also knew nothing about, I thought it would be fun to try doing these calculations in F#. In the real diff implementation we would be working with lines of characters, treating each line as an atomic value. Here we use strings dealing with the individual characters.

One (inefficient) algorithm for finding the shortest edit distance between two strings, where this is defined as the smallest number of single character add and delete operations required to make the strings equal, is to calculate a distance matrix between the two strings. The "from" string forms the rows and the "to" string forms the columns. Initially there is an extra column and row reflecting the number of steps to convert the empty string into the target. For example, converting "bad" to "sad", we'd draw an initial matrix of form.

```
. s a d  
. 0 1 2 3  
b 1  
a 2  
d 3
```

At each stage, an entry reflects the minimum distance to convert the part of the "from" string in rows above and including the current point into the "to" string in columns up to and including the current point ie when we place a value in the "d" column on the "a" row we are placing the value for the minimum number of steps to convert "ba" into "sad".

The algorithm for the calculating the next value is simply to make  $D(i,j) = 1 + \min(D(i-1,j), D(i,j-1))$  if the ith character of the "from" string and the jth character of the "to" string differ, or if they are equal we can also choose to do nothing and hence chose  $D(i,j) = \min(D(i-1,j-1), \dots)$  the above calculate ...). You can reason that this gives the minimum distance because for any minimum edit sequence we can also move the change positions as far to the left as possible to get initial sub-sequences that are minimal.

Hence we could start filling in the distances array as

```
. s a d  
. 0 1 2 3  
b 1 2 3 4  
a 2 3 2 3  
d 3 4 3 2
```

The bottom right gives us the final value, a distance of 2, and the steps are reflected by following a path from the bottom right back to (0,0). This would go along the values 2-2-2-1-0 which correspond to the edit sequence delete "b", insert "s", keep "a" and keep "d". Note that such minimal sequences are not necessarily unique and that there are optimisations on this algorithm to make it much faster. However, it is quick enough for my purposes.

We can easily implement this in F# code. F# is nice in that we can keep things mostly functional to allow us to reason about the implementation, but improve the speed by using a mutable array locally within a couple of functions, where we can see by eye what is going to be using it (and hence avoid any need for making it thread safe).

An edit consists of a pair of an edit operation and a character that is to be changed.

```
type EditOperation =  
| Keep  
| Insert  
| Delete
```

First we define routines to calculate the distance array and the path through it corresponding to the minimal edit sequence.

```

let ComputeDistanceArray fromString toString =
let fromLength = String.length fromString
let toLength = String.length toString
if fromLength = 0 || toLength = 0 then
  null
else
  let distanceArray = Array2.create (fromLength+1) (toLength+1) 0
  for j = 1 to toLength do distanceArray.[0,j] <- j
  for i = 1 to fromLength do distanceArray.[i,0] <- i
  for i = 1 to fromLength do
    for j = 1 to toLength do
      let nonEqual = 1 + min distanceArray.[i-1,j] distanceArray.[i,j-1]
      distanceArray.[i,j] <- if toString.[j-1] = fromString.[i-1] then min nonEqual distanceArray.[i-1,j-1] else nonEqual
  distanceArray

```

```

let MinimalPath (distanceArray : int[,]) (fromString : string) (toString : string) =
let rec computeMinimal i jsofar =
  if j = 0
  then if i = 0
    then sofar
  else computeMinimal (i-1) j ((Delete,fromString.Substring(i-1,1))::sofar)
  else if i = 0 then computeMinimal i (j-1) ((Insert,toString.Substring(j-1,1))::sofar)
  else
    let allowedDiagonal = fromString.[i-1] = toString.[j-1]
    let smallestNonDiagonal = min distanceArray.[i-1,j] distanceArray.[i,j-1]
    let smallestWithDiagonal = min distanceArray.[i-1,j-1] smallestNonDiagonal
    if allowedDiagonal && smallestWithDiagonal = distanceArray.[i-1,j-1]
    then computeMinimal (i-1) (j-1) ((Keep,toString.Substring(j-1,1)):: sofar)
    else if smallestNonDiagonal = distanceArray.[i-1,j]
    then computeMinimal (i-1) j ((Delete,fromString.Substring(i-1,1)):: sofar)
    else computeMinimal i (j-1) ((Insert,toString.Substring(j-1,1)):: sofar)
  computeMinimal (String.length fromString) (String.length toString) []

```

```
let CalculatePath from target =
let distances = ComputeDistanceArray from target
MinimalPath distances from target
```

This has basically given us a means of implementing diff.

```
let Diff from target =
let path = CalculatePath from target
let rec print path =
match path with
| [] -> ()
| (Insert,x)::rest ->
printfn "> %s" x
print rest
| (Delete,x)::rest ->
printfn "< %s" x
print rest
| _::rest -> print rest
print path
```

```
> CalculatePath "bad" "sad";;
val it : (EditOperation * string) list
= [(Insert, "s"); (Delete, "b"); (Keep, "a"); (Keep, "d")]
> Diff "bad" "sad";;
> s
< b
```

To display the final merge result we'll need a few routines for doing printing. We'll be passing tuples representing how far we are through the edit sequences between the original and the first version of the file, and the original and second version of the file.

```

let rec DisplayIfNot operation path stopAt =
if path <> stopAt
then
match path with
| (operation2, value)::rest ->
if operation <> operation2 then printfn "%s" value
DisplayIfNot operation rest stopAt
| _ -> failwith "stopAt should be the tail of path"

```

```

let rec SomethingToDisplayIfNot operation path stopAt =
if path <> stopAt
then
match path with
| (operation2, value)::rest ->
if operation <> operation2 then true else SomethingToDisplayIfNot operation rest stopAt
| _ -> false
else false

```

```

let Display (path1,path2) (end1, end2) =
if SomethingToDisplayIfNot Keep path1 end1
then
let originalNeedsToDisplay = SomethingToDisplayIfNot Insert path1 end1
if not(SomethingToDisplayIfNot Delete path1 end1) & & not originalNeedsToDisplay
then DisplayIfNot Delete path2 end2
else if not(SomethingToDisplayIfNot Delete path2 end2) & & not originalNeedsToDisplay
then DisplayIfNot Delete path1 end1
else
printfn "<<<<<<<<" 
DisplayIfNot Delete path1 end1
printfn "|||||||"
DisplayIfNot Insert path1 end1
printfn "======" 
DisplayIfNot Delete path2 end2
printfn ">>>>>>>>>" 
else
DisplayIfNot Delete path1 end1

```

We define worker functions to find the next stable and unstable sequences of the edit sequences.

```
let rec MoveWhileStable current =
  match current with
  | ([] , _) -> None
  | (_ , []) -> None
  | ((Keep, _)::rest1, (Keep, _)::rest2) -> MoveWhileStable (rest1,rest2)
  | ((Delete, _)::rest1, (Delete, _)::rest2) -> MoveWhileStable (rest1,rest2)
  | _ -> Some current

let rec MoveUntilStable current =
  match current with
  | (Keep,p):: rest1, (Keep,q)::rest2 -> Some current
  | (Insert, _)::rest1, rest2 -> MoveUntilStable (rest1,rest2)
  | (rest1, (Insert, _)::rest2) -> MoveUntilStable (rest1,rest2)
  | [], _ -> None
  | _,[] -> None
  | _::rest1, _::rest2 -> MoveUntilStable (rest1, rest2)
```

Lastly we define the worker algorithm for breaking the edits into stable and unstable sequences.

```
let rec ProcessDiff current =
  match current with
  | None -> None
  | Some (path1, []) ->
    Display (path1,[]) ([][])
    None
  | Some ([], path2) ->
    Display ([] ,path2) ([][])
    None
  | Some (path1, path2) ->
```

```

match MoveWhileStable (path1, path2) with
| None ->
Display (path1,path2) ([][])
None
| Some (non1, non2) ->
if non1 = path1
then
let unstable = MoveUntilStable (path1, path2)
match unstable with
| None ->
Display (path1, path2) ([][])
None
| Some seqEnd ->
Display (path1,path2) seqEnd
Some seqEnd
else
Display (path1,path2) (non1,non2)
Some (non1,non2)

```

Diff3 basically uses that function to batch and print out the results.

```

let Diff3 mine original yours =
let path1 = CalculatePath original mine
let path2 = CalculatePath original yours
Some (path1, path2)

```

We can run the example from the paper.

```

> Diff3 "145236" "123456" "124536";;
val it : ...

```

```
> ProcessDiff it;;
1
val it : ...
```

```
> ProcessDiff it;;
4
5
val it : ...
```

```
> ProcessDiff it;;
2
val it : ...
```

```
> ProcessDiff it;;
<<<<<<<<
3
|||||||
3
4
5
=====
4
5
3
>>>>>>>>
val it : ...
```

```
> ProcessDiff it;;
6
val it :
((EditOperation * string) list * (EditOperation * string) list) option
= None
```

The original paper makes a number of interesting observations, in particular that editing separate parts of the file in two different streams does not guarantee that diff3 will be able to merge the changes without conflict.

```
> Diff3 "121212" "1212" "123";;
val it : ...
```

```
> ProcessDiff it;;
1
2
val it : ...
```

```
> ProcessDiff it;;
<<<<<<<<<
1
2
1
2
||||||| |
1
2
=====
3
>>>>>>>>
val it : ((EditOperation * string) list * (EditOperation * string) list) option
= None
```

I think the paper is interesting. F# is sufficiently high level to allow easy experimentation with algorithms without having to write lots of helper code, making it a good language for exploratory programming.

1. <http://www.cis.upenn.edu/~bcpierce/papers/diff3-short.pdf>

## 2.7.2 You're a little too abstract for me (2008-07-07 12:18)

There's a good discussion on the contrast between using interfaces and abstract base classes in a [1]podcast with Juval Lowy Given my long background with COM I have always preferred to work via interfaces in the cases where we are interacting with an object via a particular contractual view. Obviously there are cases when an abstract class is good; when there is behaviour to share between the subclasses for example or when we need some kind of default behaviour (that traits would give us if they were part of the language).

To my mind interfaces are good because they give you a view on an object that hides some of the unnecessary features of the underlying object. This allows the object to be wrapped by libraries that provide additional functionality by means of intercepting method calls that happen via the interface. This kind of interception happens in frameworks that implement aspect oriented programming ideas, libraries such as the policy injection application block, and persistence frameworks such as NHibernate which need to guard the objects properties so that the object's state can be lazily loaded and so that modifications may be monitored and later written back to databases.

Microsoft Transaction Server used to do this kind of interception with COM objects. Metadata for the COM interfaces, in the form of type libraries, could be used to generate a proxy interface whose methods could later instantiate a real instance of the desired class to which interface calls could be delegated. This facilitated lazy creation and the ability to get the call into the correct context before it was allowed to proceed (e.g. setting up a transaction).

Interfaces are also very test friendly, particularly in the presence of dependency injection patterns which allow test versions of interfaces to be easily substituted during the execution of tests.

If interfaces aren't being used, then something like NHibernate finds it hard to inject a proxy into the picture (unless the object inherits from MarshalByRefObject in which cases the CLR's native proxy mechanism can be used). In such cases the proxy engine will typically dynamically define a subclass of the target class (which can't work if it is sealed) and override any virtual

methods in order to capture calls to them. Of course, this means that calls to non-virtual methods cannot be trapped.

For example, given a class such as the following:

```
public class A  
{  
    public virtual void Foo(int argument)  
    {  
        Console.WriteLine("Main method on " + argument);  
    }  
}
```

It would generate a proxy something like:

```
AppDomain currentDomain = AppDomain.CurrentDomain;  
AssemblyBuilder assemblyBuilder = currentDomain.DefineDynamicAssembly(new AssemblyName("Proxy"), AssemblyBuilderAccess.Run);  
ModuleBuilder moduleBuilder = assemblyBuilder.DefineDynamicModule("playModule");  
TypeBuilder typeBuilder = moduleBuilder.DefineType("AProxy", TypeAttributes.Class,  
typeof(A));  
MethodBuilder methodBuilder = typeBuilder.DefineMethod("Foo", MethodAttributes.Virtual |
```

```

MethodAttributes.Public,
typeof(void), new Type[] { typeof(int) });
ILGenerator ilGenerator = methodBuilder.GetILGenerator();
ilGenerator.Emit(OpCodes.Ldstr, "Proxy method called");
ilGenerator.Emit(OpCodes.Call, typeof(Console).GetMethod("WriteLine", new Type[] {
typeof(string) }));
ilGenerator.Emit(OpCodes.Ldarg_0);
ilGenerator.Emit(OpCodes.Ldarg_1);
ilGenerator.Emit(OpCodes.Call, typeof(A).GetMethod("Foo", new Type[] { typeof(int) }));
ilGenerator.Emit(OpCodes.Ret);
Type proxyType = typeBuilder.CreateType();

```

and instances could be made using code like

```

A target = (A) Activator.CreateInstance(proxyType);
target.Foo(10);

```

This would call the WriteLine inside the proxy before the call into the real method even though we only had to change the place in the original code when the instance is created to substitute the proxy.

Of course, user code could do type testing to see if the object were of the expected type

```

Console.WriteLine(target.GetType() == typeof(A));
Console.WriteLine(target2.GetType() == typeof(A));

```

but we have a fairly easy way to decorate the object with additional functionality.

---

1. <http://channel9.msdn.com/shows/ARCast.TV/ARCasttv-Juval-Lowy-on-Interface-Based-Design/>

### 2.7.3 It's a little surprising when you first see it (2008-07-07 14:49)

Coming from a dynamic languages background, where method dispatch is determined at runtime, operators in C# have taken me some time to get used to. They are resolved statically at compile time and this can lead to an unexpected operator being selected if types are not flowing in the way that you expect during the compilation. In the case of a generic class, the

operator is resolved at compile time (and not at instantiation time of the generic class). This means that operator selection happens using types inferred from where clauses (and other constraints) associated with the generic class definition.

In the following program, changing the where clause to "where T:A" gives a different result as a different operator is selected from the addition. The important point is that, despite looking a lot like C++ templates, C # generics are compiled once, and not once per type instantiation like the former.

```
public  
class  
A  
  
{  
  
    public  
    int m _Value = 0;  
  
    public A( int x)  
  
    {  
  
        m _Value = x;  
  
    }  
  
    public  
    static  
    A  
    operator +( A x, A y)
```

```
{  
    return  
    new  
    A (x.m_Value + y.m_Value);  
  
}  
  
}
```

```
public  
class  
B : A
```

```
{  
    public B( int x ) : base (x)  
  
    {  
    }  
  
    public  
    static  
    B  
    operator +( B x, B y)  
  
    {  
        return  
        new  
        B (x.m_Value + y.m_Value + 1);  
  
    }  
}
```

```
public
class
Play <T> where T: B

{

    T m_Value;

    public Play(T x)

    {

        m_Value = x;

    }

    public
    int Result()

    {

        return (m_Value + m_Value).m_Value;

    }

}

class
Program
```

```
{

static
void Main( string [] args)
```

```
{  
  
Play < B > yy = new  
Play < B >( new  
B (1));  
  
Console .WriteLine(yy.Result());  
  
}  
  
}
```

---

#### 2.7.4 It's not a question of "or" - it can be "and" (2008-07-07 18:47)

Around 1995 [1]we ported our Unix Common Lisp implementation onto the PC. Around this time Nick Levine and I had to do some work on delivery.

The Common Lisp system was delivered as an exe containing a small loader program which loaded a Lisp heap image into memory and started executing it. This was rather neat as virtually the whole system was written in Lisp, including the garbage collector, which meant that virtually all of the system could be patched by dynamically loading code to fix problems (patches were loaded early in the startup sequence so even bugs in the environment could be patched). People liked developing in the rich Lisp environment, but wanted to be able to deliver their applications to customers as standalone executables that didn't contain the Lisp programming environment, mainly to make them smaller. Previously we had facilitated this by also giving customers a "base" image, a cutdown image without the programming environment into which components and then the application code could be loaded. However, the need to have two separate images caused all sorts of problems. For the PC version, we therefore decided to have a single Lisp image and to improve the tree shaking to get rid of large chunks of the code that the user application didn't need. [It's called tree shaking because you hold the roots of the tree (the application) and shake the apple tree hard - anything not attached to the root via branches of code will fall away].

To make the tree shaking easy, the system mainly tried to clean away unneeded entry points and then get rid of information that could be reconstructed after a full garbage collection. Unfortunately there were particular problems around generic functions as defined by CLOS (the Common Lisp Object System). In CLOS the classes and the methods are separate entities. Methods are owned by generic functions which are generally bound to a symbol.

For example, here we define two classes A and B, with B a subclass of A. Two methods on a generic function FOO are defined, with the first specialising on an instance of A and the second on an instance of B.

```
CL-USER 1 > (defclass a()())
#<STANDARD-CLASS A 2170540B>
```

```
CL-USER 2 > (defclass b(a)())
#<STANDARD-CLASS B 216EC1CB>
```

```
CL-USER 3 > (defmethod foo ((x a)) (print "method on a called"))
#<STANDARD-METHOD FOO NIL (A) 200E52E3>
```

```
CL-USER 4 > (defmethod foo ((x b)) (print "method on b called"))
#<STANDARD-METHOD FOO NIL (B) 216DA093>
```

```
CL-USER 5 > (foo (make-instance 'a))
```

"method on a called"

```
CL-USER 6 > (foo (make-instance 'b))
```

"method on b called"

During the tree shaking we really wanted to say that the method specialised on B can be shaken away if either the class B can be shaken away or if the generic function FOO can be shaken away. This is difficult to express via normal objects because the garbage collector works on an "or" preservation principle; an object is kept if any of the objects pointing to it are kept.

The solution we came up with was named a gate which I added to the garbage collector. It is basically an object that points to a single target and is pointed to by a number of other objects. The target object is kept alive only if all of the objects that point to it are kept alive (or if the gate itself is referenced via another path). In the mark and sweep collector that we had, this simply meant that we only marked through to the target when all of the incoming objects had been marked.

Before I can demonstrate that, we need to have a quick look at the finaliser mechanism that the Lispworks system used. Finalisers were known as special free actions. When an object flagged for special free action was unmarked after the garbage collection, a set of special free action functions were called on it. These were what would now be called finalisers. The object would survive at least to the next collection depending on what it did. The finaliser ran on the

thread that triggered the garbage collection, and in rare occasions it could cause reentry to the garbage collector.

We define a class named Test.

```
CL-USER 7 > (defclass test()())  
#<STANDARD-CLASS TEST 2008F25B>
```

This flags all instances as requiring a special free action.

```
CL-USER 12 > (defmethod initialize-instance :after ((self test) &rest args)  
(flag-special-free-action self))  
#<STANDARD-METHOD INITIALIZE-INSTANCE (:AFTER) (TEST) 200B531F>
```

We define a finaliser for objects of type TEST and compile it.

```
CL-USER 14 > (defun check-for-test (x) (when (typep x 'test) (print "a Test instance was gc'ed")))  
CHECK-FOR-TEST
```

```
CL-USER 15 > (compile *)  
CHECK-FOR-TEST
```

```
NIL  
NIL
```

We add this function to the set of special free actions.

```
CL-USER 16 > (add-special-free-action 'CHECK-FOR-TEST)  
(CHECK-FOR-TEST EDITOR::FREE-BIG-CHUNK SYSTEM::COND-CLOSE-C-FILE-STREAM)
```

And test it, by making an instance and doing a garbage collection at which point we see the finaliser printing out a message.

```
CL-USER 17 > (progn (make-instance 'test) nil)  
NIL
```

```
CL-USER 18 > (mark-and-sweep 3)
```

```
"a Test instance was gc'ed"  
18436024  
18702000
```

```
CL-USER 19 > (mark-and-sweep 3)  
18436024
```

18702000

And this time make two instances which get collected.

```
CL-USER 20 > (progn (make-instance 'test) nil)
NIL
```

```
CL-USER 21 > (progn (make-instance 'test) nil)
NIL
```

```
CL-USER 22 > (mark-and-sweep 3)
```

"a Test instance was gc'ed"

"a Test instance was gc'ed"

18436024

18702000

To demonstrate the use of a gate we need some objects that contain pointers to the gate itself. We will use two cons cells that we will maintain references to.

```
CL-USER 1 > (defvar *a* (cons 1 1))
A*
```

```
CL-USER 2 > (defvar *b* (cons 2 2))
B*
```

We make an instance of a gate, with target an instance of the class TEST (which prints a message when it is collected).

```
CL-USER 12 > (defvar *gate* (sys::make-gate (make-instance 'test) *a* *b*))
GATE*
```

We put pointers into the two cons cells to point them at the gate.

```
CL-USER 13 > (progn (setf (car *a*) *gate* (car *b*) *gate*) nil)
NIL
```

We can now get rid of the reference from the variable to the gate (otherwise this will keep it fully alive and the target will not be collected).

```
CL-USER 14 > (setq *gate* nil)
NIL
```

Garbage collecting now will keep the instance of TEST alive as both \*a\* and \*b\* are kept alive by the system and these two variables reference the two cons cells that hold the incoming pointers into the gate.

```
CL-USER 15 > (mark-and-sweep 3)
```

```
18440920
```

```
18702000
```

Now, we release a hold on the first cons cell.

```
CL-USER 16 > (setq *a* nil)
```

```
NIL
```

This was the only reference to the first cons cell which can now be freed. Because only a single incoming pointer to the gate is now marked, it releases its target object.

```
CL-USER 17 > (mark-and-sweep 3)
```

```
"a Test instance was gc'ed"
```

```
18437248
```

```
18702000
```

```
CL-USER 18 > (type-of (car *b*))  
(SIMPLE-VECTOR 5)
```

```
CL-USER 19 > (sys::gate-target (car *b*))  
NIL
```

This allowed us to use the standard garbage collector to do the collecting of generic function methods which could be shaken away if the classes on which they specialised were shaken.

1. <http://www.lispworks.com/>

---

## 2.7.5 Cut! (2008-07-16 18:53)

[1]Show-stopper!: The Breakneck Race to Create Windows NT and the Next Generation at Microsoft

I read this book over the space of a few days. I'd seen it on the desk of a colleague at work and heard it recommended on several podcasts such as [2]Stack Overflow, so I bought it from amazon for something like five pounds. It was an interesting read giving brief coverage to the whole of the NT project via snapshots on the lives of ten or so people who worked on the project. I would have liked more technical details, but it seemed to give a flavour of what a four(?) year project with several hundred people working on it must have been like.

1. [http://www.amazon.co.uk>Show-Stopper-Breakneck-Windows-Generation-Microsoft/dp/0029356717/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1216233825&sr=1-1](http://www.amazon.co.uk>Show-Stopper-Breakneck-Windows-Generation-Microsoft/dp/0029356717/ref=sr_1_1?ie=UTF8&s=books&qid=1216233825&sr=1-1)
2. <http://blog.stackoverflow.com/>

---

## 2.8 August

### 2.8.1 Voyage to the bottom of the sea (2008-08-12 05:56)

A few weeks ago I finished reading [1]C # in depth by Jon Skeet.

This is a magnificent book that details the changes to the C # language as the language evolved into C # 2 and 3. After a very brief history and review of some of the trickier concepts of C # (delegates and value types) it dives into the changes for C # 2 starting with a great explanation of generics and offering some observations that I haven't come across before. It then moves on to an in-depth look at the co- and contra- variance of delegates and how variables are captured in anonymous methods, before covering the easier ways to write iterators using the yield statement. A final chapter on C # 2 briefly covers other changes such as partial classes, static classes, getter/setter property access separation and a couple of other changes.

After this it moves on to C # 3. Lambda expressions and expression trees are covered to a good level of detail, with an explanation of the changes to the type inference process of the compiler that were needed to allow a lot more implicit typing. (This is something I want to look at in a lot more detail). Extension methods are then introduced. After this the book moves on to query expressions and LINQ to objects, detailed coverage of the extension methods that turn IEnumerable streams into things that can be combined in very powerful ways. The final chapter covers LINQ to SQL and the way IQueryable can be used to allow Linq providers to convert expression trees into code that can be executed outside the world of C #. This is the mechanism used to convert a query written in C # into SQL that can be executed inside SQL server.

This book is truly great making loads of interesting observations along the way. I loved the way that it covers the plumbing of Linq in great detail before moving on to the query syntax that make query expressions very approachable in the C # language.

LINQ is great. It brings some of the ideas of lazy streams into a mainstream language. The idea of defining the effect of a Linq expression by defining the language to language transform needed for compiling it reminded me very much of the way macros in Lisp can be used to define advanced language concepts in terms of more basic concepts. There's a good paper by Henry Baker, [2]Metacircular semantics of Common Lisp special forms, that does this for the language of Common Lisp, explaining how many of the control constructs of the language can be defined in terms of other constructs. The original definition of Lisp defined the language by giving a definition of the interpreter written in Lisp. To me this seems an effective means for explaining the semantics of the language to someone who knows a little of the language in the first place. BTW this is the same Henry Baker who devised the first [3]incremental garbage collection algorithm which uses "to" and "from" spaces and a write barrier to get its incremental collections.

1. [http://www.amazon.co.uk/C-Depth-What-master-Master/dp/1933988363/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1218518971&sr=8-1](http://www.amazon.co.uk/C-Depth-What-master-Master/dp/1933988363/ref=sr_1_1?ie=UTF8&s=books&qid=1218518971&sr=8-1)
  2. <http://home.pipeline.com/~hbaker1/MetaCircular.html>
  3. <http://home.pipeline.com/~hbaker1/RealTimeGC.html>
- 

## 2.8.2 Let's share some info (2008-08-14 09:18)

I've just finished re-reading [1]Shared Source CLI Essentials

This is a great book that covers the Rotor release of the .NET 1.1 framework. This release runs on both Linux and Windows and contains source and assembler for many parts of the CLR. The book can't cover the material in detail because of the sheer size of the release but dives into interesting parts of the framework for a look around. I found the coverage of the PAL (the platform adaptation level) very interesting. This is the framework that implements a consistent view of a subset of windows and linux with which the CLR needs to interact; things like threading and exception handling, with coverage given to a portable implementation of structured exception handling.

I was going to blog about how clever the CLR is in optimising the layout of interface tables, only to find that the implementation of calling via an interface has now been optimised to use [2]polymorphic inline caching in the 2.0 release of the CLR.

- 
1. [http://www.amazon.co.uk/Shared-Source-Essentials-David-Stutz/dp/059600351X/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1218705025&sr=1-1](http://www.amazon.co.uk/Shared-Source-Essentials-David-Stutz/dp/059600351X/ref=sr_1_1?ie=UTF8&s=books&qid=1218705025&sr=1-1)
  2. <http://blogs.msdn.com/vancem/archive/2006/03/13/550529.aspx>

### **2.8.3 Help is at hand (2008-08-27 16:25)**

Fortunately, Ted Neward and Joel Pobar are writing a new version of SSCLI internals book aimed at the 2.0 version of the platform. This contains details about the new stub dispatch architecture and the implementation of generics, together with other interesting changes to the distribution.

It is available via [1]this blog entry.

- 
1. <http://callvirt.net/blog/entry.aspx?entryid=b9a94d0c-761a-4d6b-bc2f-d6a5f8c1a4a7>

## **2.9 September**

### **2.9.1 Flash... no..oh. (2008-09-08 18:29)**

[1]ASP.NET Ajax in action

I've been keen on doing some Ajax development for a long time now. Javascript is powerful enough to have lots of computation going on inside the browser (though only on a single thread) with callbacks to the server happening under control of this client code at appropriate points in the interaction sequence, without the GUI going dead while the callback happens. DHTML allows the display to be updated with a minimal amount of distraction (ie flashing) and CSS allows a number of quite sophisticated effects on the page. It's great to find that we are now in a position where a web application can work cross-browser, with the changes required for particular browsers isolated in the library code.

This book is great and can be divided into four parts.

An introduction to the Microsoft Ajax library. This defines a pattern for using javascript to make it feel like a object oriented language in the mode of C#. Classes have instances which support interfaces, properties and events, with a means of interrogating instances and their classes for metadata. In order to do this, the library defines a standard pattern for Function object definitions, which then allow the library to ensure that the initialization happens along an inherited constructor chain. The book then goes into details of using the ScriptManager and UpdatePanels which allow partial page updates without the C# programmer having to do any non-server side code. Lastly the first part covers how server functionality can be exposed to client side script via ASP.NET web services.

Part two of the book goes into detail about what is happening behind the scenes in the implementation of the UpdatePanel. When using the UpdatePanel, there has to be an instance of a ScriptManger on the ASP.NET page. At runtime, this causes an instance of a PageRequestManager object to be constructed on the client. It is this instance that manages the page requests that are passed using asynchronous calls back into the server. The viewstate for the page is threaded through these calls to keep everything consistent, limiting the system to a single active call per page.

The ASP.NET Ajax framework allows the user to encapsulate functionality into controls and behaviours. Behaviours take an existing DOM element and extend it by hooking code into its

events. An example of this would be something like an auto-completion extender for a text box. A DOM element could have multiple pieces of functionality added to it. Alternatively, full controls can be defined that consist of code that runs on both the client and server sides. The library makes the definition of such elements fairly straightforward with a well defined construction protocol and means to pass references to other items on the page into the javascript instance with the \$create function.

The book then gives a brief introduction to the Ajax Control Toolkit which offers a wide range of useful components, from things like simple text box auto-completers to calendar controls to animation controls like FadeAnimation, ScaleAnimation and ConditionAnimation.

Part three of the book covers ASP.NET Ajax Futures. This covered two things: XMLscript and drag and drop. I found XMLScript very interesting. It allows javascript objects, defined with metadata on top of the Ajax library, to be connected via a declarative syntax, so that you could, for example, cause an update to a textbox to trigger an action. This all works by having an inbuilt XML parser that takes the XML and uses it to probe the various objects for their metadata and events - events can be subscribed to and properties can be set. Javascript functions can be composed into actions which may connected into this process.

Part four of the book covers other aspects of Ajax. Support for history and declarative data binding were interesting parts of this section.

This book was good because it showed how the technology could be used by way of some interesting examples, but also went in depth as to how the technology worked, and why it worked that way.

1. [http://www.amazon.co.uk/ASP-NET-AJAX-Action-Alessandro-Gallo/dp/1933988142/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1220892565&sr=8-1](http://www.amazon.co.uk/ASP-NET-AJAX-Action-Alessandro-Gallo/dp/1933988142/ref=sr_1_1?ie=UTF8&s=books&qid=1220892565&sr=8-1)
- 

## 2.9.2 Play it again Sam. (2008-09-08 18:45)

[1]Beginning XNA 2.0 Game Programming

After our experimentation with writing a game for the DS, my son has decided that it would be good to write a game for our XBox. This platform offers a lot more options for writing games compared to the DS which is limited by its less sophisticated display capabilities and the need to write in C. The XNA framework for the Xbox allows the programmer to write code in C # to produce applications that can run on both the Xbox and the PC. The game can be developed inside Visual Studio, which allows fairly good debugging capabilities. The XNA framework supports both 2D sprites and 3D models, and offers the framework of the game loop into which objects can be inserted. Game objects basically support methods for updating their state, when they are passed parameters to say how long it was since the last update, and methods that tell them to render themselves. This makes simple game development much simpler than the C loop of the DS model.

This book gives coverage of the whole XNA framework. It has examples demonstrating the production of both a 2D and a 3D application. The 2D game is a "dodge asteroids" game which is extended from a single player game into a multi-player cross network game. The 3D game is a third person shooter where the player games in a 3D world containing a number of 3D monsters. The introductions to 3D graphics and the rendering and shading pipelines was good, and the examples were well chosen.

I'm looking forward to trying it all out.

1. [http://www.amazon.co.uk/BEGINNING-XNA-2-0-GAME-PROGRAMMING/dp/1590599241/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1220898724&sr=8-1](http://www.amazon.co.uk/BEGINNING-XNA-2-0-GAME-PROGRAMMING/dp/1590599241/ref=sr_1_1?ie=UTF8&s=books&qid=1220898724&sr=8-1)
- 

### 2.9.3 Stand back - I'm a F#'ing scientist (2008-09-08 19:07)

[1]F # for Scientists[2]

I attended the first [3]SQLBits conference for which all participants were awarded a thirty pound book voucher. I used that voucher to purchase this book which has only just come out. I'm supposed to be giving a talk on F # at work next month and have been trying to do lots of F # reading in preparation - this book was certainly worth the wait.

This book is aimed at scientists doing numerical modelling, and I presume is based on the author's previous book which used the O'Caml language for the programming. It offers a number of things. First it is a good introduction to the F # language which it covers from a very pragmatic perspective. It covers why functional languages are good for the type of numerical modelling that scientists need to do - high level algorithmic abstraction via higher order functions, controlled use to state and easy access to useful datatypes like trees can make it very easy to do some kinds of numerical modelling work. The book covers the CLR platform and shows how it can be easily used from F #; DirectX can be used for visualisation, threading libraries can be used for parallelisation, web services can be easily accessed and serialization can be used for storing data. Additionally F # comes with parsing and lexing tools that can be used to process interesting data formats. Interoperability with Excel and other tools is demonstrated.

The book also contains good chapters on algorithmic complexity, pointing out that instead of micro-optimisation it is often better to choose a better data representation and algorithm, and on methods for optimising functional programs.

1. [http://www.amazon.co.uk/F-Scientists-Jonathan-D-Harrop/dp/0470242116/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1220901457&sr=1-1](http://www.amazon.co.uk/F-Scientists-Jonathan-D-Harrop/dp/0470242116/ref=sr_1_1?ie=UTF8&s=books&qid=1220901457&sr=1-1)

2. <http://>
  3. <http://www.sqlbits.com/>
- 

Clive (2008-09-10 16:22:00)

The coverage of DirectX runs to about 20 pages - from 180 to 198 of the book, and isn't particularly deep. It mentions downloading Managed DirectX for .NET, defines a type for a form that allows a pluggable render function, discusses projections in a couple of pages, and then uses the DrawUserPrimitives function to show how tessellations of various shapes can be rendered and how a graph of a function of 2 parameters can be sampled to form a surface. There is nothing like the coverage of the XNA books... it's really showing how F # can plug into the libraries of the managed world.

Tom (2008-09-09 07:09:36)

Wasn't the original line from Ghostbusters 'Back off man, I'm a scientist!' :-) Could you say more about the Direct X and threading coverage since those are the technologies I'm particularly interested in using in conjunction with F #.

#### **2.9.4 It's a bit of an enigma (2008-09-08 19:16)**

[1]Can you Crack the Enigma code?[2]

I bought this in a bookshop in Ambleside when I was on holiday. I was hoping that it was going to be like [3]The Code Book by Simon Singh, but unfortunately it was nothing like it. It contained seven chapters with seven examples of codes. One chapter covered the infamous Beale papers (on which one of the National Treasure films is based), another the mystery behind the references in the Enigma Variations and another the Kryptos sculpture at the CIA headquarters.

It was an easy read and fairly enjoyable, but very little technical depth.

1. [http://www.amazon.co.uk/Can-You-Crack-Enigma-Code/dp/0752881620/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1220901540&sr=1-1](http://www.amazon.co.uk/Can-You-Crack-Enigma-Code/dp/0752881620/ref=sr_1_1?ie=UTF8&s=books&qid=1220901540&sr=1-1)
2. [http://www.amazon.co.uk/Can-You-Crack-Enigma-Code/dp/0752881620/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1220901012&sr=8-1](http://www.amazon.co.uk/Can-You-Crack-Enigma-Code/dp/0752881620/ref=sr_1_1?ie=UTF8&s=books&qid=1220901012&sr=8-1)

3. [http://www.amazon.co.uk/s/ref=nb\\_ss\\_b/203-1268316-5623149?url=search-alias%3Dstripbooks&field-keywords=the+code+book](http://www.amazon.co.uk/s/ref=nb_ss_b/203-1268316-5623149?url=search-alias%3Dstripbooks&field-keywords=the+code+book)

---

## 2.9.5 Listen to this (2008-09-08 20:33)

I thought it was time to put together a list of the podcasts that I regularly listen to.

[1].NET rocks is a twice weekly show covering all .NET technologies. The majority of the podcast is an interview with a guest who is often a Microsoft employee. This podcast has been running for several years now. I think I've listened to every episode.

[2]The java posse is a weekly podcast on java technology, though it often covers other related topics and the hosts' interests such as Scala. It is witty and very informative.

[3]Industry misinterpretations is a smalltalk podcast from Cincom, a major smalltalk vendor. This gives another perspective on the computing industry from the point of view of a niche language.

[4]Software engineering radio, "the podcast for the professional developer". This covers a variety of software engineering topics, from aspects to garbage collection to concurrency to agile methodologies to modelling. Generally interesting.

[5]Audible Ajax and [6]Open web podcast cover Ajax with interviews covering interesting Ajax development topics.

[7]Pixel8 and [8]Hanselminutes cover software development with a .NET focus.

[9]Security now covers general internet security issues with tutorials on subjects such as DNS and PKI.

[10]Stackoverflow has great discussions on software development issues, with Joel Spolsky and Jeff Atwood

[11]Alt.Net podcast covers some of the alternative .NET technologies such [12]NHibernate.

1. <http://www.dotnetrocks.com/>
  2. <http://www.javaposse.com/>
  3. <http://www.cincomsmalltalk.com/blog/blogView?searchCategory=podcast>
  4. <http://www.se-radio.net/>
  5. <http://media.ajaxian.com/>
  6. <http://openwebpodcast.com/>
  7. <http://feeds.feedburner.com/Pixel8>
  8. <http://www.hanselminutes.com/>
  9. <http://www.grc.com/securitynow.htm>
  10. <http://blog.stackoverflow.com/>
  11. <http://altnetpodcast.com/>
  12. <http://www.summerofnhibernate.com/>
- 

## 2.9.6 The good, the bad and the ugly (2008-09-21 16:41)

[1]Javascript: The good parts by Douglas Crockford

This is a fairly quick read at 150 pages and gives a great overview of the good parts of javascript. I think the good parts of this language are really good - prototypal inheritance, closures and higher order functions all within a fairly simple language. Crockford, who did a great series of [2]downloadable talks on javascript covers the material well with an style that is easy to read.

1. [http://www.amazon.co.uk/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1222014873&sr=1-1](http://www.amazon.co.uk/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742/ref=sr_1_1?ie=UTF8&s=books&qid=1222014873&sr=1-1)
  2. <http://video.yahoo.com/watch/111585/1027823>
- 

## 2.9.7 That's a very impressive CV (2008-09-21 16:46)

[1]The Universe: A biography by John Gribbin

A great easy read on cosmology. Fantastic.

1. [http://www.amazon.co.uk/Universe-Biography-John-Gribbin/dp/0141021470/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1222015327&sr=1-1](http://www.amazon.co.uk/Universe-Biography-John-Gribbin/dp/0141021470/ref=sr_1_1?ie=UTF8&s=books&qid=1222015327&sr=1-1)
- 

## 2.10 November

### 2.10.1 It's dark out there (2008-11-29 18:32)

I hate it when blogs go dark.

My excuse is that I've been working hard on a product ever since I started at [1]Red Gate at the end of January. At last the beta is out.

Rob, the tech lead, [2]comments here and Michael interviewed some of the [3]members of the team.

Mum, that's me in the photo (not the Lara Croft one).

- 
1. <http://www.red-gate.com/>
  2. <http://www.simple-talk.com/community/blogs/robertchipperfield/archive/2008/11/12/70423.aspx>
  3. [http://www.simple-talk.com/exchange/exchange-articles/the-road-to-beta---exchange-server-archiver-speaks! /](http://www.simple-talk.com/exchange/exchange-articles/the-road-to-beta---exchange-server-archiver-speaks!/)

---

### 2.10.2 Ponder it? It almost blew my mind (2008-11-29 19:18)

It's great to see that Ponder This has reached its [1]127th month. I love these monthly challenges which I first discovered in [2]November 2002. After many years, it got me interested again in doing Maths puzzles, particularly those related to combinatorics. Over the last couple of years the puzzles have moved more towards probability problems... this is a subject that I didn't enjoy at all as an undergraduate, and in an attempt to improve myself I have been working though [3]50 Challenging Problems in Probability This was a really enjoyable read. It includes such gems as the coupon collection puzzle (if a free gift comes with a breakfast cereal and there are  $n$  to collect, how many boxes will I need to buy), how long on average to draw an ace from a pack of cards, if I lay two packs of cards side-by-side how many matches will I find and the dowry problem (100 pieces of paper with an amount on them drawn one by one - what's the strategy for stopping on the largest) together with many other interesting problems. One other book of such problems is Peter Winkler's [4]Mathematical Puzzles: A Connoisseur's Collection which is also a really good read.

- 
1. [http://domino.research.ibm.com/Comm/wwwr\\_ponder.nsf/Challenges/November2008.html](http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/Challenges/November2008.html)
  2. [http://domino.research.ibm.com/Comm/wwwr\\_ponder.nsf/Challenges/November2002.html](http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/Challenges/November2002.html)
  3. [http://www.amazon.co.uk/Fifty-Challenging-Problems-Probability-Solutions/dp/0486653552/ref=sr\\_1\\_1?ie=UTF8&qid=1227982258&sr=8-1](http://www.amazon.co.uk/Fifty-Challenging-Problems-Probability-Solutions/dp/0486653552/ref=sr_1_1?ie=UTF8&qid=1227982258&sr=8-1)
  4. [http://www.amazon.co.uk/Mathematical-Puzzles-Connoisseurs-Peter-Winkler/dp/1568812019/ref=sr\\_1\\_2?ie=UTF8&s=books&qid=1227985938&sr=1-2](http://www.amazon.co.uk/Mathematical-Puzzles-Connoisseurs-Peter-Winkler/dp/1568812019/ref=sr_1_2?ie=UTF8&s=books&qid=1227985938&sr=1-2)

### **2.10.3 Style only gets you so far (2008-11-29 19:32)**

After recently doing some CSS work as part of my day job [1]my day job, I've got interested again in presentation.

The combination of HTML and CSS with Ajax techniques seems to offer a truly cross platform way to get rich interactive applications onto a range of computers and operating systems without the usual installation problems. CSS is a tidy means of customising the display to get things looking really good. However, the lack of a first class notion of layout keeps me unsatisfied - laying things out using nested divs and spans seems really low level, and resorting to tables only works for a limited set of circumstances. Existing elements can be customised using javascript, but owner drawn items are not portable (though may become so in the future).

For inspiration, I read [2]Pro CSS and HTML Design Patterns which gives a mass of useful examples of desired layouts and looks and how they can be realized in HTML and CSS. Some of the effects are truly impressive.

In the end though, what I want is a nice declarative way of putting together some layouts. More than a decade ago, I did a lot of work on the Harlequin Common Lisp implementation, where several rather smart people had designed a windows toolkit called [3]CAPI (Common Application Interface). This consisted of a means of taking a set of panes and combining them using layouts such as row layouts, column layouts and grid layouts. Elements could be constrained with minimum and maximum sizes, and layouts could have various constraints applied to them, and the system would take care of choosing appropriate sizes to get things displayed. When the top level was resized, the elements would resize appropriately.

It was a shock when I moved to the Windows world, where interfaces were defined by manually laying out elements at fixed pixel offsets relative to a parent, leading to many windows being fixed in size. Things improved a little with Window Forms and the ability to anchor and dock controls within a main window, but still HTML tables behaved far better when the window was resized. However, Windows forms had the advantage of owner draw which meant that users could easily define rich controls, and in later versions layouts too.

More than a year ago, I went to a [4]talk on WPF and it has taken me some time to get around to reading the presenter's book [5]Programming Windows Presentation Foundation. WPF feels like HTML presentation programming, being based on a declarative description of the content, whilst allowing styling with style objects and offering rich support for data binding. XAML is (just) a layer describing how a set of underlying objects should be constructed, so there is the flexibility of simply writing C # code to put together a set of objects representing the logical view of the GUI. The logical layer has a connected visual tree which the system takes care of rendering - no need for handling WM\_PAINT events. Indeed, the painters algorithm is used for painting the controls, so that controls lower down the z order show through transparent regions in higher controls. WPF separates behaviour from content, so that a button is really a behaviour (handling a pressed state and firing an event) which can have virtually anything as its content, such as a grid displaying a number of images.

There is even some possibility of getting rid of code behind using the model-view-view model pattern. There was a recent talk of this pattern at the local NxtGenUg meeting [6]Make Patterns With Patterns with the speaker's demonstration code available [7]here. The presenter took an example that consisted of some XAML with some code behind handling events, and over an hour transformed it in to XAML that used data binding against a view model, in the process getting rid of the code behind. Even event handling can be integrated into this pattern using command objects from the view model which can then be data bound into the view, though the routing of such commands can be [8]hard to understand. MVVM seems to offer a worthwhile prize. The ability to unit test a view model which is very close to what will be finally rendered via data binding and hence with no code to link the model into the display, and it certainly warrants more investigation.

WPF offers 3-D modelling which might be useful for rich data display and there's a cut down version which runs on top of Silverlight making skills in this area transferable to RIA development. There is a new podcast by Josh Holmes [9]devoted to RIA which should get interesting.

Deployment is still the one place where web applications win. There is a lot of detail about ClickOnce in the book, but the initial Clickonce buzz of several years ago seems to have died down.

There was also panel discussion on .NET rocks, where it was mentioned that Visual Studio is being [10]rewritten to use WPF. The associated discussion as to whether this was covered by NDA gave the assertion some credibility.

1. <http://www.red-gate.com/products/Exchange/index.htm>
2. [http://www.amazon.co.uk/Pro-CSS-HTML-Design-Patterns/dp/1590598040/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1227808188&sr=1-1](http://www.amazon.co.uk/Pro-CSS-HTML-Design-Patterns/dp/1590598040/ref=sr_1_1?ie=UTF8&s=books&qid=1227808188&sr=1-1)
3. <http://www.lispworks.com/documentation/lw51/CAPUG-M/html/capiuser-m.htm>
4. <http://www.bcs-spa.org/cgi-bin/view/SPA/WindowsPresentationFoundation>
5. [http://www.amazon.co.uk/Programming-Windows-Presentation-Foundation/dp/0596101139/ref=sr\\_1\\_2?ie=UTF8&s=books&qid=1227807847&sr=1-2](http://www.amazon.co.uk/Programming-Windows-Presentation-Foundation/dp/0596101139/ref=sr_1_2?ie=UTF8&s=books&qid=1227807847&sr=1-2)
6. <http://www.nxtgenug.net/ViewEvent.aspx?EventID=158>
7. <http://vistasquad.co.uk/blogs/nondestructive/archive/2008/11/19/code-available-from-mvvm-talk.aspx>
8. <http://msdn.microsoft.com/en-gb/magazine/cc785480.aspx>
9. <http://www.wiresidechat.com/blogs/ria/Default.aspx>
10. <http://www.dotnetrocks.com/default.aspx?showNum=396>

---

Unknown (2008-12-01 10:29:12)

That's why I love TableLayouts and FlowLayouts so much in WinForms... I find them far less quirky than trying to get things laid out "graphically" in the VS designer when it comes to resizing, and particularly variable DPI settings. That said, since I've started working with WinForms, I've come to appreciate the Java Swing way of doing things a lot more - it seemed clunky when I first learnt it ("Why can't I just drag and drop!?"'), but now I long for something with all the constraints and layout managers it has...

#### **2.10.4 Software East (2008-11-29 19:39)**

I recently attended the first meeting of Mark Dalgarno's [1]Software East

The meeting consisted of two talks.

"Moving from Coding to Model-Driven Development" by Steven Kelly was an overview of the advantages of domain specific modelling when compared with standard software practices. He quoted impressive productivity gains from raising the abstraction level using techniques where the model is converted to code that sits on top of a suitable framework. The idea being that the

company's expert developers write this translation and the framework on which the generated code will sit. Domain experts can then express their knowledge at the level of the model.

One example was for a company with several hundred types of insurance policy. By offering suitable modelling primitives, the non-programming domain experts could express the various policies without the need to get programmers to write the java code, giving a huge gain in terms of flexibility and avoidance of errors.

The second talk, "Get started with Software Product Lines - Key success factors and what to avoid" by Danilo Beuche was an introduction to software product lines, and gave an interesting overview on how to integrate libraries into different products to get a range of product with a range of functionalities.

The meeting was good and there was time before the talks to chat with the speakers. The next meeting [2]is in January on the subject of "starting a software business".

1. <http://www.software-east.co.uk/nov08event.php>
  2. <http://www.software-east.co.uk/>
- 

### **2.10.5 There's only one way I can react to that (2008-11-29 19:42)**

Reactive functional programming is fascinating, and recently two sets of related blog postings have passed through my RSS feed.

TomasP who is writing a book on the subject of functional programming (with examples in C # and F #) has been posting about [1]reactive LINQ. Conal Elliott has been working on a [2]new reactive programming library for Haskell with a tutorial series of posts [3]starting here.

I will definitely be digging deeper into this area in the near future.

1. <http://tomasp.net/blog/reactive-ii-csevents.aspx>
  2. <http://www.haskell.org/haskellwiki/Reactive>
  3. <http://netsuperbrain.com/blog/posts/introducing-reactive-events/>
- 

## 2.11 December

### 2.11.1 F# can even take out the garbage (2008-12-09 10:26)

With all this talk of modelling at the talks I've recently attended, I figured that it might be time to put some of this into practice. F # supports the notion of meta-programming quite nicely; there is a means to get hold of a typed abstract syntax tree for expressions that one defines in the language. In principle therefore, one could write a garbage collector in F #, getting the benefits of high level language debugging of the algorithms used and then one could use the quotation support to code generate a garbage collector in C. This kind of idea has been used before - the [1]Squeak smalltalk virtual machine was debugged in Smalltalk and then the VM itself was code generated as C from this object model. Unfortunately, I haven't yet had time to complete this project. I have finished an implementation of a rudimentary garbage collector, but haven't yet done the interesting code generation part. I figured that I might as well blog about that part and hope to get more time soon to progress further.

The type of garbage collector I am interested in code generating is a mark/sweep collector using pointer reversal. Mark/sweep is one of the very early garbage collection algorithms (from the late 1950's) and without the complication of generations it is fairly easy to implement. Pointer reversal is a technique for storing the stack, which accumulates as we mark through objects, inside the heap objects themselves, avoiding the need for a garbage collector stack or mark map. It is a technique that has fallen out of favour, one reason being that the multiple writes

to objects when we store the reverse pointers means that a lot of pages get touched multiple times, which might cause the virtual memory system some problems.

We'll build heap allocated objects on top of this type. Objects on the heap have a flag to say if they are marked (Marked), can be initialized to safe values (Initialize), can list the pointers that they contain (Fields) and allow us to store a pointer and offset in them (StorePointer) which we'll need when we store the reverse pointers into the objects. Fields takes an offset into the object and returns the next offset and value of any embedded pointer, and returns None if there isn't one. When we store a pointer (StorePointer) into an object at a given offset, we return the old contents of that pointer holding field and old stored offset. By doing this overlaying, the only extra space we need per-object is space for the mark bit and the offset value.

```
type GCable() =
class
abstract Marked : bool with get, set
abstract StorePointer : option<GCable * int> * int -> option<GCable * int>
abstract Initialize : unit -> unit
abstract Fields : int -> option<GCable * int>
end
```

We can now define an IntCell which has a head storing an int and a tail that may contain either another IntCell or None. Class types in F# don't have a null element so we need to use an option type to give us None to represent this null case.

```
type IntCell() =
inherit GCable()
let mutable marked = false
let mutable offset = -1
let mutable head = 0
let mutable tail = None
override me.Marked with get() = marked and set(value) = marked <- value
override me.StorePointer (reverseData,y) =
if y <> 1 then failwith "Invalid offset to store"
```

```

let toReturn =
if offset = -1
then None
else
match tail with
| None -> failwith "Invalid reverse tail"
| Some datum -> Some(datum, offset)
match reverseData with
| Some (x, storedOffset) ->
offset <- storedOffset
tail <- Some x
| None ->
offset <- -1
tail <- None
toReturn
override me.Initialize() =
head <- 0
tail <- None
override me.Fields(index) =
match tail with
| None -> None
| Some value -> if index = 0 then Some (value,1) else None

```

```

member me.head with get() = head and set(x) = head <- x
member me.tail
with get() = (match tail with None -> None | Some y -> Some(y :> IntCell))
and set(x : option<IntCell>) = tail <- (match x with None -> None | Some y -> Some (y :> GCable));

```

Our heap will support the following interface.

```

type IGCHheap =
interface
abstract Allocate : System.Type -> option<GCable>
abstract FreeSpace : int with get
abstract Allocated : unit -> List<GCable>
abstract MakeFree: GCable -> unit
end;;

```

Allocate will try to allocate another object of the given type. It will return None if there is no space left. The heap will track the allocated objects (Allocated) and we have access to a MakeFree function for releasing objects. FreeSpace gives us a measure of the free space available.

```
type TestHeap(size) =  
let allocatedItems = new List<GCable>()  
let freespace() = size - allocatedItems.Count  
interface IGCHeap with  
member me.Allocate(objType) =  
if freespace() = 0  
then None  
else  
let newItem = System.Activator.CreateInstance(objType) :?> GCable  
newItem.Initialize()  
allocatedItems.Add(newItem)  
Some newItem  
member me.FreeSpace with get() = freespace()  
member me.Allocated() =  
let result = new List<GCable>()  
for item in allocatedItems do  
result.Add(item)  
result  
member me.MakeFree (item) =  
item.Marked <- false  
allocatedItems.Remove(item) |> ignore  
end;;
```

We can do some simple tests against the implementation.

```
let Check x = if not x then failwith "Violation"
```

```
let heap1 = new TestHeap(10) :> IGCHeap;;
```

```
Check (heap1.FreeSpace = 10);;
Check (heap1.GCCount = 0);;
```

```
let newCell (heap : IGCHeap) =
  match heap.Allocate(typeof<IntCell>) with
  | None -> failwith "no memory left"
  | Some datum -> datum :?> IntCell;;
```

```
let obj1 = newCell(heap1);;
```

```
Check (heap1.FreeSpace = 9);;
Check (heap1.GCCount = 0);;
```

```
Check (obj1.Fields(0) = None);;
```

```
obj1.head <- 50;;
Check (obj1.Fields(0) = None);;
obj1.tail <- Some obj1;;
Check (obj1.Fields(0) <> None);;
obj1.tail <- None;;
Check (obj1.Fields(0) = None);;
```

Now we can get to implementing a collector that frees items on this heap. This is the mark routine that does the pointer reversing marking.

markThrough is called for each object - it marks the object and then enters the scan function which is responsible for marking through all of the fields of the object.

The routines pass the reverse pointers as reverseData which points to the field in the previous object in the stack - this may be None which is the case for the top level entry into the marker.

```
let rec markThrough reverseData (obj : GCable) =
  obj.Marked <- true
  scan obj 0 reverseData
```

```
and scan obj index reverseData =
  match obj.Fields(index) with
  | None -> unwind reverseData obj
  | Some (target, field) ->
    if target.Marked
    then scan obj (index+1) reverseData
    else
      let result = obj.StorePointer (reverseData, field)
      markThrough (Some(obj, field)) target
```

```
and unwind reverseData obj =
  match reverseData with
  | None -> ()
  | Some (item, offset) ->
    let previous = item.StorePointer (Some (obj,0) , offset)
    scan item offset previous
```

We can generate some cells and give it a quick check.

```
let heap2 = new TestHeap(10) :> IGCHeap;;
let obj2 = newCell heap2;;
let obj3 = newCell heap2;;
```

```
Check(not obj2.Marked);;
Check(not obj3.Marked);;
```

```
obj2.tail <- Some obj3;;
```

```
markThrough None (obj2 :> GCable);;
```

```
Check(obj2.Marked);;
Check(obj3.Marked);;
Check((fun (Some datum) -> LanguagePrimitives.PhysicalEquality datum obj3) obj2.tail);;
Check(obj3.tail = None);;
```

Now we need to sweep up after the mark phase.

```
let sweep (heap :IGCHeap) =
for item in heap.Allocated() do
if item.Marked
then item.Marked <- false
else heap.MakeFree item
```

And we can quickly check that it works to some degree.

```
Check (heap2.FreeSpace = 8);;
sweep heap2;;
Check (heap2.FreeSpace = 8);;
```

```
obj2.tail <- None;;
markThrough None (obj2 :> GCable);;
sweep heap2;;
Check (heap2.FreeSpace = 9);;
```

Pointer reversal is easy to get wrong and the bugs often leave the heap in a very corrupted state. By working at this high level, we can unit test the mark routine itself, by defining dummy object types with interesting pointer maps.

```

type TestCell(fields) =
inherit GCable()
let mutable marked = false
let mutable offset = -1
let mutable stored = None
override me.Marked with get() = marked and set(value) = marked <- value
override me.StorePointer (reverseData,y) =
printfn "Store into %d" y
let toReturn =
if offset = -1
then None
else
match fields.[y-1] with
| None -> failwith "Invalid reverse lookup"
| Some datum -> Some(datum, offset)
match reverseData with
| Some (x, storedOffset) ->
offset <- storedOffset
fields.[y-1] <- Some x
| None ->
fields.[y-1] <- None
offset <- -1
toReturn
override me.Initialize() = ()
override me.Fields(index) =
printfn "Fields %d" index
if index >= Array.length(fields)
then None
else
match fields.[index] with
| None -> me.Fields(index+1)
| Some other -> Some (other, index+1);

```

We can then verify (in this case by eye) that the expected series of actions are carried out.

```

let a = new IntCell() :> GCable;; (a :?> IntCell).head <- 1;;
let b = new IntCell() :> GCable;; (b :?> IntCell).head <- 2;;
let c = new IntCell() :> GCable;; (c :?> IntCell).head <- 3;;
let d = new IntCell() :> GCable;; (d :?> IntCell).head <- 4;;

```

```
let fields = [| Some a;Some b; Some b;Some c; Some b; Some d |];;
let testcell = new TestCell(fields);;
```

```
Check(not a.Marked);;
Check(not b.Marked);;
Check(not c.Marked);;
Check(not d.Marked);;
```

```
markThrough None (testcell :> GCable);;
```

```
Check(a.Marked);;
Check(b.Marked);;
Check(c.Marked);;
Check(d.Marked);;
```

```
Check(fields.[0] = Some a);;
Check(fields.[1] = Some b);;
Check(fields.[2] = Some b);;
Check(fields.[3] = Some c);;
Check(fields.[4] = Some b);;
Check(fields.[5] = Some d);;
```

And now we've prototyped, let's define the real thing. We're going to push the garbage collector implementation into the heap itself. The runtime will need to communicate pointer maps to the collector, so we'll allow the runtime to push and pop collections of roots (PushRoots/PopRoots). In practice these maps would be generated lazily.

```
type IGCHheap =
interface
abstract Allocate : System.Type -> option<GCable>
abstract FreeSpace : int with get
abstract PushRoots: GCable list -> unit
abstract PopRoots : unit -> unit
```

```
abstract Collect : unit -> unit
end;;
```

The heap definition now includes the code for the collector.

```
type TestHeap (size) =
let allocatedItems = new List<GCable>()
let freespace() = size - allocatedItems.Count
let mutable rootStack = []
let allocate (objType : System.Type) =
let newItem = System.Activator.CreateInstance(objType) :?> GCable
newItem.Initialize()
allocatedItems.Add(newItem)
Some newItem
let allocated() =
let result = new List<GCable>()
for item in allocatedItems do
result.Add(item)
result
let makeFree (item : GCable) =
item.Marked <- false
allocatedItems.Remove(item) |> ignore
let collector (this) =
List.iter (fun roots -> (List.iter (fun root -> markThrough None root) roots)) rootStack
for item in allocated() do
if item.Marked
then item.Marked <- false
else makeFree item
interface IGCHeap with
member me.Allocate(objType) =
if freespace() = 0
then
collector me
if freespace() = 0
then None
else allocate(objType)
else allocate(objType)
member me.FreeSpace with get() = freespace()
member me.PushRoots frameRoots = rootStack <- frameRoots :: rootStack
member me.PopRoots () = rootStack <- List.tl rootStack
member me.Collect () = collector me
end;;
```

We can then define a new heap and a utility function for allocating data.

```
let heap3 = new TestHeap(10) :> IGCHep;;
```

```
let newCell () =
  match heap3.Allocate(typeof<IntCell>) with
  | None -> failwith "no memory left"
  | Some datum -> datum :?> IntCell;;
```

And then write some test functions to exercise it.

```
let rec test (level : int) =
  if level > 0
  then
    printfn "On entry %d" heap3.FreeSpace
    let cell1 = newCell()
    try
      heap3.PushRoots [ (cell1 :> GCable) ]
      cell1.head <- 1
      newCell() |> ignore
      newCell() |> ignore
      printfn "Before collect %d" heap3.FreeSpace
      heap3.Collect()
      printfn "After collect %d" heap3.FreeSpace
      test (level - 1)
    finally
      heap3.PopRoots();;
```

Notice how the roots are communicated to the heap by pushing a new contour after a sequence of allocations.

We can test the heap by calling this function

```
heap3.FreeSpace;;  
heap3.Collect();;  
heap3.FreeSpace;;
```

```
test 5;;
```

```
heap3.FreeSpace;;  
heap3.Collect();;  
heap3.FreeSpace;;
```

So now I think we are at the stage where we can try to code generate something, but that's going to have to wait until next time.

1. <http://www.squeak.org/>

---

### 2.11.2 Let's chat about F# (2008-12-09 10:34)

In October I got to do a talk on [1]Why F # matters. This was a great chance for me to dive deeper into some parts of the language. As always, the questions people asked made lots of connections within the material that I hadn't seen before.

1. [http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Public/WhyI\\_F.ppt](http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Public/WhyI_F.ppt)
- 

### **2.11.3 Install, damn you! (2008-12-09 10:52)**

I recently finished reading [1]The Definitive Guide to the Windows Installer and, quite frankly, I'm amazed I've been in the industry this long without knowing much about installation technology. In the old days the installer was something that got written in the last few weeks of the project by some poor soul who got handed a set of artefacts and was told to get on with it. The modern trend seems to be for producing an installer as part of the build process. At my previous company I set up a large number of [2]Cruisecontrol builds. The last step of some of these was to generate an installer for the desktop application. The natural choice for this was to use [3]Wix which uses an XML syntax for declaring what actions an installer should take.

Wix itself is great and makes it very easy to do simple installations. The trouble comes when you want to do something more complicated. Wix is both a processor for generating windows installer tables and a set of custom actions for doing standard things (like installing web sites), and sometimes it is hard to see how these things interact. Hence the book mentioned above. It is fantastic. The author takes a number of scenarios and gets you to use the ORCA tool (parts of the windows installer SDK) to look at the installer tables within the msi to see how the installer puts things together to get the product installed. He takes you through defining custom actions which you need to make transactional so that their effects are undone if the installer rolls back the installation at any point, and various other scenarios. Reading the book made it clear why Wix only allows you to do certain things.

1. [http://www.amazon.co.uk/Definitive-Guide-Windows-Installer/dp/1590592972/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1228819222&sr=1-1](http://www.amazon.co.uk/Definitive-Guide-Windows-Installer/dp/1590592972/ref=sr_1_1?ie=UTF8&s=books&qid=1228819222&sr=1-1)
2. <http://ccnet.thoughtworks.com/>
3. <http://wix.sourceforge.net/>

---

## 2.11.4 Stop bleeding! (2008-12-14 08:32)

In a conversation at work the other day, I used the phrase "marshal-by-bleed" and a couple of people hadn't heard it before. There's a good explanation [1]here - in fact there's tons of interesting CLR implementation information in Chris Brumme's blog.

Marshal-by-bleed is a term for objects that are shared between AppDomains for reasons of efficiency or because the object is fundamental to the operation of the CLR and its identity needs to be preserved. Here's a small example program that creates an AppDomain and runs a thread in this new domain which tries to lock some objects that are still locked in the original.

```
//[LoaderOptimization(LoaderOptimization.MultiDomain)]
static void Main(string[] args)
{
    AppDomain newDomain = AppDomain.CreateDomain("Test Runner");
    Monitor.Enter(typeof(Program));
    Monitor.Enter("fred");
    Monitor.Enter(typeof(object));
    new Thread(delegate() { newDomain.DoCallBack(RunTest); }).Start();
    Console.ReadLine();
}

static void RunTest()
{
    Console.WriteLine("Tests starting");
    if (!Monitor.TryEnter(typeof(Program)))
        Console.WriteLine("The program type is locked");
    if (!Monitor.TryEnter("fred"))
        Console.WriteLine("The string is locked");
    if (!Monitor.TryEnter(typeof(object)))
        Console.WriteLine("object is locked");
    Console.WriteLine("Tests done");
}
```

If you run the example, you see that the string instance and the object type are shared between the AppDomains, but the Program type is loaded afresh into the newly created AppDomain so that the lock on the Program type object has no effect inside this domain.

You can get the Program type shared between the AppDomains by uncommenting the Load-erOptimization line.

1. <http://blogs.msdn.com/cbrumme/archive/2003/06/01/51466.aspx>
- 

### **2.11.5 Let's have a chat (2008-12-15 07:39)**

It was interesting hearing [1]Bran Selic talking about UML on [2]se-radio. For me there were three interesting points that I remember from the interview. He covered the history of UML. He made the great observation that people often focus on the static parts of UML such as class diagrams, whereas often the most useful diagrams are those that show the dynamic interactions of the objects. I remember from studying CORBA services in the past that the interfaces often told one very little until one looked at the interaction diagrams to see the way the methods were used. Third, he pointed out that work was being done on defining the semantics of UML. [3]This should be an interesting read.

1. <http://www.se-radio.net/podcast/2008-11/episode-117-bran-selic-uml>
  2. <http://www.se-radio.net/>
  3. <http://research.cs.queensu.ca/~stl/internal/uml2/>
- 

### **2.11.6 I'll have to check if you are my type (2008-12-15 07:43)**

I was looking through the F # specification the other day with an eye to seeing how the type inferencing works.

I've implemented a number of compilers for functional programming languages in the past, and have reasonable knowledge about how they are implemented. To my surprise, the type interfencing is defined in terms of constraint solving. I remembered that there were a number of papers that had passed by on [1]Lambda the Ultimate which reformulated the Milner-Hindley method in terms of constraints, so I dug out the first paper to get a feel for the area.

## [2]Generalizing Hindley-Milner Type Inference Algorithms

The paper covers type inference for a small language with a few base types (int/bool/string) and a very simple grammar containing just variable values, applications, lambda expressions and let bindings. Types consists of type variables, bases types or functions from types to types. There is also the notion of a type scheme which is the generalisation that happens in let bindings. For example, in the code

let id = fn x -> x in (id 1, id true)  
the function bound to id is given the type  
forall a. a -> a

The values in the tuple expression are then type checked using instances of this type scheme, allowing the two uses of id to have different type instantiations.

The way I have seen this implemented in the past is by variants of the W algorithm. This algorithm uses unification to resolve two types to get a most general type of which the two types are an instance.

Let's look at the expression

```
let id = fn x -> let y = x in y  
in id id
```

Algorithm W takes a type environment (mapping program variables to types) and an expression and returns a substitution (for type variables) and a type for the expression. It would type check our example expression as follows:

-> In empty type environment let id = fn x -> let y = x in y in id id  
--> In empty type environment fn x -> let y = x in y

--> We generate a fresh type variable representing the type of x  
 --> In type environment x:b1, let y = x in y  
 --> In type environment x:b1, x  
 --< Return no substitution and type b1 for the expression  
 --> We generalize the result type, b1, for x, in an environment containing x. This generalization doesn't effect  
 --> b1 in this case as this type is bound further out.  
 --> In the type environment x:b1, y:b1 we process y  
 --< Return no substitution and type b1  
 --< We return no substitution and the type b1  
 --< We return no substitution and type b1 -> b1  
 -< We now know that id has type b1 -> b1. Because this is a let binding though, we generalise this  
 -< to forall b1. b1 -> b1. Hence uses of id inside the let body are going to allow separate instantiations.  
 --> In type environment id: forall b1. b1 -> b1 process id id  
 --> In type environment forall b1. b1-> b1 process id  
 --< We instantiate the schema with new type variables and return no substitution and b2 -> b2  
 --> In type environment forall b1. b1-> b1 process id  
 --< We instantiate the schema with new type variables and return no substitution and b3 -> b3  
 --> The applications causes a new type variable b4 to be generated for the result type of (id id)  
 --> and we need to unify (b2-> b2) and (b3 -> b3) -> b4  
 --> This gives substitutions b2 = b3 -> b3 and b4 = b3 -> b3 and the result type is b4, which is transformed  
 --> to b3 -> b3  
 -> The final type of the expression is b3 -> b3.

The constraint approach works very differently; it happens in two phases. First a set of constraints is generated. Then the constraints are solved; it is at this point that the unification happens. Breaking things into two phases offers the possibility for generating much better type error messages.

The constraints take three forms.

$A = B$  reflects that A and B should be unified at a later stage of the type inferencing.

$A <= B$  reflects that A should be a generic instance of B.

To handle the let bound polymorphism, there is a third constraint type,

$A < \{M\} = B$  which reflects that the variables M should remain monomorphic during this unification.

Let's run through the same process on let id = fn x -> let y = x in y in id id

```

-> let id = fn x -> let y = x in y in id id
--> fn x -> let y = x in y in id id
----> let y = x in y
-----> x
-----< Generate a new type variable, b1, and return the environment x:b1 and type b1
-----> y
-----< Generate a new type variable, b2, and return the environment y:b2 and type b2
-----< Return the environment x:b2 and generate the constraints b2 < {b1 }= b1
--< Introduce a fresh type variable b3 for the result type
--< Generate constraints b2=b3
--< and return type b3->b2 and an empty environment
-> Continue the body of the let
--> Process (id id)
--> Process id
-----< Return a fresh type variable b5 and environment id:b5
--> Process id
-----< Return a fresh type variable b6 and environment id:b6
--< Add constraints b5 = b6 -> b7 for a fresh b7, and result type b7 in type environment
id:b5 id:b6
-< Result type is b7 and constraints are b5 < { }= b3 -> b2 and b6 < { }= b3 -> b2

```

Now running the rules to solve the equations gives the same result, forall a. a -> a.  
 $b2 < \{b1\} = b1$  forces  $b2=b1$  and then the key is that  $b5 < \{ \} = b3 \rightarrow b2$  can only be solved after the constraint  $b2=b3$  has been processed, showing that  $b5 < \{ \} = b1 \rightarrow b1$  ie that  $b5$  has the polymorphic type  
forall  $b1$ .  $b1 \rightarrow b1$ .

The constraint framework is neat in that we can see how the types flow while the result is constrained, potentially allowing better debugging of the result. Now to investigate the F # type inferencing in more detail.

1. <http://lambda-the-ultimate.org/>
  2. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.9348>
-



# 2009

## 3.1 January

### 3.1.1 F# I'll sieve my way through that composition (2009-01-04 11:28)

[1]Lambda The Ultimate seems to be a never ending source of interesting papers on computer science. A recent paper that went by [2]The Genuine Sieve of Eratosthenes contained a rather neat implementation of the sieve, showing how it can be implemented very efficiently in Haskell.

F # contains a library for working with lazy lists, so I thought it would be fun to implement a variant of the algorithm from the paper. The paper aims to implement the algorithm as it is execute by hand. You take a list of the numbers from 2 onwards. Pick the first uncrossed number on the list. Note that this is a prime and then cross out all of its multiples from the list. When you wish to cross off multiples of p, you may start at  $p^2$  as all of the lower mutliples of p will already have been crossed off. Repeat.

In a non-functional language, one usually implements the list as a boolean array so that crossing off numbers is implemented by setting elements of this array to false. The algorithm of the paper does this by maintaining a priority queue containing the next multiples of the primes that we have seen so far. When we are considering a particular number we compare it with the least value in this queue; if it is smaller then we have found a prime so we set up the queue with multiples of this new prime. If the number we are considering is equal to the least value then we have a composite so we can ignore this candidate. We then need to adjust the priority queue by removing the least element and placing the next multiple into the queue. The clever idea is that we can implement the multiples as lazy lists which we store in the queue - once we pass the value at the head of the list, we simply replace the list with its tail, in effect, lazily calculating the next multiple.

First, we define the priority queue using a leftist heap. This data structure comes from a book by Chris Okasaki, which I'll blog about at a later stage. Each node has a rank which is the length of right spine and the heap maintains the invariant that for every node the left child rank is at least as large as the right child rank.

```
open LazyList
```

```
type 'a LeftHeap = Empty | Tree of int * 'a * 'a LeftHeap * 'a LeftHeap;;
```

```
let rank tree =
  match tree with
  | Empty -> 0
  | Tree(r, _, _, _) -> r
```

```
let makeTree (x,a,b) =
  if rank a >= rank b
  then Tree(1 + rank b, x, a, b)
  else Tree(1 + rank a, x, b, a)
```

```
let rec merge a b =
  match (a,b) with
  | (Empty, _) -> b
  | (_, Empty) -> a
  | (Tree( _,x,a1,b1), Tree( _,y,a2,b2)) ->
```

```
if hd x <= hd y  
then makeTree(x, a1, merge b1 b)  
else makeTree(y, a2, merge a b2)
```

```
let findMin tree =  
match tree with  
| Empty -> failwith "empty tree"  
| Tree ( _,x,a,b) -> x
```

```
let deleteMin tree =  
match tree with  
| Empty -> failwith "empty tree"  
| Tree ( _, _,a,b) -> merge a b
```

```
let insert x tree = merge (Tree(1, x,Empty, Empty)) tree
```

We can quickly test that out by building a small heap. Note that our heap contains lazy lists at the nodes and is ordered by the value at the head of the list. These lists will be the multiple lists from the algorithm ie when we find 7 is prime, we add the lazy list  $7*7, 8*7, 9*7, 10*7, \dots$  to the queue.

```
let test =  
insert (LazyList.of _list [1])  
(insert (LazyList.of _list[3])  
(insert (LazyList.of _list[2])  
(insert (LazyList.of _list[4]) Empty)))
```

```

> findMin test;;
val it : int LazyList = seq [1]
> findMin (deleteMin test);;
val it : int LazyList = seq [2]
> findMin (deleteMin (deleteMin test));;
val it : int LazyList = seq [3]

```

Now we define the list of numbers from 2 upwards. Be aware that the int type is the 32 bit non-overflowing type so this list of numbers would circle back to 0 via the negative numbers if we pull too many elements off it. F# has better integer types that do not have this behaviour.

```

let twoUp =
let rec generate x = consf x (fun () -> generate (x+1))
generate 2

```

```

> to_list (take 20 twoUp);;
val it : int list
= [2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17; 18; 19; 20; 21]

```

We'll implement the algorithm via the function step. This takes as arguments the candidates list, as the parameter numbers, and the heap of multiples for the primes that we know about so far. It returns a tuple of the first prime in the candidates list, the remaining candidates, and the new multiples heap.

```

let rec step numbers heap =
let nextNumber = hd numbers
if heap = Empty or nextNumber < hd (findMin heap)
then
let newHeap = insert (map (fun x -> x * nextNumber) numbers) heap

```

```
(nextNumber, tl numbers, newHeap)
else
let newHeap = insert (tl (findMin heap)) (deleteMin heap)
let numbers' = if nextNumber = hd (findMin heap) then tl numbers else numbers
step numbers' newHeap
```

```
> let (prime, numbers, heap) = step twoUp Empty;;
```

```
> (prime, numbers, heap);
val it : int * int LazyList * int LazyList LeftHeap
= (2, seq [3; 4; 5; 6; ...], Tree (1,seq [4; 6; 8; 10; ...],Empty,Empty))
```

```
> let (prime, numbers, heap) = step numbers heap;;
```

```
> (prime, numbers, heap);
val it : int * int LazyList * int LazyList LeftHeap
= (3, seq [4; 5; 6; 7; ...],
Tree
(1,seq [4; 6; 8; 10; ...],Tree (1,seq [9; 12; 15; 18; ...],Empty,Empty),
Empty))
```

```
> let (prime, numbers, heap) = step numbers heap;;
```

```
> (prime, numbers, heap);
val it : int * int LazyList * int LazyList LeftHeap
= (5, seq [6; 7; 8; 9; ...],
Tree
(2,seq [6; 8; 10; 12; ...],Tree (1,seq [9; 12; 15; 18; ...],Empty,Empty),
Tree (1,seq [25; 30; 35; 40; ...],Empty,Empty)))
```

That looks good, so let's change the output to be another LazyList.

```
let rec step numbers heap =
let nextNumber = hd numbers
if heap = Empty or nextNumber < hd (findMin heap)
then
let newHeap = insert (map (fun x -> x * nextNumber) numbers) heap
consf nextNumber (fun () -> step (tl numbers) newHeap)
else
let newHeap = insert (tl (findMin heap)) (deleteMin heap)
let numbers' = if nextNumber = hd (findMin heap) then tl numbers else numbers
step numbers' newHeap
```

We can now define a LazyList of the primes.

```
let primes = step twoUp Empty
```

We can use functions like take and drop to get subsequences of this list.

```
> to _list (take 100 primes);
val it : int list
= [2; 3; 5; 7; 11; 13; 17; 19; 23; 29; 31; 37; 41; 43; 47; 53; 59; 61; 67; 71;
73; 79; 83; 89; 97; 101; 103; 107; 109; 113; 127; 131; 137; 139; 149; 151;
157; 163; 167; 173; 179; 181; 191; 193; 197; 199; 211; 223; 227; 229; 233;
239; 241; 251; 257; 263; 269; 271; 277; 281; 283; 293; 307; 311; 313; 317;
331; 337; 347; 349; 353; 359; 367; 373; 379; 383; 389; 397; 401; 409; 419;
421; 431; 433; 439; 443; 449; 457; 461; 463; 467; 479; 487; 491; 499; 503;
509; 521; 523; 541]
```

I love the way that the lazy lists are used in this algorithm with the lists of multiples being generated using a lazy map across the candidates at the time that the next prime is fetched.

- 
1. <http://lambda-the-ultimate.org/>
  2. <http://lambda-the-ultimate.org/node/3127>

### **3.1.2 That'll get a reaction (2009-01-04 11:45)**

I've spent some time over Christmas doing a lot more reading about reactive functional programming. I've always found the idea of representing input, such as keyboard events, as a lazy stream of incoming values fascinating. I first read about it in 1985 when the Oxford Programming Research Group was looking at writing an operating system using LispKit lisp, a lazy lisp dialect, though it seemed to require a notion of non-deterministic merge in order to allow the equivalent of a WaitAny or select. [1]Conal Elliott's blog and the [2]Reactive library on which he is working contain some fascinating insights into this world. He has a large collection of interesting [3]papers.

Some of the good ones, out of those I have read so far, include:

[4]Declarative Event-Oriented Programming which shows the motivation for reactive functional programming, in particular the composability of the definitions.

[5]Functional Reactive Animation documents the implementation of Fran, a library he implemented with Paul Hudak

[6]Simply efficient functional reactivity which is the new formalisation of functional reactivity that underlies Reactive.

1. <http://conal.net/blog/>
  2. <http://haskell.org/haskellwiki/Reactive>
  3. <http://conal.net/papers/>
  4. <http://conal.net/papers/ppdp00/>
  5. <http://conal.net/papers/icfp97/>
  6. <http://conal.net/papers/simply-reactive/>
- 

### **3.1.3 Great functional datastructures 1 (2009-01-04 11:52)**

[1]The Zipper together with some [2]related material on the Haskell Wiki.

1. <http://www.st.cs.uni-saarland.de/edu/seminare/2005/advanced-fp/docs/huet-zipper.pdf>
  2. <http://www.haskell.org/haskellwiki/Zipper>
- 

### **3.1.4 Task finished (2009-01-04 12:44)**

[1]Concurrent Programming on Windows by Joe Duffy

I've just finished reading this book, and at 950 pages of text it has taken quite some time. I've been a fan of [2]Joe Duffy's blog for a long while and couldn't wait for this book to come out. It is, quite simply, superb, going into good technical depth about concurrency as it relates to both the Win32 and the CLR platforms. It seems to cover everything, from the ways to manage program state, to the platform available threads, synchronisation objects, mutexes, condition variables and thread pools. From the asynchronous programming model to fibers. From the memory model to concurrency hazards. From performance and scalability to parallel containers and data and task parallelism. It also has chapters on the parallel extensions for .NET and a chapter on designing reusable libraries for .NET programs.

I'm really keen to get into the low level details of the CLR. In the past, one of the really good blogs was that of [3]Chris Brumme, which gave good information about why features were implemented in a particular way. Joe Duffy's book does the same, linking the Win32 feature and its CLR abstraction with details about how the abstraction is implemented - for example, he covers how the CLR Wait mechanisms need to do lots of work to deal with incoming APCs which cause the wait to terminate at the Win32 level, requiring a rescheduling of the wait at the CLR level. I enjoyed the whole book, but for me the most interesting chapter was the one covering the .NET memory model. This covers in detail the guarantees regarding the reordering of reads and writes that various compilers and jitters realise - in particular, stating the kinds of fences that the system places around the various synchronisation primitives and the meaning of volatile. Many blogs have debated the validity of the double-check lock pattern; this book offers definitive answers.

This book really pulls concurrency into the 21st century. I remember spending time on my degree course understanding algorithms like those of Dekker which implements critical sections using only atomicity of reads and writes. Early in the book, Duffy explains that the reasoning behind these algorithms is invalid on modern hardware where loads and stores may be reordered and memory caches may contain old values which will be refreshed for some time. It is only by having a memory model at both the compiler and hardware level that we can easily reason about such lock free programs. The book points out that lock free programming is very hard to get right - most of the time people should be using lock which implements a full fence at the memory model level. The book covers the Task Parallel Library which offers higher level constructs for getting tasks to run in parallel and easy mechanisms for synchronising the tasks when they finish; this library also offers easy methods for cancelling tasks that are running using cooperative termination.

Appendix A also offers some guidelines on designing concurrent reusable libraries for .NET and contains some really good insights.

This is a book that I am going to reread many times.

1. [http://www.amazon.co.uk/Concurrent-Programming-Windows-Architecture-Development/dp/032143482X/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1231070044&sr=1-1](http://www.amazon.co.uk/Concurrent-Programming-Windows-Architecture-Development/dp/032143482X/ref=sr_1_1?ie=UTF8&s=books&qid=1231070044&sr=1-1)

2. <http://www.bluebytesoftware.com/blog/Default.aspx>
  3. <http://blogs.msdn.com/cbrumme/>
- 

### **3.1.5 Let's catch up (2009-01-04 12:52)**

There's a good channel9 interview with Jeff Richter where he covers [1]asynchronous programming using C # iterators - a neat way to get tasks to wait for the completion of asynchronous operations while still allowing error handling using try...catch...finally...

1. <http://channel9.msdn.com/posts/Charles/Jeffrey-Richter-and-his-AsyncEnumerator/>
- 

### **3.1.6 Keep it secret! (2009-01-04 13:00)**

[1]Secrets of the Rock Star Programmers by Ed Burns

I bought the book for a quick read over Christmas. It consists of a series of 14 interviews of various "Rock Star" programmers, many of whom are members of the java community including the cast of the [2]The Java Posse. It was sort of interesting... many of the interview questions generated answers that could have been a lot more detailed. It was interesting to see the different perspectives, and how different the answers were to the same questions.

1. [http://www.amazon.co.uk/Secrets-Rock-Star-Programmers-Riding/dp/0071490833/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1231073231&sr=1-1](http://www.amazon.co.uk/Secrets-Rock-Star-Programmers-Riding/dp/0071490833/ref=sr_1_1?ie=UTF8&s=books&qid=1231073231&sr=1-1)
  2. <http://www.javaposse.com/>
-

### 3.1.7 Faster... faster... (2009-01-21 22:07)

I've just finished reading Chris Okasaki's [1]Purely Functional Data Structures, or rather I've just finished re-reading chunks of it in an attempt to grasp the material. I must say it was a really good read. As the back cover states, most books on data structures assume an imperative language like C or C++. Some of those data structures do not translate well into languages like ML and Haskell. This book describes data structures from the point of view of functional languages, and presents design techniques so that programmers can develop their own functional data structures.

The first section of the book shows how some common data structures can be implemented in ML. The examples include leftist heaps, binomial heaps and red-black trees. For the latter data type, the re-balancing algorithm looks very tidy when expressed using pattern matching when compared to the more imperative implementations I have seen before.

The second part of the book looks at the relationship between lazy evaluation and amortization. This part starts with a quick introduction to streams (lazy lists) supporting a lazy cons operation with evaluation being forced by pattern matching. Later chapters use lazily evaluated data types to good effect.

Chapter five introduces the idea of using amortized cost instead of worst case cost. Often when we use a data structure we are more worried about the costs of calling a particular operation multiple times rather than the worst case cost of a single call. For some of the common data structures, the worst case bounds are far worse than these amortized bounds. Two techniques are introduced for deriving amortized bounds - the banker's method and the physicist's method. Two data structures are then analysed - the standard functional queue and splay trees, a tree that gets slightly re-balanced on each operation.

We'll look at the queue implementation as it develops during the course of the book. Here's the simple version, implemented in F#, where we use two lists with the invariant that the first is empty only if the second is empty too.

```
let empty = ([],[])
```

```
let isEmpty (f,r) = f = []
```

```
let checkf queue =
  match queue with
  | ([] ,r) -> (List.rev r, [])
  | _ -> queue;;
```

```
let snoc el queue =
  match queue with
  | (f,r) -> checkf (f, el::r)
```

```
let head queue =
  match queue with
  | ([] ,_) -> failwith "Empty"
  | (x::f,r) -> x
```

```
let tail queue =
  match queue with
  | ([] ,_) -> failwith "Empty"
  | (x::f,r) -> checkf (f,r)
```

Standard disclaimer: we should be hiding the implementation details behind a type. By not hiding it, we get a better idea of how the data structure is evolving.

We can exercise the implementation:

```
> snoc 1 empty;;
val it : int list * int list = ([1], [])
> snoc 2 it;;
val it : int list * int list = ([1], [2])
> snoc 3 it;;
val it : int list * int list = ([1], [3; 2])
> snoc 4 it;;
val it : int list * int list = ([1], [4; 3; 2])
> tail it;;
val it : int list * int list = ([2; 3; 4], [])
> snoc 5 it;;
val it : int list * int list = ([2; 3; 4], [5])
> head it;;
val it : int = 2
```

snoc and head run in O(1) worst-case time and tail is O(n) worst-case. However, using the methods from the book, we can show that tail is O(1) amortized time, basically because the reverse of the rear list does work that can be allocated into subsequent calls of tail; if the reverse does a lot of work, then we make it cheap for a lot of subsequent calls to tail, and this balances out over a large series of calls.

Chapter five ends with the observation that this accounting only works if the data structure is used ephemerally ie in a single threaded fashion. If we take the queue in state ([1], [4; 3; 2]) and call tail on this particular instance, then multiple calls to tail run in O( $n^2$ ). The trouble is that the expensive reverse operation is no longer balanced by a subsequent series of cheap calls to tail.

Chapter six introduces the "fix" for this. The fix is to introduce laziness into the language. For our example that means that we do not want to do too much work associated with the reverse until we know that the element is going to be needed, and once we have done the reverse we can share it across a number of persistent (non-ephemeral calls). The banker's method and

the physicist's method need to be extended to deal with laziness to account for this delayed work.

We can reimplement the queue using a suspension for the delayed part of the computation. This suspension introduces highly controlled access to state within the functional language allowing multiple computations to refer to a value that is computed on demand and for which the result can be shared after it has been calculated only once.

We can reimplement the queue using a suspension for the delayed part of the computation.

```
type SuspensionCell<'a> = Evaluated of 'a | Unevaluated of (unit -> 'a);;
```

```
type Suspension<'a> = Empty | Cell of SuspensionCell<'a list> ref;;
```

```
let delay x = Cell(ref(Unevaluated x));;
```

```
let force susp =
  match susp with
  | Empty -> []
  | Cell value ->
    match !value with
    | Evaluated x -> x
    | Unevaluated x ->
      let result = x()
      value := Evaluated result
      result;;
```

This gives us the call-by-need semantics... well almost. As we don't do any locking around the evaluation and reassignment of the memoised value into the reference cell, in principle the computation could be executed multiple times.

We can try it and see that the computation is happening at most once.

```
> xxx;;
val it : Suspension<int> = Cell {contents = Unevaluated <fun:xxx@19>; }
> force xxx;;
val it : int list = [1; 2; 3]
> xxx;;
val it : Suspension<int> = Cell {contents = Evaluated [1; 2; 3]; }
```

The queue can now be rewritten to use the lazy evaluation.

```
let empty () = ([], 0, Empty, 0, []);;
```

```
let isEmpty queue =
let ( _, lenf, _, _, _ ) = queue
lenf = 0;;
```

```
let checkw queue =
  match queue with
  | ([]), lenf, f, lenr, r) -> (force f, lenf, f, lenr, r)
  | _ -> queue
```

```
let check queue =
  let (w, lenf, f, lenr, r) = queue
  if lenr <= lenf
  then checkw queue
  else
    let f' = force f
    checkw (f', lenf+lenr, delay(fun () -> f' @ List.rev r), 0, []);;
```

```
let snoc el queue =
  let (w, lenf, f, lenr, r) = queue
  check (w, lenf, f, lenr+1, el::r);;
```

```
let head queue =
  let (w, lenf, f, lenr, r) = queue
  if w = []
  then failwith "Empty"
  else List.hd w;;
```

```
let tail queue =
  let (w, lenf, f, lenr, r) = queue
  if w = []
  then failwith "Empty"
  else check (List.tl w, lenf-1, delay(fun () -> List.tl (force f)), lenr, r);;
```

We can exercise it.

```
> let el2 = snoc 1 (empty());;
```

```
val el2 : int list * int * Suspension<int> * int * int list
```

```
> snoc 2 el2;;
```

```
val it : int list * int * Suspension<int> * int * int list
= ([1], 1, Cell {contents = Evaluated [1]; }, 1, [2])
```

```
> snoc 3 it;;
```

```
val it : int list * int * Suspension<int> * int * int list
= ([1], 3, Cell {contents = Unevaluated <fun:check@35>; }, 0, [])
```

```
> snoc 4 it;;
```

```
val it : int list * int * Suspension<int> * int * int list
= ([1], 3, Cell {contents = Unevaluated <fun:check@35>; }, 1, [4])
```

```
> tail it;;
```

```
val it : int list * int * Suspension<int> * int * int list
= ([2; 3], 2, Cell {contents = Evaluated [2; 3]; }, 1, [4])
```

```
> tail it;;
```

```
val it : int list * int * Suspension<int> * int * int list
= ([3], 1, Cell {contents = Unevaluated <fun:tail@51>; }, 1, [4])
```

```
> tail it;;
```

```
val it : int list * int * Suspension<int> * int * int list
= ([4], 1, Cell {contents = Evaluated [4]; }, 0, [])
```

```
> tail it;;
```

```
val it : int list * int * Suspension<int> * int * int list
= ([], 0, Cell {contents = Evaluated []; }, 0, [])
```

By doing this we have extended the bounds for ephemeral use to apply to the persistent case too.

The next chapter deals with scheduling. The idea is to get better worst case behaviour by scheduling the suspended computations so that we evaluate a few of them on every operation instead of forcing a large computation in certain circumstances. We can do this by using streams. F# gives us a stream type in the form of LazyList.

```
let xxx =
LazyList.delayed
(fun () -> printfn "called" ; LazyList.consf 10 (fun _ -> printfn "called2" ; LazyList.empty()));;
```

When we evaluate the above, neither of the messages is printed. When we first call the hd function, "called" is printed.

```
> LazyList.hd xxx;;
called
val it : int = 10
> LazyList.hd xxx;;
val it : int = 10
```

When we call the tail function, the second part of the list is evaluated to print the value.

```
> LazyList.tl xxx;;
called2
val it : int LazyList = seq []
> LazyList.tl xxx;;
val it : int LazyList = seq []
```

We can rewrite our ongoing queue example using a stream for the head list, a standard list for the reverse list, and a third stream for computations that are being scheduled.

```
let empty() = (LazyList.empty(), [], LazyList.empty());;
```

```
let isEmpty queue =
let (f,r,s) = queue
not (LazyList.nonempty f);;
```

```
let rec rotate (queue1, list1, queue2) =
if LazyList.nonempty queue1
then
let delayed2 = fun () -> LazyList.cons (List.hd list1) queue2
let delayed = fun () -> LazyList.cons (LazyList.hd queue1) (rotate (LazyList.tl queue1, List.tl
list1, LazyList.delayed delayed2))
LazyList.delayed delayed
else LazyList.delayed (fun () -> LazyList.cons (List.hd list1) queue2);;
```

```
let exec (f,r,schedule) =
if LazyList.nonempty schedule
then (f,r,LazyList.tl schedule)
else
let f' = rotate (f,r,LazyList.empty())
(f', [], f');;
```

```
let snoc el queue =
let (f,r,s) = queue
exec (f, el::r, s);;
```

```
let head (queue,r,s) =
if LazyList.nonempty queue
```

```
then LazyList.hd queue  
else failwith "Empty";;
```

```
let tail (queue,r,s) =  
if LazyList.nonempty queue  
then exec (LazyList.tl queue, r,s)  
else failwith "Empty";;
```

We could exercise it as follows. We need to be careful not to get LazyLists printed by the REPL (read-eval-print-loop) as this will force the computations.

```
> let q3 = snoc 5 (snoc 4 (tail (snoc 3 (snoc 2 (snoc 1 (empty()))))))
```

```
- let h1 = head q3  
- let q4 = tail q3  
- let h2 = head q4  
- let q5 = tail q4  
- let h3 = head q5  
- let q6 = tail q5  
- ;;
```

```
> (h1, h2, h3, q6);;  
val it : int * int * int * (int LazyList * int list * int LazyList)  
= (2, 3, 4, (seq [5], [], seq [5]))
```

This gives us a queue for which all operations are O(1) worst case time; these are known as real-time queues.

The third section of the book looks at some general techniques for designing functional data structures.

The first technique is lazy rebuilding. Here the data structure is rebalanced after a batch of operations instead of incrementally after each operation.

The second is numerical representations. By using redundant bases, we can implement types such as random access binary lists.

The third is data-structural bootstrapping. Types such as tries are covered here. One example of a trie is that of implementing a dictionary lookup by storing the values in the nodes of a tree with the edges being the unique subsequences. For example, we could store the keys "cat", "dog", "car" and "cart" with values v1, v2, v3 and v4 as the trie.

```
null --ca--> null --r--> v3 --t--> v4  
      --t--> v1  
      --dog--> v2
```

The text notes that in some circumstances this type of data structure can be faster than a hash table, as not all elements of the string need to be processed during a lookup. For example, looking up the string "no" could cause a miss after looking at the initial "n".

The final method is implicit recursive slowdown.

I liked the book because it demonstrates that one can implement some efficient data structures in functional languages. I liked the way that call-by-need (call-by-name plus memoisation) allowed persistent versions of the data structures to be implemented efficiently without introducing uncontrolled update of state into the implementation. With the call-by-need, the mutable state is hidden neatly under the covers, though updates to this state could potentially require locking to ensure that calculations were only carried out once.

1. [http://www.amazon.co.uk/Purely-Functional-Structures-Chris-Okasaki/dp/0521663504/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1232286426&sr=8-1](http://www.amazon.co.uk/Purely-Functional-Structures-Chris-Okasaki/dp/0521663504/ref=sr_1_1?ie=UTF8&s=books&qid=1232286426&sr=8-1)

---

### **3.1.8 Even if you're lazy, you can do a lot at the same time (2009-01-27 21:16)**

With F # having re-ignited my interest in functional languages and functional programming, I have been re-reading my way through the series of functional pearls. I'd just read Koen Claessen's [1]A poor man's concurrency monad when the following phd thesis [2]Programmable Concurrency In a Pure and Lazy Language by Peng Li went past on [3]lambda the ultimate. The functional pearl is nice. Its idea is that we break concurrent processes into a series of continuations; these continuations can then be executed by a scheduler which can interleave them so that the processes appear to run in parallel in a cooperative tasking fashion - I tried to cover this kind of simulated concurrency in [4]this blog post when I showed how a subset of Lisp could be converted into continuations. The functional pearl uses monads and monad transformers to hide the abstractions in a very tidy fashion.

The author of the Phd thesis was interested in writing a highly concurrent network services. His first attempt was to use the monad approach to experiment with high degrees of concurrency. However, he rejects this approach for several reasons. First, the monad imposes a lot of structure on the design of the program and the continuations introduce a lot of closures on to the heap - this extra garbage can cause problems on a multiprocessor machine where a single-threaded garbage collection algorithm can impact performance. Secondly, for massively parallel algorithms, it would be nice to have more control over the runtime system - in the existing Glasgow Haskell compiler, the runtime has an abstraction of Haskell threads which are then mapped to operating system threads.

The bulk of the thesis details the implementation of some basic concurrency primitives which the author added to the GHC runtime. A concurrency library, for example a scheduler, can then be written in Haskell using these primitives. The main abstraction is that of a HEC, a Haskell execution context. The currently running HEC can be cooperatively switched to another, in which case the state of the current computation is saved in the former HEC and a new computation is loaded from the target HEC. The runtime system can also elect to handle a number of callbacks. Two such callbacks that we might typically want to handle are regular timer interrupts which can be used to implement preemption, and a black hole handler. When a Haskell program runs, the lazy evaluation is achieved by having parts of the computation represented by thunks - a thunk is executed by the first computation that needs the result. Owing to the sharing of structure, after evaluation of a thunk has started, another computation may come along which also needs the result of the thunk. A black hole is a thunk that is already being evaluated. When the system finds that someone else is computing the value, the handler can be called which can decide to yield the current HEC until the evaluation is finished. The substrate also supports primitive transactional memory (PTM) allowing atomic tests for sleeping and wakeup without the need for (language level) locking. This primitive transaction memory simply allows a series of reads and writes during the course of the atomic action.

The idea of implementing a small amount of code in the runtime and then defining the rest of the concurrency implementation in the language itself is nothing particularly new. In the [5]LispWorks lisp system, the scheduler was written in Common Lisp as was the garbage collector. In a highly dynamic language like Common Lisp this was a valuable feature as we could fix problems in the field by giving patches to customers who could load the patches into the system to use the new definitions. This allowed bugs in the runtime system as well as the compiler to be fixed dynamically.

The HECs also reminded me strongly of the smalltalk context objects from smalltalk-80. I'll have more to say about that later.

The thesis itself was an interesting read.

1. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.8039>
  2. <http://www.cis.upenn.edu/~lipeng/homepage/papers/dissertation.pdf>
  3. <http://lambda-the-ultimate.org/>
  4. <http://clivetong.spaces.live.com/blog/cns!3F21DF299C355E7F!120.entry>
  5. <http://www.lispworks.com/>
- 

## 3.2 February

### 3.2.1 You're so vain.... (2009-02-01 15:04)

It was while we were doing a vanity searches at work the other day, caused by [1]my current manager claiming that his name would get the most hits. There, amoungst a few hits on me and another Clive Tong (from Malta), was an attribution on some [2]Dylan code for a [3]CORBA implementation. It's nice to have some code out there in the public domain.

In around 1996, [4]Harlequin were developing [5]CORBA implementations in both the Common Lisp and Dylan groups. After a small amount of time working with another developer, I ended up doing all of the Common Lisp work. When the time came to do the Dylan work, Jason Trenouth and I worked together on a number of designs for the implementation, which I usually got to implement first in Common Lisp. Jason used some of this code as the basis for writing the Dylan implementation which, I assume, is the reason for the credit on the source files. As an aside, as I'd already worked on the early version of the company's ML compiler, I had been due to work on a CORBA binding for ML before the company started getting into financial problems.

CORBA was a nice way to make our languages usable as part of a solution written in more mainstream languages. The trouble with the Lisp implementation, at the time, was that it wanted to control the whole address space of the process. It could interoperate by loading in other components, but at the time it wasn't wasy to produce a dll or shared library which could be loaded into a different process. Exposing Common Lisp functionality as distributed objects made it easy for code written in Common Lisp to be used by other languages. At one point we had a demo that had components written in Common Lisp, Dylan, Java and C++ which all worked together. We also wrote, well Jason mainly wrote, a paper on the benefits of using interactive languages to script distributed objects.

For me, this has all come around again as I have been using F # to dynamically manage and debug a number of .NET components across remoting and web service boundaries. In this kind of scenario, the REPL (read-eval-print-loop) allows the inspection and modification of the components' state without the pain of either writing a C # program or connecting to the component with Visual Studio.

1. <http://www.simple-talk.com/community/blogs/richard/>
  2. [http://en.wikipedia.org/wiki/Dylan\\_programming\\_language](http://en.wikipedia.org/wiki/Dylan_programming_language)
  3. <https://www.gwydiondylan.org/cgi-bin/viewvc.cgi/branches/fundev-2-1-jam/fundev/Sources/corba/orb/poa/poa.dylan?revision=9549&view=markup&pathrev=9549>
  4. [http://en.wikipedia.org/wiki/Harlequin\\_\(software\\_company\)](http://en.wikipedia.org/wiki/Harlequin_(software_company))
  5. <http://en.wikipedia.org/wiki/CORBA>
- 

### **3.2.2 I've got a message for you! (2009-02-01 15:47)**

We were talking at work about the windows message loop and I remembered that [1]Raymond Chen's book covers this in more detail than I have ever seen elsewhere. I thought I'd take the opportunity to type up my understanding, mainly as a means of checking that I do understand this a little. This was all provoked by a discussion of .NET [2]SynchronizationContexts which I'll mention at the end.

As we know, Windows messages can be directed at other windows via a SendMessage or a PostMessage. Well, there are a few other API calls but we'll ignore them as they fit into the model in the obvious way. In Win32 land, the usual description is that SendMessage gets the message to the window directly by calling the window procedure whereas PostMessage puts the message into the message queue of the thread responsible for the target window. This thread polls for messages in its message loop, GetMessage, and then dispatches the message to the window procedure, DispatchMessage, after checking for things like accelerators, TranslateAccelerator, and after giving inbuilt utilities like the dialog manager a chance to do something with the message, IsDialogMessage.

The problem with this description is that it does not explain how a SendMessage causes the target thread to be interrupted in order for the window procedure to get its direct call. The answer of course is that it doesn't. All cross-thread message queue interaction in Windows happen in a cooperative fashion. It just wouldn't be safe to interrupt the target thread at an arbitrary point and there is no mechanism for describing safe points. Chen's book gives the key insight for understanding this. What we refer to as the message queue is in fact an object that contains three queues.

One queue is for sent messages that are targeted for a window handled by a thread which isn't the current thread. If we are sending a message to a window that is managed by the thread that we are on, we can simply call the window procedure directly. If the send is crossing a thread boundary, we have no means

to interrupt the target thread so we place the message in a queue for that thread to handle at some safe point in the future. We then wait for the response. The second queue is for posted messages, which we can fire and forget, and the third queue is for input events.

SendMessage messages can be dispatched to the window procedure at three points. If the target thread is blocked inside a cross-thread SendMessage itself, waiting for the reply, the message it is sent will hijack the thread and call the window procedure. Calls to PeekMessage and GetMessage also check the SendMessage queue before doing anything else and dispatch all messages in the queue directly to the window procedures. By this means, SendMessage messages overtake posted messages that may have been in the queue for some time.

Posted messages and input messages are handled inside PeekMessage. GetMessage is essentially a call to PeekMessage which passes relevant flags that cause no filtering on the message type and which cause the message to be removed after it is retrieved. It is PeekMessage which we need to understand to explain

how special messages such as WM \_QUIT, WM \_PAINT, WM \_MOUSEMOVE and WM \_TIMER are handled.

PeekMessage first dispatches anything in the SendMessage queue to the relevant window procedure. It then tries to find a message in the PostMessage queue. If something suitable is present in this queue it returns it.

If a QUIT is pending, determined by checking a thread flag, the flag is cleared, a WM \_QUIT message is generated and returned.

If the mouse has moved and a mouse move event is included in the filter a WM \_MOUSEMOVE is added to the input queue, but we do not return just yet.

Next the input queue is checked for a suitable message which is returned if it is present. We may remove this message depending on the flags.

If there are pending paints and a WM \_PAINT would satisfy the filter, then a WM \_PAINT is generated and returned. Note that the paint is never added to a queue.

Lastly we check for elapsed timers, and should any satisfy the filter, events are added to the input queue and returned. We may remove the message depending on the flags.

Note that when we return a message that was added to the input queue, we may not remove it. This will depend on whether the flags passed to PeekMessage specify that the removal should happen. This means that there are situations when the input queue can be filled by messages that were generated by calls to PeekMessage.

Now on to SynchronizationContext. This allows threads to be marked as having a means for having operations executed upon them. The builtin implementation executes calls on the Thread-Pool, but one can define subclasses of this class and associate them with threads. Several implementations are around in the .NET framework, in particular for windows forms and WPF.

When an instance of a windows forms object is made on a thread, the synchronisation context of the thread is set to be an instance of the WindowsFormsSynchronizationContext. In this case the Post and Send operations hook into the mechanisms detailed above, using PostMessage and SendMessage and a windows event loop to get the operation to happen on the relevant thread at a suitable safe point.

Synchronization contexts are also a means for getting the CLR to call back into the context object when either a wait is called on the thread or when an asynchronous operation is started allowing the context to track the number of outstanding operations. There is good coverage of this in [3]Joe Duffy's book.

1. [http://www.amazon.co.uk/Old-New-Thing-Development-Throughout/dp/0321440307/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1233486355&sr=8-1](http://www.amazon.co.uk/Old-New-Thing-Development-Throughout/dp/0321440307/ref=sr_1_1?ie=UTF8&s=books&qid=1233486355&sr=8-1)
  2. <http://msdn.microsoft.com/en-us/library/system.threading.synchronizationcontext.aspx>
  3. [http://www.amazon.co.uk/Concurrent-Programming-Windows-Architecture-Development/dp/032143482X/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1233499678&sr=8-1](http://www.amazon.co.uk/Concurrent-Programming-Windows-Architecture-Development/dp/032143482X/ref=sr_1_1?ie=UTF8&s=books&qid=1233499678&sr=8-1)
- 

### **3.2.3 A great read. (2009-02-02 19:05)**

I've just finished reading the first [1]Joel on Software book. It was a great read. Lots of interesting observations on the software industry and lots of good ideas for improving the state of the art. Some of the predictions turned out to be wrong and I'm not sure that I agree with all of the author's views, but a great read nevertheless.

1. [http://www.amazon.co.uk/Joe-Software-Occasionally-Developers-Designers/dp/1590593898/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1233601316&sr=8-1](http://www.amazon.co.uk/Joe-Software-Occasionally-Developers-Designers/dp/1590593898/ref=sr_1_1?ie=UTF8&s=books&qid=1233601316&sr=8-1)
-

### 3.2.4 It's not knitting (2009-02-06 21:13)

I was at a user group meeting the other day where the speaker was giving a brief introduction to design patterns. The first pattern, as always, was the classic Singleton. After talking about the bad points of the Singleton pattern itself, in particular the problems it causes for unit testing and the lack of scalability, the presenter talked through the double check locking implementation.

```
private static Class2 m_Singleton = null;  
private static object m_Lock = new object();
```

```
public static Class2 Instance()  
{  
    if (m_Singleton == null)  
    {  
        lock (m_Lock)  
        {  
            if (m_Singleton == null)  
            {  
                m_Singleton = new Class2();  
            }  
        }  
    }  
    return m_Singleton;  
}
```

He then pointed out that this was old school and that there was better way in C # to do the Singleton.

```
static Class1 m_Singleton = new Class1();
```

```
static Class1()  
{  
}
```

```
public static Class1 Instance()
{
    return m_Singleton;
}
```

Note that the empty static constructor is there to prevent the C# code compiling to IL with the field marked as beforefieldinit which would allow the CLR to run the initializer at any time. With the static constructor the first time that m\_Singleton can be initialised is on access to the class.

What he didn't point out was that there's a very obvious case where the "old" implementation is far better. If the field initialization throws an exception, then the "old" pattern will keep trying to make the Singleton when the Instance method is executed. The "new" style implementation will keep throwing the exception from the first failure.

```
public class Class1
{
    ... new style code above ...
}
```

```
static int m_Count = 0;
```

```
Class1()
{
    Console.WriteLine("Making instance");
    if (m_Count == 0)
    {
        Console.WriteLine("Throw");
        m_Count++;
        throw new Exception();
    }
}
```

```
}

}

public class Class2
{
... old style code above ....
```

```
static int m _Count = 0;
```

```
Class2()
{
Console.WriteLine("Making instance");
if (m _Count == 0)
{
Console.WriteLine("Throw");
m _Count++;
throw new Exception();
}
}
```

Trying the following harness code, first for Class1 and then with all instances of Class1 replaced by Class2,

```
static void Main(string[] args)
{
Exception caught = null;
try
{
Class1 a = Class1.Instance();
}
```

```

catch (Exception x)
{
Console.WriteLine("Exception");
caught = x;
}
try
{
Class1 b = Class1.Instance();
}
catch (Exception x)
{
Console.WriteLine("Exception 2 - {0}", x == caught);
}
}

```

We see that the "old" style keeps trying to make a Singleton and snapshots it on success, whereas the "new" style has one go at making it and then returns the same exception object thereafter.

This point brought home to me just how subtle code can be and how often "equivalent" code is only equivalent from one perspective but behaves completely differently in other dimensions. In the "new" style, the mechanics of the locking has been pushed into the virtual machine making the code cleaner, but this has also resulted in a slight loss of control for the programmer.

The GOF patterns are great but are not universal across all languages. The talk caused me to have another look at Greg Sullivan's survey of the standard patterns and how they are implemented in dynamic languages, b[1]etter patterns through reflection, which is a good read.

---

1. <ftp://publications.ai.mit.edu/ai-publications/2002/AIM-2002-005.pdf>

### **3.2.5 Virtually brilliant - part one (2009-02-08 22:14)**

It all happened in 1994 when I came across [1]the book. I couldn't put it down. The idea that you could define a virtual machine which people could implement, then sent them a heap image for the system and have it work on their implementation was a magical idea. Especially as the image could contain a portable gui that all worked using the BitBlt primitive which it used for drawing graphics and fonts and the windows itself. More impressively the system supported concurrent processes, multitasking by switching the byte interpretation to a new context on a timer. Lastly, most of the system was written in itself - the debugger, the compiler and all of the graphics. From there it was on to reading the [2]"purple book" though the book I really wanted, and which I couldn't get hold of, was the earlier blue book which covered all of the definitions for the virtual machine and its object memory. A while back I found the definitions [3]here.

In a series of posts, I'm hoping to communicate some of the beauty of the Smalltalk VM, particularly the way that the whole system including the compiler can be written in Smalltalk with the VM hooking into the system via the Behavior class. By implementing this using F # I am hoping to get a better understanding of this interplay between the system and the VM across this interface.

However, this is going to take some effort so in this first post I want to get the basic instance and class structure set up. All we are going to look at is representing data. In later posts we'll look at adding methods and then object creation into the picture. This will mean that we'll have to add to the classes that we define here, in terms of methods and instance variables. We'll also be starting with a class structure that looks like it has too many classes. I will simplify things from real Smalltalk in some cases, but the meaning of some of these classes won't become clear until we get closer to the VM.

The original smalltalk virtual machine divided the space of words into two by using the lower bit as a tag. One represented an integer in 2's complement form and a zero lower bit represented a pointer value. This pointer pointed into an object table that contained details of the class of the object together with a pointer to the object's data area.

```
type SmalltalkValue = SmallInt of int | Instance of InstanceValue
```

```
and InstanceValue = { mutable classType : SmalltalkValue option; slots : SlotValues }
```

```
and SlotValues = Standard of SmalltalkValue array | StringValue of string | Bytes of byte array;;
```

The first key detail is that values are either small integers that are representable in a word of memory or are instances of another type. Hence whenever we have a value in our hand we may ask for the objects type, first by checking if it is the well known integer value in which case the VM has a pointer to the class object for the SmallInteger type or we can go to the class slot inside the instance object and follow that to get the class object. The classType is a mutable option. This is because the actual class hierarchy is quite large and contains some circularities which can't be filled in when the object is created.

The instance itself contains some values - typically these are the values of the instance variables which will be represented as an array of normal Smalltalk values. For items like strings we'll have a special representation and for code objects we'll use a byte array. The real Smalltalk-80 uses an object table to record all live instances. We'll not be simulating that here.

In some of the coming examples we'll use a Point object as an example. A Point instance has an x and y coordinate, so we'd represent the [4]1@2, the point with x coordinate 1 and y coordinate 2 as

```
let point _1 _2 = Instance { classType = None; slots = Standard [| SmallInt 1; SmallInt 2 |] };;
```

Smalltalk now has something rather clever. Our point (1,2) is an instance of a class named Point, like in languages such as C#. However, we're also going to make this class object an instance of another class, a metaclass. Inside the class object for a Point we are going to need to maintain an array of instance variable names. This will be represented using an array of strings.

```
let string _x = Instance { classType = None; slots = StringValue "x" }
```

```
let string _y = Instance { classType = None; slots = StringValue "y" }
```

```
let PointInstanceVars = Instance { classType = None; slots = Standard [| string _x; string _y |] };;
```

We'll fill in the classTypes later to reflect the types of the objects. For the strings we'll need a String class and for the array we'll need an array class. Points are also going to inherit from a base class type named Object. Let's start constructing the class object for object. This has slots representing the superclass, a slot representing the method dictionary and a slot representing the array of instance variables introduced by this class. We'll use nil for this for the moment. In order for us to define the superclass of Object as nil, we need an object representing nil. In SmallTalk, nil is the singleton instance of the class UndefinedObject.

```
let nil = Instance { classType = None; slots = Standard [| |] }
```

```
let UndefinedObjectInstanceVars = Instance { classType = None; slots = Standard [| |] }
```

```
let ObjectInstanceVars = Instance { classType = None; slots = Standard [| |] }
```

```
let Object = Instance { classType = None; slots = Standard [| nil; nil; ObjectInstanceVars; |] }
```

```
let UndefinedObject = Instance { classType = None; slots = Standard [| Object ; nil; UndefinedObjectInstanceVars|] }
```

We can now define a function for filling in the class type of an instance.

```
let setType instance value =
match instance with
| Instance y -> y.classType <- Some value
| _ -> failwith "bad instance";;
```

We can now use this to set the type of nil.

```
setType nil UndefinedObject;;
```

Let us also define the String and Array classes. These are both subclasses of Object.

```
let String = Instance { classType = None; slots = Standard [| Object ; nil; UndefinedObjectInstanceVars |] }
```

```
let Array = Instance { classType = None; slots = Standard [| Object ; nil ; UndefinedObjectInstanceVars |] }

setType string _x String;;
setType string _y String;;
setType UndefinedObjectInstanceVars Array;;
setType ObjectInstanceVars Array;;
setType PointInstanceVars Array;;
```

We may now define the Point class

```
let Point = Instance { classType = None; slots = Standard [| Object ; nil; PointInstanceVars |] }

setType point _1 _2 Point;;
```

Now we're starting to have a class hierarchy! We have a type Object with subclasses such as Point and UndefinedObject. We have made an instance of the former bound into point \_1 \_2 and of the latter bound to nil.

```
let Number = Instance { classType = None; slots = Standard [| Object ; nil ;UndefinedObjectInstanceVars|] }
```

```
let SmallInteger = Instance { classType = None; slots = Standard [| Number ; nil ; UndefinedObjectInstanceVars|] }
```

We can get the class of an instance using

```
let ClassOf x =
  match x with
  | Instance x ->
    match x.classType with
    | None -> failwith "Improper class hierarchy"
    | Some c -> c
    | SmallInt y -> SmallInteger
```

```
let Super x =
  match x with
  | Instance data ->
    match data.slots with
    | Standard slots -> slots.[0]
    | _ -> failwith "Invalid class instance"
    | _ -> nil
```

```
let rec Subtype x y =
  if x = y
  then true
  else
    if Super x = nil
    then false
    else Subtype (Super x) y
```

We can check that point \_1 \_2 is an instance of Object, because the first two of the following return true and the third retruns false.

Subtype (ClassOf point \_1 \_2) Object

Subtype (ClassOf point \_1 \_2) Point

Subtype Number Point

We can write some functions to determine all of the instance variables for a given instance:

```
let UnpackString s =
match s with
| Instance data ->
if data.classType <> Some String
then failwith "Expecting a string"
else
match data.slots with
| StringValue x -> x
| _ -> failwith "Invalid string data"
| _ -> failwith "Expecting a string"
```

```
let FlattenArray inst =
match inst with
| Instance data ->
if data.classType <> Some Array
then failwith "Expecting an array"
```

```
else
match data.slots with
| Standard data -> List.map UnpackString (List.of_array data)
| _ -> failwith "Expected an array"
| _ -> failwith "Expected an array"

let rec AllInstVarNames c =
if c = nil
then []
else
match c with
| Instance data ->
match data.slots with
| Standard slots -> (AllInstVarNames (Super c)) @ (FlatternArray slots.[2])
| _ -> failwith "Invalid class instance"
| _ -> failwith "Invalid class"
```

And test it using

AllInstVarNames Object

AllInstVarNames Point

Now is where things start to get interesting. Object represents a class, but there is no reason why it should be special. It would be more useful if Object were itself an instance of another class.

```

let string _superclass = Instance { classType = Some String; slots = StringValue "superclass"
}

let string _methodDictionary = Instance { classType = Some String; slots = StringValue
"methodDict" }

let BehaviorInstanceVars =
Instance { classType = Some Array; slots = Standard [| string _superclass; string _methodDictionary |] };;

let Behavior = Instance { classType = None; slots = Standard [| Object ; nil; BehaviorInstance-
Vars |] }

let string _instanceVars = Instance { classType = Some String; slots = StringValue "instance-
Variables" }

let ClassDescriptionInstanceVars =
Instance { classType = Some Array; slots = Standard [| string _instanceVars |] };;

let ClassDescription = Instance { classType = None; slots = Standard [| Behavior ; nil ; Class-
DescriptionInstanceVars |] }

let Class = Instance { classType = None; slots = Standard [| ClassDescription ; nil ; Undefine-
dObjectInstanceVars|] }

let MetaClass = Instance { classType = None; slots = Standard [| ClassDescription ; nil ; Unde-
finedObjectInstanceVars|] }

let ObjectClass = Instance { classType = Some MetaClass; slots = Standard [| Class ; nil; Un-
definedObjectInstanceVars |] }

setType Object ObjectClass

```

```
let BehaviorClass = Instance { classType = Some MetaClass; slots = Standard [] ObjectClass ; nil; UndefinedObjectInstanceVars[] }
```

```
setType Behavior BehaviorClass
```

```
let ClassDescriptionClass = Instance { classType = Some MetaClass; slots = Standard [] BehaviorClass ; nil; UndefinedObjectInstanceVars[] }
```

```
setType ClassDescription ClassDescriptionClass
```

```
let MetaClassClass = Instance { classType = Some MetaClass; slots = Standard [] ClassDescriptionClass ; nil; UndefinedObjectInstanceVars[] }
```

```
setType MetaClass MetaClassClass
```

```
let ClassClass = Instance { classType = Some MetaClass; slots = Standard [] ClassDescriptionClass ; nil; UndefinedObjectInstanceVars[] }
```

```
setType Class ClassClass
```

Now we can add what we needed to do when we created the class Point. We needed to also create a class named PointClass, making it the type of Point. This class needs to inherit from ObjectClass and is of type MetaClass.

```
let PointClass = Instance { classType = Some MetaClass; slots = Standard [] ObjectClass ; nil; UndefinedObjectInstanceVars[] }
```

In a full implementation of Smalltalk-80, [5]Squeak, we get the following values.

```
Point allInstVarNames -> #('x' 'y')
```

```
Point class allInstVarNames -> #('superclass' 'methodDict' 'format' 'instanceVariables' 'organization' 'subclasses' 'name' 'classPool' 'sharedPools' 'environment' 'category' 'traitComposition' 'localSelectors')
```

In our model we get the values:

```
> AllInstVarNames Point;;
val it : string list = ["x"; "y"]
> AllInstVarNames PointClass;;
val it : string list = ["superclass"; "methodDict"; "instanceVariables"]
```

So it looks like we might have the initial class hierarchy right. Next time we will move on to adding some methods and a partial VM for interpreting them. If any of this is of interest, Eliot Miranda's [6]Cog Blog is currently covering his improvements to the existing VM implementation in Squeak.

1. [http://www.amazon.co.uk/Smalltalk-80-Bits-History-Words-Advice/dp/0201116693/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1233440104&sr=8-1](http://www.amazon.co.uk/Smalltalk-80-Bits-History-Words-Advice/dp/0201116693/ref=sr_1_1?ie=UTF8&s=books&qid=1233440104&sr=8-1)
2. [http://www.amazon.co.uk/Smalltalk-80-Language-Addison-Wesley-Computer-Science/dp/0201136880/ref=sr\\_1\\_2?ie=UTF8&s=books&qid=1233440218&sr=1-2](http://www.amazon.co.uk/Smalltalk-80-Language-Addison-Wesley-Computer-Science/dp/0201136880/ref=sr_1_2?ie=UTF8&s=books&qid=1233440218&sr=1-2)
3. <http://users.ipa.net/~dwighth/>
4. mailto:1@2

- 
5. <http://www.squeak.org/>
  6. <http://www.mirandabanda.org/cogblog/>

### **3.2.6 I'll parse that in a sec (2009-02-08 22:24)**

I've been doing some reading on denotational semantics lately and want a quick way to write a parser for some of the demonstration languages. [1]FParsec looks like it might be great for the job. I haven't had time to try it yet, but I did have some time to read the paper on the original [2]library on which it is based. It looks interesting.

- 
1. <http://www.quanttec.com/fparsec/>
  2. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.5807>

### **3.2.7 It's a joint effort (2009-02-14 10:06)**

I met [1]Claudio Russo when I was working at [2]Harlequin. He worked on the Dylan team and we worked together a little when he was implementing a CORBA ORB. He later moved to [3]Microsoft Research Cambridge and I occasionally browse through his publications on such interesting topics as SML.NET and [4]GADTs. Ages ago I read the paper on the [5]joins library which came about as a portable expression of some of the constructs of the experimental [6]C-omega language. Recently, joins support was [7]added into an experimental version of the Visual Basic compiler.

The joins idea is fairly straight forward. We make instances of join points which we communicate with via channels. The join point can be told what function to call when values are available on a subset of its channels. Communication with the join point via the channel can be synchronous or asynchronous. In the synchronous case the value of the executed function defined on the join point can be returned to the client. In the asynchronous case, no value is returned.

Let's try a quick example.

```
#I "C:\Program Files\Microsoft Research\Joins\LibraryAssemblies";;
#r "Microsoft.Research.Joins.dll";;
open Microsoft.Research.Joins;;
```

We allocate a join point.

```
let join = Join.Create();;
```

We define a synchronous method which communicates a string to the join point, and returns an int result from any function that gets executed on the values.

```
let get = Synchronous<int>.CreateChannel<string>(join);;
```

We define another method that communicates a string. This is asynchronous, so it will not wait for the join point to execute any functions.

```
let put = Asynchronous.CreateChannel<string>(join);;
```

Let's define a method on the join point. This waits until get and put channels have pending values, at which point it prints the two strings and returns the sum of their lengths.

```
type StringPattern = JoinPatterns.JoinPattern<int>.OpenPattern<string,string>;
```

```
join.When(get).And(put).Do(
StringPattern.Continuation(
fun a ->
fun b ->
printfn "Arg1: %s" a;
printfn "Arg2: %s" b;
a.Length + b.Length));;
```

get and put are realised as delegates. The syntax for calling them in F # isn't as concise as it C #, so we'll need to use the Invoke. In a real example we'd encapsulate them into new functions. We can push multiple values into the put channel.

```
> put.Invoke "hello";;
val it : unit = ()
> put.Invoke "mum";;
val it : unit = ()
```

As soon as we push a value into the get channel, the function on the join point is triggered, returning a result.

```
> get.Invoke "how";;
Arg1: how
Arg2: hello
val it : int = 8
```

We didn't lose the second value we pushed. It was buffered in the channel.

```
> get.Invoke "are";;
Arg1: are
Arg2: mum
val it : int = 6
```

Get is synchronous so it will block if a function cannot be triggered. In the following example, the thread calling the get is blocked until the background thread executes.

```
System.Threading.ThreadPool.QueueUserWorkItem(
fun _ ->
    System.Threading.Thread.Sleep(10000);
    put.Invoke "another");;

get.Invoke "you";;
```

As a slightly less contrived example, we can implement a standard one element buffer. We'll have put and get channels as well as a channel for denoting that the buffer is empty and a channel for holding on to the buffered value.

```
let buffer = Join.Create();;
```

The get channel will be used for returning the string value from the buffer. If no value is present it will block until a value is ready.

```
let bufferGet : Synchronous<string>.Channel = Synchronous<string>.CreateChannel(buffer-);;
```

We can synchronously put values into the channel.

```
let bufferPut = Synchronous<string>.CreateChannel<string>(buffer);;
```

We need two internal channels. The first signals when the buffer is empty.

```
let bufferEmpty : Asynchronous.Channel = Asynchronous.CreateChannel(buffer);;
```

And the second channel contains the value that is held by the buffer.

```
let bufferContains = Asynchronous.CreateChannel<string>(buffer);;
```

If the buffer is holding a value in its Contains channel and someone calls Get, then we return the value and put ourselves back into the empty state.

```
type BufferFetchPattern = JoinPatterns.JoinPattern<string>.OpenPattern<string>;
```

```
buffer.When(bufferGet).And(bufferContains).Do(
    BufferFetchPattern.Continuation(
        fun a ->
            bufferEmpty.Invoke();
            a
    )));

```

If the buffer is empty and a value is added to the Put channel, then we capture it in the buffer.

```
let response = BufferFetchPattern.Continuation<string,string>(fun x -> (bufferContains.Invoke
    x; x));;
```

```
buffer.When(bufferPut).And(bufferEmpty).Do(response);;
```

The buffer is initially in the empty state.

```
bufferEmpty.Invoke();;
```

```
> bufferPut.Invoke "1";;
val it : unit = ()
> bufferGet.Invoke();;
val it : string = "1"
```

If we try to put into a filled buffer, then the put operation blocks.

```
> bufferPut.Invoke "2";;
val it : string = "2"
> bufferPut.Invoke "3";;
- Interrupt
```

```
> bufferGet.Invoke();;
val it : string = "2"
```

Get will also block until a value is ready for us to fetch.

```

System.Threading.ThreadPool.QueueUserWorkItem(
fun _ ->
System.Threading.Thread.Sleep(10000);
bufferPut.Invoke "another";
());
;

bufferGet.Invoke ();
;
```

The [8]joins page and the [9]slide deck contain more details of the implementation and numerous examples such as how actors can be implemented in this model.

1. <http://research.microsoft.com/en-us/people/crusso/>
  2. [http://en.wikipedia.org/wiki/Harlequin\\_\(software\\_company\)](http://en.wikipedia.org/wiki/Harlequin_(software_company))
  3. <http://research.microsoft.com/en-us/labs/Cambridge/>
  4. <http://research.microsoft.com/en-us/um/people/akenn/generics/gadtoop.pdf>
  5. <http://research.microsoft.com/en-us/um/people/crusso/papers/pad107.pdf>
  6. <http://research.microsoft.com/en-us/um/cambridge/projects/comega/>
  7. <http://research.microsoft.com/en-us/um/people/crusso/papers/cb.pdf>
  8. <http://research.microsoft.com/en-us/um/people/crusso/joins/index.htm>
  9. <http://research.microsoft.com/en-us/um/people/crusso/papers/pad107.ppt>
- 

### **3.2.8 Basically you can join things together (2009-02-27 22:25)**

Coincidentally, after my last post on the joins library, a video on concurrent basic has just been posted on channel9: [1]Expert to Expert: Inside Concurrent Basic.

1. <http://channel9.msdn.com/shows/Going+Deep/Claudio-Russo-and-Lucian-Wischik-Inside-Concurrent-Basic/>
- 

## 3.3 March

### 3.3.1 Ajax without the 'j' (2009-03-05 18:22)

I've been meaning to dive deeper into the meta-programming side of F # for some time, in particular to try to finish off a [1]side-project I started work on some time ago. In my stash of papers that I intend to read, I've had [2]TomasP's thesis for some time and decided to give it a read to get more data about the facilities. As an aside he gives a very good summary of the contents [3]here. The thesis project can be downloaded [4]here, though it doesn't work on the latest releases of F # - I spent a little time trying to fix it up in vain.

The idea is really grand... you simply write everything in F # and the system compiles the client side version of the F # into javascript and runs it on the client side, passing values back and forth between the client and server using serialization. Definitions can be processed by the system to make browser javascript functions, like those for manipulating the DOM, available to the client side F # code. Computation expressions, the monadic syntax from F #, are used to mark the client and server side parts of the code and allow the type system to be used to keep them separate.

It all looks really neat and would be great fun to play with... hopefully, it will be fixed up at some point to run in the latest versions of F #. The only missing part seems to be the debugging story; there are some examples of the translated javascript versions of some F # functions in the thesis, but this is likely to be hard to debug. However, lots of the current Ajax style applications still use server side processing for doing all of the hard work and use the client side code for doing the UI manipulation so things might not be too bad. It's certainly something I will be keeping an eye on.

1. <http://clivetong.spaces.live.com/blog/cns!3F21DF299C355E7F!188.entry>
2. <http://tomasp.net/articles/webtools-thesis/fswebtools.pdf>

3. <http://tomasp.net/articles/fswebtools-intro.aspx>
  4. <http://www.codeplex.com/fswebtools>
- 

Frank (2009-03-07 12:15:57)

Yeah, same thing here! I also fiddled with it but couldn't get it to work. Hopefully Tomas will be able to finish his book and get to fixing it.

### **3.3.2 Just like busses (2009-03-05 22:11)**

Two great posts from fellow Red Gate people, both posted on the same day and both with the potential of generating lively discussions. [1]The Greasy Pole hits the nail on the head regarding developers and why management shouldn't be the only career path after a few years of coding; coincidentally, out of the three books mentioned, I have only read the first two. [2]Projects from hell - chill or scream? mentions the project our team has been working on.

1. [http://www.simple-talk.com/community/blogs/tony\\_davis/archive/2009/03/04/72324.aspx](http://www.simple-talk.com/community/blogs/tony_davis/archive/2009/03/04/72324.aspx)
  2. <http://blog.businessofsoftware.org/2009/03/projects-from-hell---chill-or-scream.html>
- 

### **3.3.3 Some low level surgery on F# (2009-03-10 20:03)**

It's been a bit annoying, not being able to try out the F # web tools as I documented [1]here. At the weekend, I decided to jump in and see if there was anything more I could do. Note that I am working here with the 1.9.3.14 version of the F # code base, and I'm not even sure if this is the right solution. It seems to get me further though.

After downloading the [2]F # web tools and making the necessary changes to the configuration, I was at the stage of getting the [3]following error.

Exception Details: System.Reflection.AmbiguousMatchException: Ambiguous match found.

Source Error:

```
Line 101: if (mi.MemberType = MemberTypes.Constructor) then Seq.empty else
Line 102: (mi.GetGenericArguments() |> Array.to_seq |> Seq.map ( fun _ -> objType ) ); ]) |>
Seq.to_list;
Line 103: let defn = ResolveTopDefinition (modDef, genArgs) in
Line 104: let defn = if defn = None then ResolveTopDefinition (modDef2, genArgs) else defn in
Line 105:
```

Source File: C:\fsharp\fswebtools\source\WebTools\CoreWeb\_WebTools.fs Line: 103

Stack Trace:

```
[AmbiguousMatchException: Ambiguous match found.]
System.DefaultBinder>SelectMethod(BindingFlags bindingAttr, MethodBase[] match, Type[] types, ParameterModifier[] modifiers) +2798322
System.RuntimeType>GetConstructorImpl(BindingFlags bindingAttr, Binder binder, CallingConventions callConvention, Type[] types, ParameterModifier[] modifiers) +134
System.Type>GetConstructor(BindingFlags bindingAttr, Binder binder, Type[] types, ParameterModifier[] modifiers) +63
Microsoft.FSharp.Quotations.Raw.bindCtor(Tuple'2 _arg44, List'1 tyargs) +90
```

As the F# release comes with source code, I could see that the GetConstructor is being called in the quotations.fs file in the bindCtor function.

```
let bindCtor ((tc,argTypes : (Type list tyenv)),tyargs) =
let typ = MkNamedType(tc,tyargs)
let argtyps = argTypes (List.nth tyargs)
typ.GetConstructor(bindingFlags,null,Array.of _list argtyps,null) |> nonNull ("bindCtor: failed
to bind constructor")
```

It is passing the binding flags:

```
let bindingFlags = BindingFlags.Instance ||| BindingFlags.Static ||| BindingFlags.Public |||
BindingFlags.NonPublic ||| BindingFlags.DeclaredOnly
```

The code in question is trying to regenerate the top level reflected definitions for the code in the WebTools.Controls module. These top level definitions are stored as resources within the assembly. When a call is made to fetch the reflected definitions, they are reloaded from the resource stream and cached. The only odd thing is that the system is using BindingFlags.Static when reflecting over the constructors of a type.

I was running on Vista, so I called up windbg, running it as administrator, and connected to the w3wp.exe ASP.NET worker process. We now need to find and breakpoint the correct overload of GetConstructor. This is relatively straight forward using the SOS debugger extensions that come as part of the framework.

We load up the sos extensions and find the method table for the System.Type type.

```
0:023> .loadby sos mscorewks  
0:023> !Name2EE *!System.Type
```

Module: 790c2000 (mscorlib.dll)  
Token: 0x020000f1  
MethodTable: 79106894  
EEClass: 79106824  
Name: System.Type

We can then dump the method table to find the overload that interests us.

```
0:023> !DumpMT -MD 79106894
```

793a04d4 7925e390 PreJIT System.Type.GetConstructor(System.Reflection.BindingFlags, System.Reflection.Binder, System.Type[], System.Reflection.ParameterModifier[])

And we can get the details about the method including its code.

```
0:023> !DumpMD 7925e390  
Method Name: System.Type.GetConstructor(System.Reflection.BindingFlags, System.Reflection.Binder, System.Type[], System.Reflection.ParameterModifier[])  
Class: 79106824  
MethodTable: 79106894  
mdToken: 06000d50  
Module: 790c2000  
IsJitted: yes  
m_CodeOrIL: 793a04d4
```

We can disassemble to look at the function.

```
0:023> u 793a04d4
mscorlib _ni+0x2e04d4:
793a04d4 57 push edi
793a04d5 56 push esi
793a04d6 53 push ebx
793a04d7 55 push ebp
793a04d8 8b5c241c mov ebx,dword ptr [esp+1Ch]
793a04dc 8b7c2418 mov edi,dword ptr [esp+18h]
793a04e0 8b6c2414 mov ebp,dword ptr [esp+14h]
793a04e4 85ff test edi,edi
```

And then set a breakpoint on the first instruction.

```
0:023> bp 793a04d4
```

We can then run until we get to a point where the backtrace looks like that of the ASP.NET error page.

```
0:023> g
Breakpoint 0 hit
eax=05c4d960 ebx=05902050 ecx=05902050 edx=0000003e esi=05a57db4 edi=0197f9d4
```

```

eip=793a04d4 esp=0fa4e8a0 ebp=0fa4e8bc iopl=0 nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000206
mscorlib _ni+0x2e04d4:
793a04d4 57 push edi
0:021> !ClrStack
OS Thread Id: 0x1cbc (21)
ESP EIP
0fa4e8a0 793a04d4 System.Type.GetConstructor(System.Reflection.BindingFlags, System.Reflection.Binder, System.Type[], System.Reflection.ParameterModifier[])
0fa4e8b0 0ea7f4b7 Microsoft.FSharp.Quotations.Raw.bindCtor(Microsoft.FSharp.Core.Tuple`2<System.Type,Microsoft.FSharp.Core.FastFunc`2<Microsoft.FSharp.Core.FastFunc`2<Int32,System.Type>,Microsoft.FSharp.Collections.List`1<System.Type>>>, Microsoft.FSharp.Collections.List`1<System.Type>)
0fa4e8c4 0ea87f5e [4]Microsoft.FSharp.Quotations.Raw+u _constSpec@1155
_1.Invoke(Microsoft.FSharp.Collections.List`1<System.Type>)
0fa4e8d4 0ea89e3f [5]Microsoft.FSharp.Quotations.Raw+u _expr@1110
_2.Invoke(Microsoft.FSharp.Core.FastFunc`2<Int32,System.Type>)
0fa4e8ec 0ead27b1 Microsoft.FSharp.Collections.PrivateListHelpers.map[[System. __Canon, mscorlib],[System. __Canon, mscorlib]](Microsoft.FSharp.Core.FastFunc`2<System. __Canon, System. __Canon>, Microsoft.FSharp.Collections.List`1<System. __Canon>)
0fa4e948 0ea89e67 [6]Microsoft.FSharp.Quotations.Raw+u _expr@1110
_2.Invoke(Microsoft.FSharp.Core.FastFunc`2<Int32,System.Type>)
0fa4e960 0ea8772e [7]Microsoft.FSharp.Quotations.Raw+u _expr@1115.Invoke(Microsoft.FSharp.Core.FastFunc`2<Int32,System.Type>) 0fa4e974 0ead28f2 Microsoft.FSharp.Collections.PrivateListHelpers.map[[System. __Canon, mscorlib],[System. __Canon, mscorlib]](Microsoft.FSharp.Core.FastFunc`2<System. __Canon, System. __Canon>, Microsoft.FSharp.Collections.List`1<System. __Canon>)

```

We find it is the sixth call which fails with the following stack. We can use -a to look at the addresses of the arguments.

```

0:021> !ClrStack -a
OS Thread Id: 0x1cbc (21)
ESP EIP
0fa4e530 793a04d4 System.Type.GetConstructor(System.Reflection.BindingFlags, System.Reflection.Binder, System.Type[], System.Reflection.ParameterModifier[])
PARAMETERS:
this = 0x05999fd0
bindingAttr = 0x00000003e
binder = 0x000000000
types = 0x05c39708

```

```
modifiers = 0x00000000
LOCALS:
<no data>
```

We can look at the this point to check that it looks relevant.

```
0:021> !DumpObj 0x05999fd0
Name: System.RuntimeType
MethodTable: 790ff734
EEClass: 790ff6c4
Size: 20(0x14) bytes
Type MethodTable: 7912f0c0
Type Name: System.Collections.Generic.List`1[[System.Object, mscorelib]]
```

When I did this the first time, I simply tried changing the BindingFlags which I found in the edx register from 0x3e to 0x37 to get rid of the Static binding flag. That worked, so now we'll patch the generated code to pass this instead of the original.

We dump the stack to find the return address.

```
0:021> dd esp
0fa4e530 0ea7f4b7 00000000 05c39708 00000000
0fa4e540 05c3963c 05c3967c 05c3966c 05c3963c
0fa4e550 0ea87f5e 05c389c8 0195d840 05c3965c
0fa4e560 0ea89e3f 0fa4e574 0edb71c 0ed13c24
0fa4e570 0ee8f780 0195d858 0ead28f2 0ed13c24
0fa4e580 0ed13c24 0edb718 0edb714 0ed13c24
0fa4e590 0ed13c24 0edb718 0ed5c784 05953128
0fa4e5a0 05c3952c 0edb714 05c3953c 0195d858
```

```

0:021> u 0ea7f4b7-20
FSharp_Core_ni+0x3f497:
0ea7f497 0e push cs
0ea7f498 8b30 mov esi,dword ptr [eax]
0ea7f49a 6a00 push 0
0ea7f49c ba60ffe70e mov edx,offset FSharp_Core_ni+0x43ff60 (0ee7ff60)
0ea7f4a1 e8c2ec0300 call FSharp_Core_ni+0x7e168 (0eabe168)
0ea7f4a6 50 push eax
0ea7f4a7 6a00 push 0
0ea7f4a9 8bcb mov ecx,ebx
0:021> u
FSharp_Core_ni+0x3f4ab:
0ea7f4ab ba3e000000 mov edx,3Eh
0ea7f4b0 3909 cmp dword ptr [ecx],ecx
0ea7f4b2 e81d10926a call msclib_ni+0x2e04d4 (793a04d4)
0ea7f4b7 8bd0 mov edx,eax
0ea7f4b9 684849e80e push offset FSharp_Core_ni+0x444948 (0ee84948)
0ea7f4be 8bce mov ecx,esi
0ea7f4c0 b838f3a70e mov eax,offset FSharp_Core_ni+0x3f338 (0ea7f338)
0ea7f4c5 6a00 push 0

```

We can now see the load of 0x3E into the edi register prior to the call. We can patch that using the memory window in windbg.

Revisiting the ASP.NET page, [8]things now appear to be working. However I don't have SQL server on my home machine so I get an exception later in the run when the server is running callback code, though at least we get to that point.

1. <http://clivetong.spaces.live.com/blog/cns!3F21DF299C355E7F!220.entry>
2. <http://www.codeplex.com/fswebtools>
3. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Pictures/fswebtools-not-working.jpg>
4. mailto:Microsoft.FSharp.Quotations.Raw+u\_constSpec@1155\_1.Invoke(Microsoft.FSharp.Collections.List`1<System.Type)
5. mailto:Microsoft.FSharp.Quotations.Raw+u\_expr@1110\_2.Invoke(Microsoft.FSharp.Core.FastFunc`2<Int32, System.Type)
6. mailto:Microsoft.FSharp.Quotations.Raw+u\_expr@1110\_2.Invoke(Microsoft.FSharp.Core.FastFunc`2<Int32, System.Type)

7. mailto:Microsoft.FSharp.Quotations.Raw+u\_expr@1115.Invoke(Microsoft.FSharp.Core.FastFunc`2<Int32, System.Type)
  8. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Pictures/fswebtools-working.jpg>
- 

### 3.3.4 You will keep count won't you (2009-03-11 20:51)

We were having a discussion at [1]work the other day regarding one of the low level Marshal functions for doing COM operations in .NET, the [2]GetObjectForIUnknown function. This takes an [3]IntPtr containing the address of a COM object and wraps it with a .NET object that offers a number of useful methods. We reasoned that the GetObjectForIUnknown must increment the reference count of the COM object, which meant that the user must be responsible for releasing the reference represented by the IntPtr too. However, we wanted to be sure of this. The solution was to quickly knock together some C++ code representing a simple COM object that only implements IUnknown, and then exercise this from some .NET code. In our case this code was written in C #, but in this post I'm going to do the same operations from F #.

To qualify as a COM object is fairly simple. You need to implement the IUnknown interface, which at the binary level, means implementing a vtable with three methods. The following C++ code does just that.

First we define the standard IID structure, which holds the guid for any interface for which a COM object is being demanded.

```
typedef struct
{
    unsigned long Data1;
    unsigned short Data2;
    unsigned short Data3;
    char Data4[ 8 ];
} IID;
```

We need to recognise the IUnknown interface which has a standard guid.

```
bool IsIUnknown(IID *target)
{
    if (target->Data1 != 0) return false;
    if (target->Data2 != 0) return false;
    if (target->Data3 != 0) return false;
    if (target->Data4[0] != (char) 0xc0) return false;
    if (target->Data4[7] != (char) 0x46) return false;
    for(int i = 1; i < 7; i++)
    {
        if ((int) target->Data4[i] != 0) return false;
    }
    return true;
}
```

Next we define a C++ class with three virtual methods. These correspond to QueryInterface, AddRef and Release, the three key methods of COM. The first is a way for checking which interfaces the object supports, the second is used when code keeps hold of a pointer which it was passed (say as an argument in a function) and the third is used when a pointer is no longer being used. We add some debugging statements to these methods. The class maintains a reference count of the outstanding references, allowing it to track when it is no longer being used. In the QueryInterface method, we only succeed if the client is asking for our IUnknown interface, returning a standard error result (E\_NOINTERFACE) if another interface is requested.

```
class MyFoo
{
public:
    long refCount;
```

```
virtual long __stdcall QueryInterface(IID * a, void **result)
{
if (IsIUnknown(a))
{
result = this;
refCount++;
fprintf (stderr, "Returning IUnknown refCount: %xn", refCount);
return 0;
}
else
{
result = NULL;
return 0x80004002L;
}
}

virtual long __stdcall AddRef()
{
++refCount;
fprintf(stderr, "Addref: %xn", refCount);
return refCount;
}

virtual long __stdcall Release()
{
-refCount;
fprintf (stderr, "Release: %xn", refCount);
if (refCount == 0)
{
fprintf(stderr, "DEADn");
}
return refCount;
}

};
```

Finally, we expose a factory method for generating fresh instances. We will be building this project as a dll so that it can run in memory in any client. When we pass out any objects, we set their initial reference count to 1 ie any callers of MakeOne take ownership of the returned pointer, so it is their responsibility to release the object when they are finished with it.

```
extern "C" __declspec(dllexport) MyFoo* __stdcall MakeOne()
{
    MyFoo* f = new MyFoo();
    f->refCount = 1;
    return f;
}
```

We can now exercise the object from F #.

First we open some namespaces.

```
> open System;;
> open System.Runtime.InteropServices;;
```

Then we define the interface to our dll and its factory method.

```
> [<DllImport(@"C:/Profiles/clive.tong/My Documents/MyLib/Debug/MyLib.dll",Calling-  
Convention=CallingConvention.StdCall )>]  
- extern IntPtr MakeOne();
```

```
val MakeOne : unit -> IntPtr
```

Now we make an instance of the COM object.

```
> let x = MakeOne();;  
val x : IntPtr
```

Next we call the method we were interested in. Note that it calls QueryInterface on our object, probing it to fetch the IUnknown. There is a net gain of one in the reference count when the method is called.

```
> let p1 = Marshal.GetObjectForIUnknown x;;
```

```
val p1 : obj  
Returning IUnknown refCount: 2  
Addref: 3  
Addref: 4  
Release: 3  
Release: 2
```

If we call it again, the object has its QueryInterface called followed by a Release, giving no net gain in the reference count.

```
> let p2 = Marshal.GetObjectForIUnknown x;;
val p2 : obj
Returning IUnknown refCount: 3
Release: 2
```

The .NET object representing COM instance is the same from the two calls.

```
> System.Object.ReferenceEquals(p1,p2);;
val it : bool = true
```

We can now release the original IntPtr. This leaves the reference count at 1, the outstanding reference being that from the .NET wrapper object.

```
> Marshal.Release x;;
Release: 1
val it : int = 1
```

We can now release the wrapper object. This has a reference count itself and it is when this reference count reaches zero that it releases the reference on the underlying COM object.

```
> Marshal.ReleaseComObject p1;;
val it : int = 1
```

```
> Marshal.ReleaseComObject p2;;
Release: 0
DEAD
```

```
val it : int = 0
```

This model for interface based programming impressed me deeply when I first came across it around 1996. The ability for objects to be proxied allowed objects to be transparently (in some sense) remoted by infrastructure like that of DCOM and also allowed objects to be made transactional and just-in-time activated by infrastructure such as that of [4]MTS.

1. <http://www.red-gate.com/>
  2. <http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.marshal.getobjectforiunknown.aspx>
  3. [http://msdn.microsoft.com/en-us/library/system.intptr\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/system.intptr(VS.71).aspx)
  4. [http://en.wikipedia.org/wiki/Microsoft\\_Transaction\\_Server](http://en.wikipedia.org/wiki/Microsoft_Transaction_Server)
- 

### **3.3.5 That might be true (2009-03-13 20:34)**

[1]Understanding probability: Change Rules in Everyday Life by Henk Tijms

In my constant quest to improve my hit rate for the [2]IBM Ponder challenge I thought it was time to revise some probability theory. This book is truely outstanding. Its first section covers a range of topics by way of solving a large number of problems and emphasising computer simulation as a means of getting a feeling for what is going on. In the second section, it goes back through the topics and handles them more rigourously. I enjoyed the coverage of

probability distributions. Great explanations and a great text.

- 
1. [http://www.amazon.co.uk/Understanding-Probability-Chance-Rules-Everyday/dp/0521701724/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1236975796&sr=1-1](http://www.amazon.co.uk/Understanding-Probability-Chance-Rules-Everyday/dp/0521701724/ref=sr_1_1?ie=UTF8&s=books&qid=1236975796&sr=1-1)
  2. [http://domino.research.ibm.com/Comm/wwwr\\_ponder.nsf/pages/index.html](http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/pages/index.html)

### **3.3.6 It might be secure (2009-03-13 20:37)**

[1]Programming Windows Security by Keith Brown

This is fairly old book now, it was published in 2000, but it covers a range of security topics very well. Of particular interest to me were the sections on Windows security by way of process and login tokens, object security by way of DACLs, network security by Kerberos and the role based security of COM+. All topics were explained clearly with numerous code examples (in C) giving extra depth. I will definitely be reading this book again.

- 
1. [http://www.amazon.co.uk/Programming-Windows-Security-Developers-DevelopMentor/dp/0201604426/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1236976096&sr=1-1](http://www.amazon.co.uk/Programming-Windows-Security-Developers-DevelopMentor/dp/0201604426/ref=sr_1_1?ie=UTF8&s=books&qid=1236976096&sr=1-1)

### **3.3.7 Prove it (2009-03-16 10:07)**

I spent a little time at the weekend experimenting with securing a remoting connection. A secure mode was added to remoting in .NET version 2, but I have not used it in the past. Like most exploratory scenarios, it was great to be able to use F # to explore the solution dynamically.

The first thing to do was to build an interface dll, which I wrote in C # and compiled into an assembly dll WorkerInterface.dll.

```

namespace WorkerInterface
{
    public interface IWorker
    {
        void DoWork();
    }
}

```

It was then easy to write a server in the interactive F # console.

Import the interface definition

```
#I "C:UsersCliveDesktopWork";;
#r "WorkerInterface.dll";;
```

Define an implementation object. We'll register this as a Singleton so we need the override on InitializeLifetimeService to prevent it timing out if no calls are made to it within the lease timeout interval. We print out the identity as this will be passed when we start using a secure connection later.

```

type MyImpl() =
    class
        inherit System.MarshalByRefObject() as base
        do printfn "New instance made"
        interface WorkerInterface.IWorker with
            member self.DoWork () =
                printfn "Method called %s" System.Threading.Thread.CurrentPrincipal .Identity.Name
            end
            override this.InitializeLifetimeService() = null
        end;;
    
```

We load a small amount of configuration from a file Config1.txt containing

```

<configuration>
    <system.runtime.remoting>
        <application>
            <channels>
                <channel ref="tcp" port="8080" />
            </channels>
        </application>
    </system.runtime.remoting>
</configuration>

```

```
System.Runtime.Remoting.RemotingConfiguration.Configure("C:UsersCliveDesko      pWorkC-
onfig1.txt", false);;
```

We can then make an instance of our implementation object which we can register with the

remoting system.

```
let target = new MyImpl();;
System.Runtime.Remoting.RemotingServices.Marshal (target, "target.rem");;
```

In another F# interactive console we can exercise this server.

```
#I "C:\Users\Clive\Desktop\Work";;
#r "WorkerInterface.dll";;
let obj = System.Activator.GetObject(typeof<WorkerInterface.IWorker>,
"tcp://localhost:8080/target.rem");
let typedObj = ((box obj) :?> WorkerInterface.IWorker);
typedObj.DoWork();;
```

Now, suppose we want to encrypt and sign the message as it passes across the network. We may also want to identify the caller.

On the server side, we can turn this on by simply calling

```
System.Runtime.Remoting.RemotingConfiguration.Configure("C:\Users\Clive\Desktop\Work\config1.txt", true);;
```

passing the second argument as true instead of false.

At this point if we re-run the same client sequence as above, the code hangs while making the call to DoWork. This is apparently because the server is now trying to do the handshake for securing the channel and the client isn't able to respond.

The obvious fix seemed to be to

```
System.Runtime.Remoting.RemotingConfiguration.Configure(... some config file..., true);;
```

with the true parameter telling the system to make things secure. What confused me initially was that the configuration file didn't contain a system.runtime.remoting element which caused the Configure call to be ignored. Using a configuration file of the form,

```
<configuration>
<system.runtime.remoting>
<application>
<channels>
<channel ref="tcp" />
</channels>
</application>
</system.runtime.remoting>
```

```
</configuration>
```

was successful, and the server now displays.

Method called PHILIP-J-FRYClive

The identity of the caller has been passed and made available to the server code.

Using the interactive REPL, it is easy to navigate around the remoting structures to check that the underlying stream is indeed secure.

```
let getPrivateField (obj : System.Object) name =
let flags = System.Reflection.BindingFlags.NonPublic ||| System.Reflection.BindingFlags.Instance
let fieldInfo = obj.GetType().GetField(name, flags)
fieldInfo.GetValue obj
let proxy = System.Runtime.Remoting.RemotingServices.GetRealProxy(typedObj);;
proxy.GetType().Name;;
let flags = System.Reflection.BindingFlags.NonPublic ||| System.Reflection.BindingFlags.Instance
||| System.Reflection.BindingFlags.FlattenHierarchy ||| System.Reflection.BindingFlags.GetProperty;;
proxy.GetType().GetProperty("IdentityObject", flags);;
let identity = it.GetValue(proxy, null);;
identity.GetType().GetProperty("ChannelSink", flags);;
let sink1 = it.GetValue(identity, null);;
sink1.GetType().GetProperty("NextChannelSink");;
let transport = it.GetValue(sink1, null);;
getPrivateField transport " _channel" ;;
```

I love the way that F # allows me to easily explore what is going on, without the need to use a debugger and gives me the ability to easily write code to aid the exploration.

---

## 3.4 April

### 3.4.1 Bring out your dead (2009-04-14 17:16)

I recently gave a talk at work on garbage collection, a subject in which I have been interested for a long time. The slides are available [1]here and there are some examples in C # [2]here which were intended to highlight some of issues that I talked about in the slides.

1. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Public/GarbageCollection.pptx>
  2. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Public/GarbageCollectionExamples.zip>
- 

### **3.4.2 Caught up in the web (2009-04-22 17:12)**

At the weekend I attended [1]WebDD down at Microsoft Reading. Like all of the community organised events, there was an interesting range of talks. Some simply brought together content that I had seen already from blogs. However, there were three talks that I found very useful.

ASP.NET MVC Best Practices by [2]Sebastien Lambla was an energy filled presentation. The presenter showed how the default MVC project structure could be modified to make more use of dependency injection. Even though his Mac crashed a few minutes into the presentation, corrupting the disc and preventing him from writing any more code in front of us, Sebastien's presentation highlighted the benefits of MVC - in particular, the testability and control of the rendered HTML. In particular he seemed to be advocating the use of a strongly typed view model, an idea prevalent in WPF's MVVM pattern. He promised to write up some of the presentation in a series of blog posts in the coming weeks.

[3]Barry Dorrans' presentation on the top ten web site vulnerabilities was also very good, as was [4]Mike Taulty's demonstration of Silverlight 3. The latter's presentation was the first time in ages that I have seen a demo fail at one of these events - when he tried to make his Silverlight demo work out of browser, it failed to install on the computer, though he had it all working again by lunchtime. Phil Purglove's presentation on ASP.NET caching also covered options I hadn't seen before.

Thank you to the organisers for setting this up.

1. <http://developerdeveloperdeveloper.com/webdd09/Schedule.aspx>
  2. <http://serialseb.blogspot.com/>
  3. <http://idunno.org/>
  4. <http://mtaulty.com/>
- 

### **3.4.3 Come at me with the banana (2009-04-23 07:27)**

I forgot to mention the excellent [1]Python Coding Dojo that Garry Bodsworth ran a month or so ago. It was an interesting day where I learned a lot. I liked the style, where the attendees could listen to his presentation and try the examples on their laptops at the same time. This raised lots of interesting questions that kept things nice and interactive. Thanks to Garry for the six (or so) hours of continuous presentation.

1. <http://blog.programmerslog.com/?p=352>
- 

### **3.4.4 There's something in my eye (2009-04-24 16:37)**

[1]Beautiful Architecture: Leading Thinkers Reveal the Hidden Beauty in Software Design by Diomidis Spinellis and Georgios Gousios

This book consists of a collection of fourteen essays on various aspects of computer architecture, ranging from system architecture to computer languages to enterprise architecture. The range of essays is impressive and the content is quite a good read.

I particularly enjoyed the chapters on the beauty of the architecture of the [2]Smalltalk system with its hierarchy of classes and metaclasses, the [3]Xen hypervisor, the [4]Jikes runtime for Java and [5]JPC, a virtualization system for x86 computers written in Java.

The Jikes chapter reminded me how much I enjoyed working on Lisp systems in the past, where the runtime was also written in the language that it implemented (making it metacircular). This meant that compiler optimisations affected both the user programs compiled for the system and the performance of the runtime itself. In addition, the system used the heap for holding the code vectors allowing the garbage collector to collect the space if the code were not longer accessible.

Most of the other essays were also interesting, covering general architecture and system design and a good article that explained why the EMACS extensibility model has been so successful.

1. [http://www.amazon.co.uk/Beautiful-Architecture-Leading-Thinkers-Software/dp/059651798X/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1240589509&sr=8-1](http://www.amazon.co.uk/Beautiful-Architecture-Leading-Thinkers-Software/dp/059651798X/ref=sr_1_1?ie=UTF8&s=books&qid=1240589509&sr=8-1)
  2. <http://www.squeak.org/>
  3. <http://www.xen.org/>
  4. <http://jikes.sourceforge.net/>
  5. <http://www-jpc.physics.ox.ac.uk/>
- 

### **3.4.5 Greengrocers know best! (2009-04-25 21:46)**

[1]Kepler's Conjecture by George G Szpiro

I just finished rereading this book. It is the story of the Kepler conjecture, and the two dimensional equivalent, from its formulation in 1611 to the solution by Thomas Hales in 1998. A great read which is full of interesting details.

1. [http://www.amazon.co.uk/Keplers-Conjecture-Greatest-History-Problems/dp/0471086010/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1240695719&sr=8-1](http://www.amazon.co.uk/Keplers-Conjecture-Greatest-History-Problems/dp/0471086010/ref=sr_1_1?ie=UTF8&s=books&qid=1240695719&sr=8-1)

---

### 3.4.6 Take your time (2009-04-25 21:56)

There was a little discussion of the F # async form at work, so I thought I'd set up a simple example demonstrating it. This pattern is a really neat way to get blocks of a computation executed on the threadpool avoiding the need to permanently allocate a thread to the computation.

We can get something to execute on the thread pool using a timer callback, so this code causes the string "hello" to be printed after a ten second delay.

```
let func = new System.Threading.TimerCallback(fun _ -> printfn "hello");;
```

```
new System.Threading.Timer(func, null, 10000, -1);;
```

We can define a type of object that is responsible for firing an operation after a ten second delay. We keep the timers in a list to avoid a garbage collection disposing them and hence stopping them from firing. We use Async.Primitive to build up an async object; this gives us a continuation to call in the success case and another to call in the failure case. The TimedIncrement method simply increments the incoming value, but does so after waiting for ten seconds. By using a Timer instance, we effectively release the current thread and then a new thread is grabbed from the threadpool by the Timer when it fires.

```
type TimedAction() =
let mutable actions = []
```

```
member this.TimedIncrement(x) =  
    Async.Primitive (fun (cont,econt) ->  
        let callback = new System.Threading.TimerCallback(fun _ -> cont (x+1))  
        let next = new System.Threading.Timer(callback , null, 10000, -1)  
        actions <- next :: actions  
    ())
```

We can use a function to get the current thread id

```
let getId () = System.Threading.Thread.CurrentThread.ManagedThreadId;;
```

We can then use this in a workflow.

```
let worker (delay) =  
    async { let timer = new TimedAction()  
        do System.Threading.Thread.Sleep(delay * 1000 : int)  
        let! result1 = timer.TimedIncrement(1)  
        do printfn "Running on thread %d" (getId())  
        let! result2 = timer.TimedIncrement(result1)  
        do printfn "Then on thread %d" (getId())  
        return result2  
    }
```

Running some simple examples we can see how the threadpool threads are being reused.

```
> Async.Run(worker(0));;
Running on thread 10
Then on thread 10
val it : int = 3
> Async.Run(Async.Parallel [ worker(0); worker(1); worker(2) ]);;
Running on thread 7
Running on thread 10
Running on thread 7
Then on thread 7
Then on thread 7
Then on thread 7
val it : int array = [|3; 3; 3|]
```

---

### 3.4.7 A couple of good papers (2009-04-26 20:01)

There are a couple of papers that I have read recently that I thought were quite good and which I meant to blog about some time ago.

[1]Automated Verification of Practical Garbage Collectors by Chris Hawblitzel and Erez Petrank presents two mechanically verified garbage collectors that have been used in the Singularity project to run real world C # benchmarks at rates competitive with the Bartok collectors. Seeing the formulation of correctness for such a collector was informative and it was interesting to see how pre- and post-conditions, invariants and assertions on x86 code can then be mechanically verified.

[2]Runtime Support for Multicore Haskell by Simon Marlow, Simon Peyton Jones and Satnam Singh describes the implementation of Haskell on a multiprocessor machine, detailing the various design tradeoffs and giving various measurements that informed the decisions that they made.

1. <http://research.microsoft.com/pubs/74285/popl09-vgc-hawblitzel-petrank.pdf>
2. <http://research.microsoft.com/en-us/um/people/simonpj/papers/parallel/multicore-ghc.pdf>

---

## 3.5 May

### 3.5.1 Next please (2009-05-10 03:53)

My first Red Gate project is now completed. [1]Exchange Server Archiver is now available for download and purchase. It's been great working with the [2]postmasters on this project and hopefully I'll get to work with them again in the future.

1. <http://www.red-gate.com/products/Exchange/index.htm>
  2. <http://www.simple-talk.com/exchange/editors-corner/the-postmasters/>
- 

### 3.5.2 Javascript is Bak (2009-05-10 22:55)

There are some great recent videos of sessions at the recent [1]Lang.Net symposium. Out of the few that I have viewed so far, the most interesting was certainly the [2]talk and subsequent [3]Channel9 interview with Lars Bak, the designer of the V8 Javascript engine that is used in Chrome. In order to make it easy to do the usual tricks for implementing PICs (polymorphic inline caches), the javascript objects, which are best thought of as property bags, get grouped into categories (which he called classes) according to their named properties maintained by the order in which they were added.

Objects are represented in the heap as a simple three element structure. The first slot points to the type (the class object), the second to a vector of slot values, and the third contains indexed properties. This looks very much like the wrapper notion that I have seen before in [4]Portable Common Loops. The system then maintains state transition graphs to allow the type to change as properties are dynamically added to the object. For example, if you start with an object with no properties its wrapper may point to class C\_0. If you then set this.x (in the constructor) the system would generate a new class C\_1 and record that the "adding a property x" transition from C\_0 goes to C\_1. If you then set this.y, another transition would be recorded from C\_1 to C\_2. Any subsequent objects that have x and y properties set in

this order would then end up having the same class C\_2. This transition graph is maintained on the heap, so can be collected when segments of it are no longer accessible. The classes allow the generated code to quickly validate the shape of the object on which the code is acting. When a property is deleted, the system stops tracking the type and simply uses a general code generation method.

As Lars Bak makes clear, the aim is to keep things really simple, making it easy to optimise, and to hit the general cases. As delete is fairly infrequent in javascript code, there is no intention to optimise it. Small integers in the system can be represented using an inline tag avoiding the need to place them on the heap - this is the same trick Lisp systems use for representing fixnums. The V8 system is impressively fast when compared against other implementations, and there are many additional optimisations that are being considered.

There were a few other functional programming related talks that I liked. [5]Amanda Launcher did a quick 15 minute presentation on rewriting the rules in an insurance application in F # and how concurrency in the calculation could easily be exploited using async and lazy workflow expressions. Two other talks focussed on reactive programming. [6]Eric Meijer gave a great talk on a different way to look at Enumeration and Tomas Petricek talked about adding [7]reactive patterns to F #. These are currently implemented using the [8]joins library.

1. <http://www.langnetsymposium.com/2009/agenda.aspx>
2. <http://www.langnetsymposium.com/2009/talks/18-LarsBak-JavaScript.html>
3. <http://channel9.msdn.com/shows/Going+Deep/Expert-to-Expert-Erik-Meijer-and-Lars-Bak-Inside-V8-A-Javascript-Virtual-Machine/>
4. [http://en.wikipedia.org/wiki/Common\\_Lisp\\_Object\\_System](http://en.wikipedia.org/wiki/Common_Lisp_Object_System)
5. <http://www.langnetsymposium.com/2009/talks/21-AmanderLauter-FSharpConcurrency.html>
6. <http://www.langnetsymposium.com/2009/talks/23-ErikMeijer-LiveLabsReactiveFramework.html>
7. <http://www.langnetsymposium.com/2009/talks/22-TomasPetricek-Reactive.html>
8. <http://clivetong.spaces.live.com/blog/cns!3F21DF299C355E7F!217.entry>

---

### 3.5.3 I'll return your call later (2009-05-20 21:20)

It was a while ago when I came across the paper [1]a technique for implementing first-class continuations in languages that do not support stack manipulation and inspection. The tech-

nique is rather clever, using a source to source transformation to produce a set of functions which can use exception handling and chaining to capture the active stack frames. These frames can then be restarted at a later stage. The technique has been tried out in various implementations of Scheme.

The web page mentioned above details an implementation in C#. However, I think that the verbosity of the C# implementation hides some of the essential details and therefore want to walk through an example using F# as the implementation language. I should point out that we need to use the obj (System.Object) type in some places so we don't gain all of the benefits of the type checking that F# carries out. However, the conciseness of the implementation more than makes up for this.

We need an example function to work through. Having been recently reminded of the classic co-routine example of matching the fringes of two trees, we'll work on an implementation of a function which takes a tree and returns an element from the fringe together with a continuation which will return the next element.

```
type 'a Tree =
| Leaf of 'a
| Node of 'a Tree * 'a Tree

let rec getNext tree =
match tree with
| Leaf x -> callcc (fun k -> Some(x, k))
| Node (l,r) ->
match getNext l with
| Some _ as result -> result
| None -> getNext r
```

The idea is that we'll define a suitable callcc, which stands for call with current continuation. If we called this function on Node(Leaf(1), Leaf(2)) we'd expect to get back a pair of 1 and a continuation. We could then pass None into this continuation to get the next result which would be a pair of 2 and another continuation.

The first step in the source to source conversion is to introduce temporaries

```
let rec getNext_an tree =
  match tree with
  | Leaf x -> callcc (fun k -> Some(x, k))
  | Node (l,r) ->
    let temp0 = getNext_an l
    match temp0 with
    | Some _ as result -> result
    | None -> getNext_an r
```

and then to convert into tail recursive functions

```
let rec getNext_0 tree =
  match tree with
  | Leaf x -> callcc (fun k -> Some(x,k))
  | Node (l,r) ->
    let temp0 = getNext_0 l
    getNext_1 (temp0, r)
```

```
and getNext_1 (temp0, r) =
  match temp0 with
  | Some _ as result -> result
  | None -> getNext_0 r
```

We can now start defining some of the machinery. The representation of a continuation frame which supports an Invoke function to invoke the continuation. This object contains a list of frames further down the stack, allowing these frames to be shared if the continuation is captured again.

```
[<AbstractClass>]
type ContinuationFrame() =
```

```

let mutable continuation : ContinuationFrame list = []
member v.Continuation with get() = continuation and set(value) = continuation <- value
abstract member Invoke : obj -> obj

```

We now define the exception which will be passed up the stack. It will accumulate the frames as we go.

```

type ContinuationData() =
let mutable new _frames: ContinuationFrame list = []
let mutable old _frames : ContinuationFrame list = []
member a.NewFrames with get() = new _frames and set(v) = new _frames <- v
member b.OldFrames with get() = old _frames and set(w) = old _frames <- w

```

exception SaveContinuation of ContinuationData

We can extend the continuation frame type with a Reload method which will be used for re-building the stack. Type extension in F # is great for doing this.

```

type ContinuationFrame with
member v.Reload(frames _above : ContinuationFrame list, restart _value) =
let continue _value =
if frames _above = []
then restart _value
else List.hd(frames _above).Reload(List.tl frames _above, restart _value)
try
v.Invoke(continue _value)
with
SaveContinuation data as ex ->
data.OldFrames <- v.Continuation
raise ex

```

We can now define a type representing a continuation.

```
type Continuation(new_frames, old_frames) =  
let mutable frames = []  
let mutable new_frames Mutable = new_frames  
do frames <- old_frames  
do while (new_frames Mutable <> []) do  
let new_frame : ContinuationFrame = List.hd new_frames Mutable  
new_frames Mutable <- List.tl new_frames Mutable  
new_frame.Continuation <- frames  
frames <- new_frame :: frames  
member v.Frames () = frames  
member v.Reload (restart_value) =  
let rev = List.rev frames  
List.hd(rev).Reload(List.tl rev, restart_value)
```

And the structures needed for getting the continuation capture started. We'll need to run all computation inside the thunk of EstablishInitialContinuation. This function adds an initial exception handler onto the stack which can capture the exception which is accumulating the stack frames, and which can then reset the stack and continue.

```
type CWCC_frame0 (receiver : Continuation -> obj) =  
inherit ContinuationFrame()  
override v.Invoke return_value =  
let asCont = (return_value :?> Continuation)  
receiver (Continuation(List.rev(List.tl(asCont.Frames()))),[]))
```

```
let BeginUnwind() =  
raise (SaveContinuation(ContinuationData()))
```

```
let CWCC receiver =  
try
```

```
BeginUnwind()
with
SaveContinuation data as ex ->
data.NewFrames <- (CWCC_frame0(receiver) :> ContinuationFrame) :: data.NewFrames
raise ex
```

exception WithInitialContinuation of (unit -> obj)

```
let InitialContinuationAux thunk =
try
thunk()
with
SaveContinuation data as ex ->
let k = Continuation(data.NewFrames, data.OldFrames)
raise (WithInitialContinuation(fun () -> k.Reload(k)))
```

```
let EstablishInitialContinuation th =
let mutable thunk = th
let mutable again = true
let mutable result = 0 :> obj
while again do
again <- false
try
result <- InitialContinuationAux thunk
with
WithInitialContinuation new _thunk ->
thunk <- new _thunk
again <- true
result
```

For our example function, we now need to define some specialized frames, and we need to capture the relevant exceptions in the body of the function.

```

type GetNext _f0<'a>(r : 'a Tree, f : ('a * Continuation) option * 'a Tree -> ('a * Continuation) option) =
inherit ContinuationFrame()
override v.Invoke continue _value =
let value = ((box continue _value) :?> ('a * Continuation) option)
f(value, r) :> obj

let rec getNext _e0 tree =
match tree with
| Leaf x ->
CWCC (fun k -> Some(x,k) :> obj)
| Node (l,r) ->
let mutable temp0 = None
try
temp0 <- getNext _e0 l
with
SaveContinuation data as ex ->
data.NewFrames <- (GetNext _f0(r, getNext _e1) :> ContinuationFrame) :: data.NewFrames
raise ex
getNext _e1 (temp0, r)

and getNext _e1 (temp0, r) =
match temp0 with
| Some _ as result ->
result
| None ->
getNext _e0 r

```

We can now take an example tree.

```
let tree = Node(Leaf(0), Node(Node(Leaf 1, Leaf 2), Leaf 3))
```

And get the first result.

```
let res0 = EstablishInitialContinuation (fun () -> getNext _e0 tree :> obj)
```

We can now get hold of the continuation, and return None from it. This will cause the code to find the next element of the fringe.

```
let cont0 = (fun (Some ( _,x)) -> x) (res0 :?> (int * Continuation) option)
```

```
let res1 = EstablishInitialContinuation(fun () -> cont0.Reload(None));;
```

We can then do this several more times.

```
let cont1 = (fun (Some ( _,x)) -> x) (res1 :?> (int * Continuation) option)
```

```
let res2 = EstablishInitialContinuation(fun () -> cont1.Reload(None));;
let cont2 = (fun (Some ( _,x)) -> x) (res2 :?> (int * Continuation) option)
```

```
let res3 = EstablishInitialContinuation(fun () -> cont2.Reload(None));;
let cont3 = (fun (Some ( _,x)) -> x) (res3 :?> (int * Continuation) option)
```

The web page goes into the details of each of the classes and functions.

As a quick aside, it is really neat that F # now comes as part of the beta for Visual Studio 2010, making it feel like a first class .NET language.

1. <http://eval.apply.googlepages.com/stackhack4.html>

---

### 3.5.4 Play it again (2009-05-30 22:21)

I've always thought that the [1]Actor model of concurrency has good potential for the multi-core future. It was therefore interesting to see all of the recent coverage of [2]Axum (nee Maestro), an Actor based concurrency framework currently being implemented as an incubation project at Microsoft. There was an interesting, and quick, [3]overview talk at the recent [4]Lang.Net symposium.

Basically, actors are isolated into domains, where a domain is a boundary that protects shared state. The actors are tagged as readers and writers by the way that they access this shared state, and the system ensures that only a single writer or multiple readers may execute at the same time within the same domain. Communication happens between the domains using channels, which are a lot like C # interfaces in the that they control the values and messages that may be communicated. They are stronger than interfaces in that they may contain state machine definitions which reflect the patterns of messages that may pass across the channel. The messages may carry values which are passed by value, and the system generates the marshalling code for passing the data, avoiding the need to rely on user defined serialization functions. The system guarantee that only a single writer will be running within the same domain makes it a lot easier to avoid the need for locking within the application code, and the channel patterns make it a lot easier to ensure that the access patterns avoid deadlock in the application code.

Axum is currently implemented as a new language that looks like C# and which runs on top of the [5]CCR which in turn sits on the CLR.

There is a [6]Channel9 interview with the Maestro team, a [7]Channel9 Axum tutorial, and an [8]interview with some of the Axum team on .NET rocks.

1. <http://clivetong.spaces.live.com/blog/cns!3F21DF299C355E7F!163.entry>
  2. <http://blogs.msdn.com/maestroteam/>
  3. <http://www.langnetsymposium.com/2009/talks/32-JoshuaPhillips-Axum.html>
  4. <http://www.langnetsymposium.com/>
  5. <http://msdn.microsoft.com/en-us/library/bb648752.aspx>
  6. <http://channel9.msdn.com/shows/Going+Deep/Maestro-A-Managed-Domain-Specific-Language-For-Concurrent-Programming/>
  7. <http://channel9.msdn.com/posts/Charles/Building-your-first-Axum-application/>
  8. <http://www.dotnetrocks.com/default.aspx?showNum=449>
- 

### **3.5.5 It's a matter of trust (2009-05-30 22:35)**

I've just finished reading [1]Professional ASP.NET 2.0 Security, Membership and Role Management by Stefan Schackow

For me this was a book of two halves.

The first half covers the processing of an ASP.NET page on IIS6 from a security viewpoint, with chapters covering the dispatch of the request into the ASP.NET pipeline and how the thread identity is changed in the process, the code access security settings and how they affect the code of the ASP.NET page, how forms authentication is handled and how it can be integrated into legacy ASP applications, how session state is handled and the security implications of the way pages are compiled. This half has great descriptions on how HTTP modules handle the various bits and pieces during the processing of the ASP.NET pipeline, and explains everything really well.

The second half covers the provider model with an introductory chapter explaining the design patterns that it uses, and then moves on to in depth descriptions of the membership and role manager providers and the SQL and Active Directory versions that come as part of the framework. I must admit that when I first read this book several years ago (on a beach in Wales) I only read the first half of the book, but on the second read I found this second half was really quite interesting.

An interesting book, though I would like to see an IIS7 version to see how things have changed over the years.

1. [http://www.amazon.co.uk/Professional-ASP-NET-Security-Membership-Management/dp/0764596985/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1243722205&sr=8-1](http://www.amazon.co.uk/Professional-ASP-NET-Security-Membership-Management/dp/0764596985/ref=sr_1_1?ie=UTF8&s=books&qid=1243722205&sr=8-1)
- 

## 3.6 June

### 3.6.1 What category would you place that in? (2009-06-11 19:00)

I've decided to spent some time reading up on denotational semantics and concurrency and as a prelude I thought it would be a great idea to read up on some category theory. Despite studying some algebraic topology at college, the course didn't cover any category theory. I picked up the nice and slim [1]Basic Category Theory For Computer Scientists by Benjamin Pierce which turned out to be a good introduction to the subject. Of course, the only way to get really familiar with the material is to work through the exercises, and as part of that I really needed a way to look at some examples.

The obvious answer is to use F # to look at some simple categories. The REPL (read-eval-print loop) makes it easy to take arrows from the category and study their properties in order to

find counter-examples. This should be not surprise - ML was originally invented as the implementation language of the LCF theorem proving system. Theorems could be represented as instances of abstract datatypes within the language, and the datatypes could expose functionality that allowed only valid theorems to be constructed. There is a book which I have just started reading, [2]Computational Category Theory for which the source is [3]available here, which covers some implementation of category theory in ML.

A category is basically a collection of objects and arrows with a means for finding the start and end objects of a given arrow, together with a means for getting the identity arrow corresponding to any object and a means of composing the arrows. Of course, there are constraints on these various functions which we cannot express in the type system.

```
type Category<'arrow, 'ob> =
{ Start : 'arrow -> 'ob; End : 'arrow -> 'ob; Identity : 'ob -> 'arrow; Compose : 'arrow -> 'arrow -> 'arrow }
```

We can then define a type representing an arrow, and define a function that generates one of the standard finite categories, that of a preorder on a list of elements where each element is < than the elements to the right of it in a list.

```
type Arrow<'a> = { From : 'a; To : 'a }
```

```
let GenerateOrderCategory elements =
let startfn x = x.From
let endfn x = x.To
let check x =
if not (List.mem x elements)
then failwith "Invalid element for generating identity arrow"
else x
```

```

let identity x =
{ From = check x; To = x }
let compose arrow1 arrow2 =
if check(arrow1.To) = arrow2.From
then { From = check arrow1.From; To= check arrow2.To }
else failwith "Invalid composition"
{ Start = startfn; End=endfn; Identity= identity; Compose=compose }

```

```

let OrderCategoryArrows elements =
let rec generate a bsofar =
if b = []
then if a = []
then sofar
else generate (List.tl a) (List.tl a) sofar
else generate a (List.tl b) ( { From = List.hd a; To = List.hd b } :: sofar)
Set.of _seq(generate elements elements [])

```

A finite category can be expressed as a collection of functions representing the category, together with a Set of the arrows.

```

type FiniteCategory<'arrow, 'ob> =
{ Category : Category<'arrow, 'ob>; Arrows : Set<'arrow> }

```

We can then define the finite category over a finite set of elements.

```

let OrderCategory elements =
{ Category = GenerateOrderCategory elements; Arrows = OrderCategoryArrows elements }

```

```
let cat1 = OrderCategory [1]
```

```
let cat2 = OrderCategory [1;2]
```

```
let cat3 = OrderCategory [1;2;3]
```

We cannot express the constraints on the functions representing the category in the type system, but for a finite category we can loop across all of the arrows to validate that the rules hold.

```
let canCompose category arrow1 arrow2 =
category.End arrow1 = category.Start arrow2
```

```
let CheckFiniteCategory finiteCategory =
let arrows = finiteCategory.Arrows
let category = finiteCategory.Category
let mutable objs = Set.empty
for arrow in arrows do objs <- Set.add (category.Start arrow) objs
for arrow in arrows do objs <- Set.add (category.End arrow) objs
// Identities
for ob in objs do
let id = category.Identity ob
if not (Set.mem id arrows & & category.Start id = ob & & category.End id = ob)
then failwith "No identity"
// Well defined
for a1 in arrows do
for a2 in arrows do
if canCompose category a1 a2
then
let result = category.Compose a1 a2
if not (Set.mem result arrows & & category.Start result = category.Start a1 & & category.End
```

```

result = category.End a2)
then failwith "Invalid composition"
// Associative
for a1 in arrows do
for a2 in arrows do
for a3 in arrows do
if canCompose category a1 a2 & & canCompose category a2 a3
then
let r1 = category.Compose (category.Compose a1 a2) a3
let r2 = category.Compose a1 (category.Compose a2 a3)
if r1 <> r2 then failwith "Not associative"
// Check identity composes with f to give f
for ob in objs do
let id = category.Identity ob
for arrow in arrows do
if canCompose category id arrow & & category.Compose id arrow <> arrow
then failwith "Composing identity and arrow"
if canCompose category arrow id & & category.Compose arrow id <> arrow
then failwith "Composing arrow and identity"

```

We can then check the categories we have defined.

```

CheckFiniteCategory cat1
CheckFiniteCategory cat2
CheckFiniteCategory cat3

```

For the finite case, we can validate that certain arrows are epi- and mono- morphisms.

```

let isMonomorphism arrow finiteCategory =
let arrows = finiteCategory.Arrows
let category = finiteCategory.Category
Seq.for_all
(fun g ->

```

```

Seq.for_all
(fun h ->
if category.Start g = category.Start h & & category.End g = category.End h
then if canCompose category g arrow
then if category.Compose g arrow = category.Compose h arrow
then g = h
else false
else true
else true)
arrows)
arrows

```

```

let isEpimorphism arrow finiteCategory =
let arrows = finiteCategory.Arrows
let category = finiteCategory.Category
Seq.for_all
(fun g ->
Seq.for_all
(fun h ->
if category.Start g = category.Start h & & category.End g = category.End h
then if canCompose category arrow g
then if category.Compose arrow g = category.Compose arrow h
then g = h
else false
else true
else true)
arrows)
arrows

```

```

isMonomorphism {From=1;To=3 } cat3
isEpimorphism {From=1;To=3 } cat3;;

```

Another standard example is that of representing a group using arrows and a single object. We can construct it like so.

```
type GroupElement = Element
```

```
let CycleGroupProduct n =
let startfn _ = Element
let endfn _ = Element
let identity Element = (0,0)
let compose (a,b) (c,d) = ((a + c) % n, (b+d) % n)
let category =
{ Start=startfn; End=endfn; Identity=identity; Compose=compose }
let arrows =
seq {
for i in 0 .. n-1 do
for j in 0 .. n-1 do
yield (i,j)
}
{ Category = category; Arrows = Set.of_seq arrows }
```

We can then define the Klein four group as a category and validate the representation.

```
let V4 = CycleGroupProduct 2
```

```
CheckFiniteCategory V4
```

```
isMonomorphism (0,0) V4
isMonomorphism (1,0) V4
isMonomorphism (1,1) V4
```

There's much more to do, but at least we have a start.

- 
1. [http://www.amazon.co.uk/Category-Computer-Scientists-Foundations-computing/dp/0262660717/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1244745401&sr=8-1](http://www.amazon.co.uk/Category-Computer-Scientists-Foundations-computing/dp/0262660717/ref=sr_1_1?ie=UTF8&s=books&qid=1244745401&sr=8-1)
  2. [http://www.amazon.co.uk/Computational-Category-Prentice-International-Computing/dp/0131627368/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1244746068&sr=8-1](http://www.amazon.co.uk/Computational-Category-Prentice-International-Computing/dp/0131627368/ref=sr_1_1?ie=UTF8&s=books&qid=1244746068&sr=8-1)
  3. <http://www.cs.man.ac.uk/~david/categories/book/book.pdf>
- 

### **3.6.2 Divide (2009-06-11 19:25)**

The other day, I was thinking through some of the interesting compiler tricks that I came across while working on Lisp compilers in the 1990s. One interesting technique was to convert division by a small constant into multiplication and shift operations which could generate a code sequence that was much faster. Searching using google, I quickly found a site that [1]describes the technique.

- 
1. <http://www.cs.uiowa.edu/~jones/bcd/divide.html>
- 

### **3.6.3 NxtGen Fest 09 (2009-06-15 11:24)**

Last Friday I attended the [1]NxtGen Fest 09 event at Microsoft Research, Cambridge. This consisted of a day of generally interesting talks on a variety of topics.

The keynote and the first talk were on the subject of machine learning using probabilistic models. Microsoft Research have just released a beta of [2]Infer.NET, a library that lets the user set up a probabilistic model; results can then be fed into the model and the Infer.NET framework will back-propagate the data values to infer more accurate probability distributions. For

a simple example, there's a model for throwing two coins in the documentation [3]here with an explanation [4]here. This type of technology has been used in TrueSkill for ranking players on XBox.

The next talk was on Silverlight 3. I'd seen this talk already at WebDD, where [5]Mike Taulty presented a list of Silverlight improvements in version 3. The first item on the list was the ability to set an application running out of browser. At WebDD this demo failed and at Fest 09 there seemed to be problems uninstalling the application after it had been installed out of browser.

[6]Dinis Cruz followed up with a great talk on using analysis tools for detecting security problems in web applications. He talked a little about [7]CAT.NET and then went on to demonstrate a tool on which he'd been working, which is part of [8]OunceOpen. As well as static analysis, he was interested in analysing dynamic traces of call paths through the application - to do this he had taken the Microsoft managed debugger sample ([9]Mdbg) and written code on top of that to install breakpoints on the application code and capture the trace. The emphasis was on having rules that detected exploitable pieces of code, say execution of a dynamically generated SQL query in a function that took a string parameter which was concatenated with other strings to make the query, and then analysing the code to find entry points to the application that could pass in a value that found its way to this call site.

[10]Chris Hay did a good talk on the way Silverlight applications can communicate with the host web site. His talk mentioned [11]NPAPI and showed that some of the limitations in the communication from a Silverlight application were down to the need to run on top of this api.

[12]The last talk by Alisson Sol was fantastic. He discussed some of the practices inside Microsoft for categorising and handling bugs. The talk had a few interesting examples of subtle bugs, which would be very tricky to find... this was one motivation for freezing code early and releasing in large beta programs to allow the code to be tested on a wide variety of machines. He also suggested that some technical interviews concentrate on the wrong things - who cares if some whiteboard code contains an off-by-one error. That is the kind of things that testers are going to find. The talk made a lot of interesting observations.

1. <http://www.nxtgenug.net/Fest09/Agenda.aspx>

2. <http://research.microsoft.com/en-us/um/cambridge/projects/infernet/>

3. <http://research.microsoft.com/en-us/um/cambridge/projects/infernet/docs/FSharpTwoCoins.aspx>
  4. <http://research.microsoft.com/en-us/um/cambridge/projects/infernet/docs/TwoCoinTutorial.aspx>
  5. [http://mtaulty.com/communityserver/blogs/mike\\_taultys\\_blog/default.aspx](http://mtaulty.com/communityserver/blogs/mike_taultys_blog/default.aspx)
  6. <http://www.owasp.org/index.php/User:Dinis.cruz>
  7. <http://www.microsoft.com/downloads/details.aspx?FamilyId=0178e2ef-9da8-445e-9348-c93f24cc9f9d&displaylang=en>
  8. <http://www.o2-ounceopen.com/>
  9. <http://www.microsoft.com/downloads/details.aspx?FamilyID=38449a42-6b7a-4e28-80ce-c55645ab1310&displaylang=en>
  10. <http://silverlightuk.blogspot.com/>
  11. <http://en.wikipedia.org/wiki/NPAPI>
  12. [http://www.alissonsol.com/slides/2009-06-12-NxtGenUG\\_Fest\\_09.pdf](http://www.alissonsol.com/slides/2009-06-12-NxtGenUG_Fest_09.pdf)
- 

### 3.6.4 What's next? (2009-06-24 20:44)

[1]Microsoft 2.0: How Microsoft Plans to Stay Relevant in the Post-gates Era by Mary Jo Foley

I wasn't really sure what to make of this book. When it was talking about the history of the company and the characters involved, it was quite interesting. However, at some points it ended up just listing diverse technologies that were predicted to be directions in which Microsoft's many divisions might head in the future. In most cases there wasn't a lot of evidence supplied to back up the claims. There also wasn't a lot of technical depth.

1. [http://www.amazon.co.uk/Microsoft-2-0-Plans-Relevant-Post-gates/dp/0470191384/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1245862266&sr=1-1](http://www.amazon.co.uk/Microsoft-2-0-Plans-Relevant-Post-gates/dp/0470191384/ref=sr_1_1?ie=UTF8&s=books&qid=1245862266&sr=1-1)
- 

### 3.6.5 It's one thing after another (2009-06-24 20:53)

[1]Communicating Sequential Processes by C.A.R.Hoare

In a continuing attempt to improve my knowledge of program semantics, I thought it  
278

would be a good idea to read this book before moving on to the pi-calculus. [2]his book can be freely downloaded and the material is sketched out [3]on the wikipedia.

The book is great. It covers concurrent communicating processes by defining a calculus for expressing their events and potential interactions, together with a rather neat denotational semantics based on traces. The book is well structured, starting simple and introducing more complex ideas as the chapters progress. The discussion of the various operators is really good and there are many examples which help you to get to grips with the material. Also, theorems are proved in some detail, making the book less woolly than some other books on the subject. For example, the author proves that the recursive definitions do have a single solution.

The book covers the difference between the specification of the behaviour of the processes and the definitions of the processes themselves, and discusses how we can prove that a process obeys a specification. The operators of the calculus have example definitions in functional Lisp (Lispkit Lisp) gaining the author bonus points in my eyes.

1. [http://www.amazon.co.uk/Communicating-Sequential-Processes-Prentice-Hall-International/dp/0131532898/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1245862535&sr=1-1](http://www.amazon.co.uk/Communicating-Sequential-Processes-Prentice-Hall-International/dp/0131532898/ref=sr_1_1?ie=UTF8&s=books&qid=1245862535&sr=1-1)
  2. <http://www.usingcsp.com/>
  3. [http://en.wikipedia.org/wiki/Communicating\\_Sequential\\_Processes](http://en.wikipedia.org/wiki/Communicating_Sequential_Processes)
- 

### **3.6.6 What difference does it make? (2009-06-24 20:58)**

[1]Categories For Types by Roy Crole

Having read the [2]Pierce book on category theory, I really wanted to see how category theory could be used for giving a denotational semantics to functional programming languages. This book was a great explanation of how that can be done. It contains introductory chapters on domain theory and category theory and then dives into categorical semantics, including the relationship between Cartesian Closed Categories and the lambda calculus. Some of the details were too much for me, but I do feel that reading the book has given me a good understanding of this branch of computer science. Over the next few weeks, I will be spending time working through the examples.

1. [http://www.amazon.co.uk/Categories-Types-Roy-L-Crole/dp/0521457017/ref=sr\\_1\\_2?ie=UTF8&s=books&qid=1245862972&sr=1-2](http://www.amazon.co.uk/Categories-Types-Roy-L-Crole/dp/0521457017/ref=sr_1_2?ie=UTF8&s=books&qid=1245862972&sr=1-2)
  2. [http://www.amazon.co.uk/Category-Computer-Scientists-Foundations-computing/dp/0262660717/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1245876974&sr=8-1](http://www.amazon.co.uk/Category-Computer-Scientists-Foundations-computing/dp/0262660717/ref=sr_1_1?ie=UTF8&s=books&qid=1245876974&sr=8-1)
- 

### **3.6.7 Don't get in a flap (2009-06-24 21:02)**

A while ago I read a [1]thesis on Flapjax, a system for doing reactive event driven programming in javascript. There's now a [2]channel9 interview on this project which looks really interesting; sadly the interview doesn't cover the project in much detail but steps back to try to see how such technology fits into the bigger picture.

1. <http://www.eecs.berkeley.edu/~lmeyerov/projects/flapjax/thesis8.pdf>
  2. <http://channel9.msdn.com/shows/Going+Deep/Expert-to-Expert-Web-Programming-with-Flapjax/>
- 

## **3.7 July**

### **3.7.1 I can see right through you (2009-07-04 21:28)**

Once communication can happen across the boundary between a virtual machine and the code running on top of it, things can get a little interesting. In particular, behaviour that seems obvious may not happen. This blog entry shows a short journey in the quest to get the following piece of code to fail.

```
object obj = new MyObject();
MyObject obj2 = (MyObject)obj;
```

We start with MyObject having the definition

```
class MyObject { }
```

Now it turns out that the .NET framework has support for an object called a *Transparent proxy*. This special type, which is known to the virtual machine, has the ability to turn standard function calls into objects supporting the IMessage interface. This interface allows the message object to be queried for details about the target of the call, the actual method and the arguments. There is additional support for returning a result from the trapped call. Of course, [1]this interception doesn't come for free, requiring the VM to disable various optimisations. This is done on a per-object basis by making the object inherit from MarshalByRefObject. The virtual machine can then test to see if an object inherits from this class in order to see if various optimisations are allowed.

The first thing we need to do is to hijack the object creation process, allowing us to put the transparent proxy between the newly minted object and the client. For ContextBound objects, where ContextBound is a subclass of MarshalByRefObject, the CLR will look for attributes of a certain type during the object creation process and use the CreateInstance method on these attributes to form the target object. We define a suitable attribute for inserting our proxy into the creation process.

```
[
```

```
AttributeUsage(AttributeTargets.Class]
class MyProxyAttribute : ProxyAttribute
{
    public override MarshalByRefObject CreateInstance(Type serverType)
    {
        MarshalByRefObject target = base.CreateInstance(serverType);
        RealProxy proxy = new MyProxy(target, serverType);
        return (MarshalByRefObject)proxy.GetTransparentProxy();
    }
}
```

Our proxy is a subclass of RealProxy. It will override the Invoke method. This is the method that is passed an object reflecting the call that is being made. We need to handle the case where we are being called in the construction process by checking for IConstructionCallMessage in which case we can use InitializeServerObject to allow the construction of the object to complete. Typically in the non-constructor case we would use something like RemotingServices.ExecuteMessage to convert the object representation of the call into a normal C # call. Here we simply assume that the method will return a string and return a string containing the name of the called method.

Normally, the proxy would not implement IRemotingTypeInfo for which we define CanCastTo and TypeName items that throw an exception, but we'll use these later.

```

class MyProxy : RealProxy, IRemotingTypeInfo
{
    readonly MarshalByRefObject m _Target;
    public MyProxy(MarshalByRefObject target, Type serverType) : base(serverType)
    {
        m _Target = target;
    }
    public override System.Runtime.Remoting.Messaging.IMessage Invoke(System.Runtime.Remoting.Messaging.IMessage msg)
    {
        IMethodCallMessage call = (IMethodCallMessage)msg;
        IConstructionCallMessage ctor = call as IConstructionCallMessage;
        if (ctor != null)
        {
            RealProxy rp = RemotingServices.GetRealProxy(m _Target);
            rp.InitializeServerObject(ctor);
            MarshalByRefObject tp = (MarshalByRefObject)this.GetTransparentProxy();
            return EnterpriseServicesHelper.CreateConstructionReturnMessage(ctor, tp);
        }
        string result = "This was a call to " + call.MethodName;
        return new ReturnMessage(result, null, 0, null, call);
    }
    public bool CanCastTo(Type fromType, object o)
    {
        throw new NotImplementedException();
    }
    public string TypeName
    {
        get
        {
            throw new NotImplementedException();
        }
        set
        {
            throw new NotImplementedException();
        }
    }
}

```

We add a method to MyObject to demonstrate this.

[

```
MyProxy]
class MyObject : ContextBoundObject
{
    public string Message1()
    {
        return "Hello";
    }
}
```

Calling the following code will now leave result1 with the value "This was a call to Message1"

```
object obj = new MyObject();
MyObject obj2 = (MyObject)obj;
string result1 = obj2.Message1();
```

The CanCastTo method is used by the VM to determine the result of type tests. If we add a new interface

```
interface IAnother
{
    string Message2();
}
```

and change the definition of CanCastTo to

```
public bool CanCastTo(Type fromType, object o)
{
    return true;
}
```

then we can also run the code, even though MyObject does not implement this interface, leaving result2 equal to "This was a call to Message2".

```
IAnother obj3 = obj2 as IAnother;
string result2 = obj3.Message2();
```

If we now define a new class

```
class Dummy : MarshalByRefObject { }
```

and redefine the proxy to prevent the casting in CanCastTo and to make this type the proxy target type

```
class MyProxy : RealProxy, IRemotingTypeInfo
{
    public MyProxy(MarshalByRefObject target, Type serverType) : base(typeof(Dummy))
    {
    }
    public override System.Runtime.Remoting.Messaging.IMessage Invoke(System.Runtime.Remoting.Messaging.IMessage msg)
```

```

{
IMethodCallMessage call = (IMethodCallMessage)msg;
IConstructionCallMessage ctor = call as IConstructionCallMessage;
if (ctor != null)
{
MarshalByRefObject tp = (MarshalByRefObject)this.GetTransparentProxy();
return EnterpriseServicesHelper.CreateConstructionReturnMessage(ctor, tp);
}
string result = "This was a call to " + call.MethodName;
return new ReturnMessage(result, null, 0, null, call);
}
public bool CanCastTo(Type fromType, object o)
{
return false;
}
public string TypeName
{
get
{
throw new NotImplementedException();
}
set
{
throw new NotImplementedException();
}
}
}
}

```

then running this code causes an InvalidCastException on the second line.

```

object obj = new MyObject();
MyObject obj2 = (MyObject)obj;

```

What has this shown? Simply that Transparent proxies live in the region between the virtual machine and the normal world of code that compiles to IL, and that code running in this region can generate strange behaviours.

- 
1. <http://blogs.msdn.com/cbrumme/archive/2003/07/14/51495.aspx>

### 3.7.2 Thanks for clearing that up (2009-07-04 21:38)

The other day I was reading the section on accessibility domains in the [1]C # language specification 3.0 when I came across the section 3.5.3 on protected access for instance members. I didn't quite understand the rationale for this, but fortunately [2]Eric Lippert's blog has an old post on the subject which explains it well.

- 
1. <http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx>
  2. <http://blogs.msdn.com/ericlippert/archive/2005/11/09/491031.aspx>

### 3.7.3 Some more reactions (2009-07-11 08:28)

I'd just finished re-reading the section of Paul Hudak's [1]book on implementing Functional Reactive Programming in the FRAN style and had read the section 12 of a thesis on [2]Lula which detailed some of the implementation ideas, when along comes a [3]channel9 [4]video where Eric Meijer talks about some of the ideas behind the .NET Reactive Framework and its IObservable dual of the of the IEnumerator interface. It will be interesting to see where Microsoft go with this framework in the future.

- 
1. [http://www.amazon.co.uk/Haskell-School-Expression-Functional-Programming/dp/0521644089/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1247300448&sr=8-1](http://www.amazon.co.uk/Haskell-School-Expression-Functional-Programming/dp/0521644089/ref=sr_1_1?ie=UTF8&s=books&qid=1247300448&sr=8-1)
  2. <http://tobias-lib.ub.uni-tuebingen.de/volltexte/2001/266/pdf/lula-thesis.pdf>
  3. <http://channel9.msdn.com/>
  4. <http://channel9.msdn.com/shows/Going+Deep/Expert-to-Expert-Brian-Beckman-and-Erik-Meijer-Inside-the-NET-Reactive-Framework-Rx/>

### 3.7.4 If you re-implement, you may get more than you bargained for (2009-07-13 21:53)

I love interfaces. They provide a nice contract against which you can implement groups of related functions and also allow different concrete implementations to be plugged into a system. COM displayed some of the beauty of such a scheme.

C# has a notion of an interface being implemented just when it is specified following the base class list in the class definition. The specification goes out of its way to say that if two interfaces IA and IB are related as subtypes with IB : IA, then whenever IB is re-implemented, so is IA. One question is whether re-implementing IA can change the behaviour of IB.

```
class Program
{
    static void Main(string[] args)
    {
        One one = new One();
        ((IA)one).Method1();
        ((IB)one).Method1();
        ((IB)one).Method2();
        ((IC)one).Method1();
        Console.WriteLine();
        Two two = new Two();
        ((IA)two).Method1();
        ((IB)two).Method1();
        ((IB)two).Method2();
        ((IC)two).Method1();
    }
}
```

```
interface IA
{
    void Method1();
}
```

```
interface IB : IA
{
    void Method2();
}
```

```
interface IC
{
```

```
void Method1();
}

class One : IA,IB,IC
{
public void Method1()
{
Console.WriteLine("Method One");
}
public void Method2()
{
Console.WriteLine("Method Two");
}
}
```

In the following definition, the interface IB, and hence the subtype IA will also be re-implemented.

```
class Two : One, IB
{
public new void Method1()
{
Console.WriteLine("Method One: Next definition");
}
public new void Method2()
{
Console.WriteLine("Method Two: Next definition");
}
}
```

This will lead to the following output.

Method One  
Method One  
Method Two  
Method One

Method One: Next definition  
Method One: Next definition

Method Two: Next definition  
Method One

We can change the implementation to make the class Two implement IA instead of IB.

```
class Two : One, IA
{
    public new void Method1()
    {
        Console.WriteLine("Method One: Next definition");
    }
    public new void Method2()
    {
        Console.WriteLine("Method Two: Next definition");
    }
}
```

We get the following output, showing that we see the re-implementation of IA and the methods of IB that IB inherits from IA.

Method One  
Method One  
Method Two  
Method One

Method One: Next definition  
Method One: Next definition  
Method Two  
Method One

### **3.7.5 The release event has arrived (2009-07-24 06:10)**

At last it looks as if a [1]version of Rx, the Reactive framework is available. Contained in System.Reactive.dll and released as part of the [2]Silverlight Toolkit, this is certainly going to be an interesting library for CLR 4. [3]This InfoQ posting contains some general information including a link to a Channel9 interview with Erik Meijer which I've previously blogged about.

- 
1. <http://themechanicalbride.blogspot.com/2009/07/introducing-rx-linq-to-events.html>
  2. <http://silverlight.codeplex.com/SourceControl/ListDownloadableCommits.aspx>
  3. <http://www.infoq.com/news/2009/07/Reactive-Framework-LINQ-Events>

---

### **3.7.6 So that's what you do all day (2009-07-24 06:12)**

I recently got the chance to [1]blog about what I've been up to at work in the last couple of months.

- 
1. <http://www.simple-talk.com/community/blogs/clivet/archive/2009/07/20/74071.aspx>

---

## **3.8 August**

### **3.8.1 As easy as Pi (2009-08-02 15:53)**

[1]Communicating and mobile systems: The pi-calculus by Robin Milner

This was a really fantastic book, which develops a mathematical treatment of concurrency using some really beautiful mathematics.

The book starts with a chapter on Automata, covering regular languages and Arden's rule which can be used to express the accepted languages of an automaton as a regular expression. The text then covers how we can study the interaction of such automata, by way of complementary transitions - one automata transitions using 'a' and the other transitions using 'a bar' at the same time. The text then moves on to labelled transition systems. Within this framework, we study the notion of strong simulation, where two such systems are equivalent if a series of external observations on one can be simulated by the other. This leads to a notion of bisimulation, when the two system simulate each other, making them equivalent when observed by an outside party which carries out experiments on them. Strong simulation, for example, allows the two machines  $a.(b|c)$  and  $a.b | a.c$  to be differentiated.

A language of process expressions is then defined, and a set of rules is given for determining the internal reactions of the system. This is studied and then a labelled transition system is defined for the system of process expressions. At this point we can apply strong bisimulation to allow us to determine when two process expressions are equivalent. At this point in the text, the use of induction over the height of the reduction tree together with case analysis of the last step of the reduction, allows us to almost magically derive a whole series of interesting results.

It turns out however, that we really want the internal transitions of the system to be ignored when we are considering if two processes are equivalent. Hence the text moves onto a notion of weak bisimulation. In this notion, any transition can be surrounded before and after by an arbitrary number of internal reactions. The book then has a large number of examples of system for which the equivalence is proved.

Part two of the book moves onto the pi-calculus. This extends the process expressions from simply having two processes interact by having them both doing complementary transitions at the same time, to allowing a value to pass across a channel between the two processes. This monadic pi-calculus is extended to a polyadic pi-calculus using syntactic sugaring, allowing a vector of values to be passed by first passing new channel and then passing values across this new local channel. We then go on to look at strong and weak bisimulation using this new system. Along the way, there are many examples, including the translation of the lambda calculus into this computational model and the use of sorts which allows the checking of the validity of the channel interactions and also allows more proofs to be derived for certain types

of interactions where the sorts constrain the interactions. Examples also show how an object oriented language can be embedded into the framework.

In summary, a thought provoking book with some beautiful theory.

1. [http://www.amazon.co.uk/Communicating-Mobile-Systems-Pi-Calculus-Calculus/dp/0521658691/ref=sr\\_1\\_3?ie=UTF8&qid=1249228039&sr=8-3](http://www.amazon.co.uk/Communicating-Mobile-Systems-Pi-Calculus-Calculus/dp/0521658691/ref=sr_1_3?ie=UTF8&qid=1249228039&sr=8-3)
- 

### **3.8.2 A nett gain of knowledge (2009-08-02 16:00)**

[1]Expert .NET 2.0 IL Assembler by Serge Lidin

I've been meaning to read this book for a while, not really from the point of view of understanding the syntax of IL assembler, but as a means of gaining a better understanding of the CLR itself. The low level of the IL assembler code allows one to see exactly what the platform can do, rather than seeing it via the simplification of the languages such as C # and VB that run on top of the framework. In this regard (and many others), the book is great. It covers the format of the assemblies and modules on disc, covering parts of the PE format and various metadata tables that are used by the CLR. There are great chapters on generics, exceptions and the IL instructions, all containing lots of interesting examples and observations about the way the facility is used by higher level languages and the historical context. Indeed in some places, the author admits that he cannot see the reason for something being as it is, apart from history. There is some coverage of the interoperability against COM. (Incidentally, there was a good talk at PDC on this [2]TL34)

Knowledge of IL assembler is good for experimenting with the platform, with the trick of disassembling using ILDASM being a good way to get hold of a block of code that can be easily modified, so this book is well worth reading just for an understanding of this. However, it contains so much more, and I'll be going back to chunks of it in the future to improve my

understanding.

1. [http://www.amazon.co.uk/Expert-NET-2-0-IL-Assembler/dp/1590596463/ref=sr\\_1\\_1?ie=UTF8&qid=1249228528&sr=8-1](http://www.amazon.co.uk/Expert-NET-2-0-IL-Assembler/dp/1590596463/ref=sr_1_1?ie=UTF8&qid=1249228528&sr=8-1)
  2. <http://mschnlnine.vollnwd.net/d1/pdc08/WMV-HQ/TL34.wmv>
- 

### 3.8.3 How good is your translation? (2009-08-02 16:08)

[1]Transposing F to C #: Expressivity of parametric polymorphism in an object-oriented language by Andrew Kennedy and Don Syme.

I'd been meaning to read this paper for a while, but have only just got around to it. This is an interesting paper giving a translation of System F into C #, showing that virtual functions and inheritance can be used to express parametric polymorphism that is more powerful than that found in languages like ML. A translation is given to a featherweight version of C # and the authors prove the correctness of the translation.

In order to get a handle on the translation I thought I would have a go at expressing the translation of a simple System F expression into F # and C #. As usual the translation into F # is much more concise than the translation into C #.

We are looking at

All X. All Y. fn (y : Y) (fn x : Y -> X) (x y)

This example is complicated enough to show how the translation uses closure conversion to close over the values from outer scopes into fields of the inner scope, and how we use All classes to express a collection of similar schemes of polymorphism. Function types are translated into instances of Arrow which reflects the typed signature of the function.

We work from the inside function, translating function application into calls of app and type application into calls of tyapp.

```
[<AbstractClass>]  
type Arrow<'x,'y>() =  
abstract member app : 'x -> 'y
```

```
type C2<'x,'y> (y) =  
inherit Arrow<Arrow<'y,'x>, 'x>()  
override this.app(x) = x.app(y)
```

```
type C1<'x,'y> () =  
inherit Arrow<'y,Arrow<Arrow<'y,'x>, 'x>>()  
override this.app(y) = new C2<'x, 'y>(y) :> Arrow<Arrow<'y,'x>, 'x>
```

```
[<AbstractClass>]  
type All_type_2<'z>() =  
abstract member tyapp<'y> : unit -> Arrow<'y,Arrow<Arrow<'y, 'z>, 'z>>
```

```
type T2<'x>() =  
inherit All_type_2<'x>()  
override this.tyapp<'y>() = new C1<'x,'y>() :> Arrow<'y,Arrow<Arrow<'y, 'x>, 'x>>
```

We can test this out using a function that returns the length of an incoming string

```
type LengthFn() =  
inherit Arrow<string, int>()  
override this.app(x) = x.Length
```

```
> (new T2<int>()).tyapp<string>();  
val it : Arrow<string,Arrow<Arrow<string,int>,int>>  
= FSI_0002+C1'2[System.Int32,System.String]  
> it.app("hello");;  
val it : Arrow<Arrow<string,int>,int>  
= FSI_0002+C2'2[System.Int32,System.String]  
> it.app(new LengthFn());;  
val it : int = 5
```

And a function that concatenates an incoming string to itself

```
type StringFn() =  
inherit Arrow<string, string>()  
override this.app(x) = x + x
```

```
> (new T2<string>()).tyapp<string>();  
val it : Arrow<string,Arrow<Arrow<string,string>,string>>  
= FSI_0002+C1'2[System.String,System.String]  
> it.app("hello");;  
val it : Arrow<Arrow<string,string>,string>  
= FSI_0002+C2'2[System.String,System.String]  
> it.app(new StringFn());;  
val it : string = "hellohello"
```

### 3.8.4 Say that again... (2009-08-02 16:14)

We can also translate the code of the previous entry into C #

```
abstract class Arrow<X, Y>
{
    public abstract Y app(X x);
}

class C2<X,Y> : Arrow<Arrow<Y,X>, X>
{
    Y y;
    public C2(Y y)
    {
        this.y = y;
    }
    public override X app(Arrow<Y,X> x)
    {
        return x.app(y);
    }
}

class C1<X,Y> : Arrow<Y, Arrow<Arrow<Y,X>, X>>
{
    public override Arrow<Arrow<Y,X>,X> app(Y x)
    {
        return new C2<X,Y>(x);
    }
}

abstract class All_Type_2<Z>
{
    public abstract Arrow<Y, Arrow<Arrow<Y,Z>, Z>> tyapp<Y>();
}
```

```

class T2<X> : All_Type_2<X>
{
public override Arrow<Y,Arrow<Arrow<Y,X>,X>> tyapp<Y>()
{
return new C1<X,Y>();
}
}

class LengthFn : Arrow<string, int>
{
public override int app(string x)
{
return x.Length;
}
}

class Program
{
static void Main(string[] args)
{
var app1 = (new T2<int>()).tyapp<string>();
var app2 = app1.app("hello");
Console.WriteLine(app2.app(new LengthFn()));
}
}

```

---

### **3.8.5 I wish I were that clever (2009-08-07 16:53)**

[1]Journey through genius: The great theorems of mathematics by William Dunham

This was a fantastic holiday read. I'd previously read Dunham's [2]The Calculus Gallery which is also a very good book. This book highlights several of the great theorems of mathematics, giving historical context and biographical details of the theorem's creator. It has some great coverage of Euclid's Elements, Archimedes' calculation of Pi and coverage of Heron's theorem. In all cases the historical information is great and the proof is given in a very clear fashion.

1. [http://www.amazon.co.uk/Journey-Through-Genius-Theorems-Mathematics/dp/014014739X/ref=sr\\_1\\_1?ie=UTF8&qid=1249663503&sr=8-1](http://www.amazon.co.uk/Journey-Through-Genius-Theorems-Mathematics/dp/014014739X/ref=sr_1_1?ie=UTF8&qid=1249663503&sr=8-1)
  2. [http://www.amazon.co.uk/Calculus-Gallery-Masterpieces-Newton-Lebesgue/dp/0691136262/ref=pd\\_sim\\_b\\_1](http://www.amazon.co.uk/Calculus-Gallery-Masterpieces-Newton-Lebesgue/dp/0691136262/ref=pd_sim_b_1)
- 

### **3.8.6 Money, money, money. (2009-08-07 17:03)**

[1]The Ascent of Money by Niall Ferguson

An interesting book giving the history of the world's financial system, including bonds and stocks and how these items came into existence. A very interesting read.

1. [http://www.amazon.co.uk/Ascent-Money-Financial-History-World/dp/014103548X/ref=sr\\_1\\_1?ie=UTF8&qid=1249664413&sr=8-1](http://www.amazon.co.uk/Ascent-Money-Financial-History-World/dp/014103548X/ref=sr_1_1?ie=UTF8&qid=1249664413&sr=8-1)
- 

### **3.8.7 Refactor this. (2009-08-07 17:21)**

There's a good edition of the [1]Industry Misinterpretations podcast where the presenters interview the people who created the first refactoring browser (for the dynamic language smalltalk). All the talk of object models representing programs got me thinking again about [2]Intentional Programming and representing programs in a form that isn't source code. Using a tool like [3]Reflector, it sometimes feels as if the program really exists as IL and source code views in several languages can be constructed on the fly by a tool.

1. [http://www.cincomsmalltalk.com/blog/blogView?showComments=true&printTitle=Industry\\_Misinterpretations\\_147\\_Refactoring\\_for\\_Fun\\_and\\_Profit&entry=3426663681](http://www.cincomsmalltalk.com/blog/blogView?showComments=true&printTitle=Industry_Misinterpretations_147_Refactoring_for_Fun_and_Profit&entry=3426663681)

2. [http://en.wikipedia.org/wiki/Intentional\\_programming](http://en.wikipedia.org/wiki/Intentional_programming)
  3. <http://www.red-gate.com/products/reflector/>
- 

### **3.8.8 I have achieved closure (2009-08-26 17:09)**

[1]Programming Clojure by Stuart Halloway

A great read which introduced a very interesting language by examining its use in defining a build system.

Clojure itself looks really interesting. It's a cut down version of Lisp (or Scheme) which integrates well with the java platform on which it runs. There is an emphasis on functional styles of programming and the language is designed with an eye on concurrency, including a simple transactional memory system and actors in the language itself. Basing it on Lisp, which facilitates the use of macros, means the language can have a small core and is easily extensible via internal dsls.

The book is really clear and introduces the concepts very well indeed. I'll have more to say on Clojure soon.

1. [http://www.amazon.co.uk/Programming-Clojure-Pragmatic-Programmers-Halloway/dp/1934356336/ref=sr\\_1\\_1?ie=UTF8&qid=1251305294&sr=1-1](http://www.amazon.co.uk/Programming-Clojure-Pragmatic-Programmers-Halloway/dp/1934356336/ref=sr_1_1?ie=UTF8&qid=1251305294&sr=1-1)
-

### **3.8.9 You're deeper than you look (2009-08-26 17:11)**

[1]The Formal Semantics Of Programming Languages by Glynn Winskel

A great introductory text covering operational, axiomatic and denotational semantics, showing their application to define a simple imperative and a simple higher typed functional language. The material is well ordered and clear, and has a lot of discussion around the various points that it makes. The proofs of the equivalence of the operational and denotational semantics are a good read and there's an interesting chapter on concurrency. It contains some interesting exercises too.

- 
1. [http://www.amazon.co.uk/Formal-Semantics-Programming-Languages-Introduction/dp/0262731037/ref=sr\\_1\\_1?ie=UTF8&qid=1251305704&sr=1-1](http://www.amazon.co.uk/Formal-Semantics-Programming-Languages-Introduction/dp/0262731037/ref=sr_1_1?ie=UTF8&qid=1251305704&sr=1-1)

### **3.8.10 If you don't succeed, retry. (2009-08-26 17:14)**

[1]STM.NET version 1.0 is out.

I haven't tried it yet, but have read the [2]manual. STM is available (as a first experimental version) for the .NET platform. It will be interesting to see where this technology goes in the future. There's an old channel9 interview [3]here and a newer one [4]here.

- 
1. <http://blogs.msdn.com/stmteam/archive/2009/07/28/stm-net-released.aspx>
  2. [http://download.microsoft.com/download/9/5/6/9560741A-EEFC-4C02-822C-BB0AFE860E31/STM\\_User\\_Guide.pdf](http://download.microsoft.com/download/9/5/6/9560741A-EEFC-4C02-822C-BB0AFE860E31/STM_User_Guide.pdf)
  3. <http://channel9.msdn.com/shows/Going+Deep/Software-Transactional-Memory-The-Current-State-of-the-Art/>
  4. <http://channel9.msdn.com/posts/Charles/STMNET-Who-What-Why/>

## 3.9 September

### 3.9.1 Put it together, then break it apart (2009-09-02 18:26)

Ian recently posted a question on [1] how to make data-binding more type safe. While looking at his original solution, I realised how nice the new lambda expressions of C # are. They are typechecked, giving the programmer some safety, and can be decomposed.

Given a class like the following, we can represent the data binding against the Foo property using a lambda () => instance.Foo. This lambda is typechecked so we will get compile time errors if we try to use an unknown property.

```
public class A
{
    public int Foo
    {
        get
        {
            return 20;
        }
    }
}
```

Then later we can take this lambda expression apart to get the instance and the property information.

```
A instance1 = new A();
Expression<Func<int>> theFunction = () => instance1.Foo;
```

```
A instance2;
System.Reflection.PropertyInfo property;
ExtractTargetAndProperty<A>(theFunction, out instance2, out property);
```

This method to extract the instance and property can be easily written as:

```
static void ExtractTargetAndProperty<T>(Expression<Func<int>> xx, out T thisPointer, out System.Reflection.PropertyInfo member)
{
    MemberExpression exp = (MemberExpression) xx.Body;
    Expression thisValue = exp.Expression;
    Expression<Func<T>> getThis = Expression.Lambda<Func<T>>(thisValue, null);
    Func<T> thisFetcher = getThis.Compile();
    thisPointer = thisFetcher();
    member = (System.Reflection.PropertyInfo) exp.Member;
}
```

The need to wrap the expression with a () => to delay the evaluation isn't as tidy as I'd like. In languages like Common Lisp, macros are a great way to get the wrapping automatically added. Maybe we'll get them in C # one day.

1. <http://stackoverflow.com/questions/1329138/how-to-make-databinding-type-safe-and-support-refactoring/1333874#1333874>
- 

### 3.9.2 I've got a good memory for transactions (2009-09-08 19:51)

The .NET [1]transactional memory team blog has been getting a few interesting posts of late. Additionally, some of the team have produced [2]a good paper on tracking the correspondance between pointers and the objects to which they point. There's also an interesting Microsoft Research [3]paper on using optimistic concurrency for reading and pessimistic concurrency for writing when implementing software transactions.

1. <http://blogs.msdn.com/stmteam/>
  2. [http://transact09.cs.washington.edu/17\\_paper.pdf](http://transact09.cs.washington.edu/17_paper.pdf)
  3. <http://research.microsoft.com/en-us/um/people/tharris/papers/2006-pldi.pdf>
-

### 3.9.3 It's not what you do, but the way that you do it (2009-09-20 15:25)

Take a simple few lines of C #.

```
static void Main()
{
A foo = new A();
}
```

If it compiles, then it can't fail at runtime, right?

C # has got some extensions to make it "easier" to deal with legacy COM components (though I'm not sure I believe that). If an interface is marked with appropriate attributes then you're allowed to use new on the interface type and the compiler will, behind the scenes, transform this instantiation into a construction of another class. The compiler has no problem with this class being unrelated to the interface that we started with.

[System.Runtime.InteropServices.

```
CoClass(typeof(Test)),
System.Runtime.InteropServices.ComImport,
System.Runtime.InteropServices.Guid("00000000-0000-0000-0000-000000000000" )]
public interface A
{
}
```

```
public class Test
{
}
```

There's some discussion of this [1]here on Stack Overflow.

1. <http://stackoverflow.com/questions/1093536/how-does-the-c-compiler-detect-com-types/1094065#1094065>

### **3.9.4 You should have seen through that (2009-09-26 16:17)**

Take this simple bit of C # and execute it. What happens?

```
class Handler
{
    public Assembly TypeResolve(object sender, ResolveEventArgs args)
    {
        return null;
    }
}

static void Main(string[] args)
{
    AppDomain myDomain = AppDomain.CreateDomain("myDomain", null, AppDomain.CurrentDomain.BaseDirectory, "", false);
    Handler handler = new Handler();
    myDomain.TypeResolve += new ResolveEventHandler(handler.TypeResolve) ;
}
```

The answer becomes obvious if you add the line

```
Console.WriteLine(RemotingServices.IsTransparentProxy(myDomain));
```

after the creation of the AppDomain.

The delegate that is passed into the event handler needs to be remoted, and there will be serialization exception when marshaling the instance of Handler as it is not attributed as Serializable. When a colleague came up to me and said he had an event handler that worked when the handler method was static, but not when the handler was an instance method, I was sceptical, but he was proved right in the end.

---

### **3.9.5 I'm not too lazy to watch them! (2009-09-26 16:41)**

There are some good videos from the recent Haskell Symposium available [1]here. I've only had time to watch a couple of them, but the talks on types as calling conventions and defunctionalization were both really interesting.

While we're on the subject of online videos, there's a good one on the addition of dynamic to C # 4.0 [2]here and an excellent interview with Butler Lampson [3]here, where he uses the slogan "stripe, stream or struggle" as a way of expressing the ways to gain parallelism in programs (data parallelism, pipelining or dataflow, or struggle).

- 
1. <http://www.vimeo.com/album/128530/page:1>
  2. <http://channel9.msdn.com/posts/CharlieCalvert/CSharp-4-Dynamic-with-Chris-Burrows-and-Sam-Ng/>
  3. <http://channel9.msdn.com/shows/Going+Deep/E2E-Erik-Meijer-and-Butler-Lampson-Abstraction-Security-Embodiment/>

---

### **3.9.6 So that's how it works! (2009-09-28 08:31)**

[1]Microsoft Windows Internals (fourth edition) by Mark Russinovich and David Solomon

A thoroughly enjoyable read. A book which probes into the internals of the Windows operating system, with great coverage of aspects such as security, device drivers and the memory manager. The book contains a number of experiments that should be tried to gain familiarity with the various kernel data structures. Great explanations.

- 
1. [http://www.amazon.co.uk/Microsoft-Windows-Internals-Fourth-Pro-Developer/dp/0735619174/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1254122686&sr=8-1](http://www.amazon.co.uk/Microsoft-Windows-Internals-Fourth-Pro-Developer/dp/0735619174/ref=sr_1_1?ie=UTF8&s=books&qid=1254122686&sr=8-1)

---

### **3.9.7 That's what its good for (2009-09-28 08:32)**

[1]The Calculus Wars by Jason Bardi

Having heard bits and pieces about the struggle for priority between Leibnitz and Newton, it was great to read a book that documented the history of their disagreement. The book lots of interesting bits and pieces including the fact that Leibnitz invented binary notation and the challenge behind the Brachistochrone Problem.

1. [http://www.amazon.co.uk/Calculus-Wars-Jason-Bardi/dp/1843440369/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1254122924&sr=1-1-spell](http://www.amazon.co.uk/Calculus-Wars-Jason-Bardi/dp/1843440369/ref=sr_1_1?ie=UTF8&s=books&qid=1254122924&sr=1-1-spell)
- 

### **3.9.8 Different, yet the same (2009-09-28 08:33)**

The slides for the recent jvm languages summit are available [1]here. Lots of interesting material including a trace based JIT, better support for dynamic languages and some good material related to Clojure.

1. [http://wiki.jvmlangsummit.com/Main\\_Page](http://wiki.jvmlangsummit.com/Main_Page)
- 

### **3.9.9 That was quicker than I expected (2009-09-30 16:58)**

Most of the time data that you need to share between threads is protected by a lock, and the barrier properties of a lock guarantee that stores and loads don't get reordered by the combination of the compiler, the jit and the hardware. However, there are times when you aren't using a lock to protect data in one of the threads, and in this case you need to be aware of the .NET memory model in order to be able to reason about the consequences.

Why am I thinking about this now? Well, Martin Simmons recently did a talk about the [1]multi-processor extension to Lispworks. In order to give guarantees to a programmer, so that they can be sure that a program will work across architectures, the program needs to stay within the realm of the defined behaviour of the memory model.

So what is the .NET memory model?

It is a set of rules that define how loads and stores to memory can appear to change order when seen from multiple threads. From a single thread, causality guarantees that a store to a location followed by a read will see the value that was written. However when a store to a location and a read from a different location occur, effects such as caching in the memory subsystem can be reasonned about by considering how the load and store may be reordered.

The CLR 2.0 memory model defines the following.

- (i) A load may move pass a load that follows in the instruction stream - load-load is possible.
- (ii) load-store is possible
- (iii) store-store is not possible
- (iv) store-load is possible.

There are then several kinds of fence that prevent the motion of loads and stores.

- (i) an acquire fence prevents loads and stores that follow it from moving before it.
- (ii) a release fence prevents loads and stores that precede it from moving after it
- (iii) a full fence prevents loads and stores moving across it in either direction.

Taking a lock (using the Monitor class for example), or using an Interlocked operation or using Thread.MemoryBarrier, all create a full fence. A read of a volatile or using Thread.VolatileRead is an acquire fence. A write of a volatile or using Thread.VolatileWrite is a release fence. Some

compiler optimisations are also prevented; the compiler/JIT cannot introduce extra load or stores for volatile variables or for variables that reference the GC heap, though it can merge multiple consecutive load or stores to the same location.

It is these kind of guarantees that almost makes the following singleton pattern safe on the CLR.

```
private Class m_Instance = null; // Needs to be volatile
private object m_Lock = new object();
```

```
public Class Instance
{
    get
    {
        if (m_Instance == null)
        {
            lock(m_Lock)
            {
                if (m_Instance == null)
                {
                    m_Instance = new Class();
                }
            }
        }
    }
}
```

In the weaker ECMA memory model, the store-store reorder may happen, so the stores initialising the new instance and the store setting the m\_Instance variable may intermix, potentially allowing another thread to see the partially constructed object. In the CLR 2.0 memory model, store-store isn't allowed, so no other thread may see the partially constructed instance.

However, we DO need to make `m_Instance` volatile. The .NET memory model doesn't guarantee that load-load reordering won't happen (though in reality it will only happen on the [2]IA64). This means that the data could be read from the fields of `m_Instance` before they are initialized, much in the same way the stores can happen out of order in the weaker memory models when the singleton instance is constructed.

Interesting stuff that makes it clear how hard it is to do lock-free programming on mutable datastructures.

1. [http://www.lispworks.com/SMP\\_LispWorks\\_ECLM2009.pdf](http://www.lispworks.com/SMP_LispWorks_ECLM2009.pdf)
  2. [http://www.pctechguide.com/21Architecture\\_IA-64.htm](http://www.pctechguide.com/21Architecture_IA-64.htm)
- 

## 3.10 October

### 3.10.1 Assembly required! (2009-10-03 07:43)

There's a good article in the [1]May issue of MSDN magazine regarding the Assembly binder, describing how it behaves when you mix `LoadFrom` and `Load`. The binder takes care of ensuring that if you do a `LoadFrom` on an assembly that would be found in the directory structures that are scanned by `Load`, then a subsequent `Load` finds the same assembly. It achieves this by having two caches, one for `Load` and one for `LoadFrom`, and promotes entries from the `LoadFrom` cache into the `Load` cache if they meet the criterion above.

```
AppDomain current = AppDomain.CurrentDomain;

bool loadFirst = Convert.ToBoolean(Console.ReadLine());
string loadFromAssemblyPath = Console.ReadLine();
string privatePath = Console.ReadLine();

if (!String.IsNullOrEmpty(privatePath))
{
    current.AppendPrivatePath(privatePath);
}
```

```

Assembly loadFromAssembly = null;
Assembly loadAssembly = null;

Console.WriteLine("AppBase: " + current.BaseDirectory);
Console.WriteLine("Private: " + current.RelativeSearchPath);

foreach (bool pass in new bool[] { true, false })
{
    if (loadFirst == pass)
    {
        loadFromAssembly = Assembly.LoadFrom(loadFromAssemblyPath);
        Console.WriteLine("LoadFrom: " + loadFromAssembly.Location);
    }
    else
    {
        loadAssembly = Assembly.Load("ClassLibrary1");
        Console.WriteLine("Load :" + loadAssembly.Location);
    }
}

```

Console.WriteLine("Assemblies equal: " + (loadAssembly == loadFromAssembly));

Taking a ClassLibrary1.dll and putting it into c:\temp and c:\temp2, we get the following results:

```

AppBase: C:\temp
Private:
LoadFrom: C:\temp\ClassLibrary1.dll
Load :C:\temp\ClassLibrary1.dll
Assemblies equal: True

```

```

AppBase: C:\temp
Private:
Load :C:\temp\ClassLibrary1.dll
LoadFrom: C:\temp\ClassLibrary1.dll
Assemblies equal: True

```

```

AppBase: C:\temp
Private:
LoadFrom: c:\temp2\ClassLibrary1.dll
Load :C:\temp\ClassLibrary1.dll
Assemblies equal: False

```

```

AppBase: C:\temp
Private:
Load :C:\temp\ClassLibrary1.dll
LoadFrom: c:\temp2\ClassLibrary1.dll

```

Assemblies equal: False

Putting ClassLibrary1.dll in c:\temp, c:\tempprivate and c:\temp2

AppBase: C:\temp

Private: private

LoadFrom: C:\temp\ClassLibrary1.dll

Load :C:\temp\ClassLibrary1.dll

Assemblies equal: True

AppBase: C:\temp

Private: private

LoadFrom: c:\tempprivate\ClassLibrary1.dll

Load :C:\temp\ClassLibrary1.dll

Assemblies equal: False

Putting ClassLibrary1.dll in c:\tempprivate and c:\temp2

AppBase: C:\temp

Private: private

LoadFrom: C:\tempprivate\ClassLibrary1.dll

Load :C:\tempprivate\ClassLibrary1.dll

Assemblies equal: True

AppBase: C:\temp

Private: private

Load :C:\tempprivate\ClassLibrary1.dll

LoadFrom: C:\tempprivate\ClassLibrary1.dll

Assemblies equal: True

1. <http://msdn.microsoft.com/en-us/magazine/dd727509.aspx>

---

### 3.10.2 LINQ into the data stream (2009-10-03 07:56)

There have been a couple of recent channel9 interviews covering Rx: The Livelabs Reactive Framework. You can find them [1]here and [2]here. Currently the only release of this framework is as part of the [3]Silverlight Toolkit. However, using the awesome Cecil, you can [4]modify the assembly to run on top of the standard 3.5 framework.

It is then really easy to get some simple examples going. Start a WindowsForms project, set a reference to the System.Reactive.dll, add a button and a couple of text boxes and try the following.

```
IObservable<Event<EventArgs>> button = Observable.FromEvent<EventArgs>(button1, "Click");
button.Subscribe(() => MessageBox.Show("Hello"));
```

That example causes a message box to appear everytime the button is clicked.

The real power of the Reactive framework is that LINQ queries can be used against it, so we could decide to update a second textbox only when the number of characters in the text is even.

```
IObservable<Event<EventArgs>> text = Observable.FromEvent<EventArgs>(textBox1, "TextChanged");
var data = from _ in text select textBox1.Text;
var eventLengthData = from x in data where x.Length % 2 == 0 select x;
eventLengthData.Subscribe(datum => textBox2.Text = datum);
```

We can have multiple subscribers to the event stream, so we can print out all of the content changes even if we only update the textbox when the data length is even.

```
data.Subscribe(x => Console.WriteLine(x));
```

This concept of merging the streams of data can lead to some interesting results. We'll  
312

look at the example of the generation of a suggestions list for a given set of textbox data, but that will have to wait until next time.

Of course, there are always complications to do with side-effects and threads and their associated synchronisation contexts. There's a good post [5]here discussing some of this in relation to LINQ.

1. <http://channel9.msdn.com/shows/Going+Deep/E2E-Erik-Meijer-and-Wes-Dyer-Reactive-Framework-Rx-Under-the-Hood-1-of-2/>
  2. <http://channel9.msdn.com/shows/Going+Deep/E2E-Erik-Meijer-and-Wes-Dyer-Reactive-Framework-Rx-Under-the-Hood-2-of-2/>
  3. <http://www.codeplex.com/Silverlight>
  4. <http://evain.net/blog/articles/2009/07/30/rebasing-system-reactive-to-the-net-clr>
  5. <http://bartdesmet.net/blogs/bart/archive/2009/09/12/taming-your-sequence-s-side-effects-through-ienumerable-let.aspx>
- 

### 3.10.3 Go on, suggest something (2009-10-05 22:15)

Of course, to get more a grip on a new framework, it is always good to try to write something using it. In one of videos on Rx, the presentors talk about expressing a suggestions framework using a LINQ query. For this, we'll need a textbox, textBox1, and a ListBox, listBox1. When you type into the textbox, suggestions will be generated. These will take the form of the original text with ascending integers appended to the end with a new suggestion being added every two seconds. When the text changes, we'd like the suggestions process to be restarted.

In a second we'll look at the code for an object that raises the suggestions. We'll use an object called Suggestor, which will take the initial string and return an array of suggestions with these returned arrays getting an extra element every two seconds. First, we'll do the code in the form to hook the suggestions into the components.

```
// Grab the synchronization context for the windows form, so that all work is done on the right thread
```

```

SynchronizationContext context = SynchronizationContext.Current;

// Notice changes to the textbox
IObservable<Event<EventArgs>> textChanged = Observable.FromEvent<EventArgs>(textBox1-
, "TextChanged");

// Get a stream of new values of the textbox
var data = from _ in textChanged select textBox1.Text;

// Grab the groups of suggestions, starting a new Suggestor whenever the string in the textbox
changes.
var suggestion =
from start in data
from item in new Suggestor(start).Until(textChanged)
select item;

// Set the new suggestions into the listbox
suggestion.Subscribe(
delegate(string[] x)
{
context.Post(newValue => listBox1.DataSource=newValue, x);
});

```

The code for the windows form looks quite straight-forward. All the interesting code to do with IObservable is hidden in the Suggestor object. This uses a threadpool thread to run code which returns the new suggestions array every two seconds. When the result of the Subscribe is disposed, the thread can be shut down.

```

class Suggestor : IObservable<string[]>
{
readonly string m _Initial;

public Suggestor(string initial)
{
m _Initial = initial;
}

public IDisposable Subscribe(IObserver<string[]> observer)
{
Generator generator = new Generator(observer, m _Initial);
ThreadPool.QueueUserWorkItem(delegate { generator.DoWork(); });
return generator;
}

class Generator : IDisposable

```

```

{
volatile IObserver<string[]> m _Observer;
readonly string m _Initial;
readonly object m _Lock = new object();

public Generator(IObserver<string[]> observer, string initial)
{
m _Observer = observer;
m _Initial = initial;
Console.WriteLine("Generator started " + m _Initial);
}

public void Dispose()
{
lock (m _Lock)
{
m _Observer = null;
Console.WriteLine("Dispose on " + m _Initial);
}
}

public void DoWork()
{
List<string> suggestions = new List<string>();
suggestions.Add(m _Initial);
for(int index = 0; ; index++)
{
lock (m _Lock)
{
suggestions.Add(m _Initial + index);
if (m _Observer == null)
break;
Console.WriteLine("Next on " + m _Initial);
m _Observer.OnNext(suggestions.ToArray());
}
Thread.Sleep(2000);
}
Console.WriteLine("Generator stopped " + m _Initial);
}
}
}
}

```

You can see the final result working here: [1]<http://cid-3f21df299c355e7f.skydrive.live.com/self.-aspx/Pictures/Sugge stions.jpg>

1. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Pictures/Suggestions.jpg>

---

### 3.10.4 Some trees can only have certain kinds of branches (2009-10-18 17:22)

Lambda syntax in C # lets you express more than expressions, but you can't convert such things into expression trees.

```
Func<int> lambda = () => 20;
Expression<Func<int>> tree = () => 20;
Func<int> lambda2 = () => { return 20; };
Expression<Func<int>> tree2 = () => { return 20; };
```

The last fails to compile with the error  
A lambda expression with a statement body cannot be converted to an expression tree

---

### 3.10.5 It's alive, but not as we know it (2009-10-18 17:31)

It was [1]Mark who made me do it. He raved enthusiastically about a Lisp that was now available on the JVM. Of course, Clojure is designed from the ground up to integrate into the Java platform. That makes it curiously like the Common Lisp I know and love, but with differences that also make it feel very alien. I thought I'd list some early observations on the difference between Clojure and Common Lisp.

The first neat feature is the easy way that you can call into the java platform. The standard classes of `java.lang` are automatically made available with the user being able to import other

classes. There is a function, bean, that takes the properties of the java object and converts them into an instance of Clojure's map type. This gives us an easy way to see how Clojure is translated down into the platform.

We can make an instance of the Thread type using the trailing . notation

```
user=> (Thread.)  
#<Thread Thread[Thread-4,5,main]>
```

This is equivalent to using the new function to do the work.

```
user=> (new Thread)  
#<Thread Thread[Thread-5,5,main]>
```

The bean function can then be used to look at the various properties of the created thread via an instance of a read-only map.

```
user=> (bean *1)  
{:interrupted false, :threadGroup #<ThreadGroup java.lang.ThreadGroup[name=main,  
maxpri=10]>, :state #<State NEW>, :class java.lang.Thread, :name "Thread-  
5", :stackTrace #<StackTraceElement[] [Ljava.lang.StackTraceElement;@f102d3>,  
:daemon false, :alive false, :id 13, :contextClassLoader #<AppClassLoader  
sun.misc.Launcher$AppClassLoader@11b86e7>, :uncaughtExceptionHandler #<Thread-  
Group java.lang.ThreadGroup[name=main,maxpri=10]>, :priority 5 }
```

```
user=> (class *1)
clojure.proxy.clojure.lang.APersistentMap
```

One of the curious properties is that there are notions of Symbol and Keyword in Clojure but they aren't the Symbol and Keyword that we know and love. These symbols index into structures called namespaces, but they don't have value or functions cells like in Common Lisp. Instead the binding is held in a cell of type Var which is associated with the symbol or keyword in the namespace.

```
user=> (def foo 10)
#'user/foo
user=> (defn bar [x] x)
#'user/bar
```

We have just defined a variable and a function.

```
user=> foo
10
user=> (bar 20)
20
```

We can use some symbols

```
user=> (bean 'foo)
{:namespace nil, :name "foo", :class clojure.lang.Symbol }
```

```

user=> (bean 'bar)
{:namespace nil, :name "bar", :class clojure.lang.Symbol }
user=> (bean 'user/foo)
{:namespace "user", :name "foo", :class clojure.lang.Symbol }
user=> (bean 'user/bar)
{:namespace "user", :name "bar", :class clojure.lang.Symbol }

```

We look up a symbol in a namespace using the #' reader macro. This is a little confusing as the #' reader macro in Common Lisp expands to a use of the symbol-function special form.

```

user=> (bean #'foo)
{:watches { }, :validator nil, :tag nil, :root 10, :public true, :macro false, :class clojure.lang.Var,
:bound true }
user=> (bean #'user/foo)
{:watches { }, :validator nil, :tag nil, :root 10, :public true, :macro false, :class clojure.lang.Var,
:bound true }

```

```

user=> (bean #'bar)
{:watches { }, :validator nil, :tag nil, :root #<user $bar __664 [3]user $bar __664@e8a021>,
:public true, :macro false, :class clojure.lang.Var, :bound true }
user=> (bean #'user/bar)
{:watches { }, :validator nil, :tag nil, :root #<user $bar __664 [4]user $bar __664@e8a021>,
:public true, :macro false, :class clojure.lang.Var, :bound true }

```

The symbol is resolved using the namespace in the symbol or by using the current namespace which is bound to \*ns\*.

```

user=> (bean *ns*)
{:name user, :mappings {sorted-map #'clojure.core/sorted-map, read-line #'clojure.core/read-
line, re-pattern #'clojure.core/re-pattern, keyword? #'clojure.core/keyword?, ..., unchecked-
divide #'clojure.core/unchecked-divide }, :class clojure.lang.Namespace, :aliases { } }

```

We can see this resolution by moving into another namespace and then trying to resolve the foo symbol from the earlier user package.

```
(in-ns 'bar)
(clojure.core/use 'clojure.core)
```

```
bar=> (bean #'foo)
java.lang.Exception: Unable to resolve var: foo in this context (NO_SOURCE_FILE:143)
bar=> (bean #'user/foo)
{:watches {}, :validator nil, :tag nil, :root 10, :public true, :macro false, :class clojure.lang.Var,
:bound true }
```

There is also a class of keywords, which have various purposes. Typically they are used as the key in maps.

```
user=> (bean :foo)
{:namespace nil, :name "foo", :class clojure.lang.Keyword }
```

In the LispReader.java file we can see some of this resolution happening.

```
public static class EvalReader extends AFn {
    public Object invoke(Object reader, Object eq) throws Exception {
```

```

if (!RT.booleanCast(RT.READEVAL.deref()))
{
throw new Exception("EvalReader not allowed when *read-eval* is false.");
}

PushbackReader r = (PushbackReader) reader;
Object o = read(r, true, null, true);
if(o instanceof Symbol)
{
return RT.className(o.toString());
}
else if(o instanceof IPersistentList)
{
Symbol fs = (Symbol) RT.first(o);
if(fs.equals(THE_VAR))
{
Symbol vs = (Symbol) RT.second(o);
return RT.var(vs.ns, vs.name); //Compiler.resolve((Symbol) RT.second(o),true);
}
if(fs.name.endsWith(".")) {
Object[] args = RT.toArray(RT.next(o));
return Reflector.invokeConstructor(RT.className(fs.name.substring(0, fs.name.length() - 1)), args);
}
if(Compiler.namesStaticMember(fs))
{
Object[] args = RT.toArray(RT.next(o));
return Reflector.invokeStaticMethod(fs.ns, fs.name, args);
}
Object v = Compiler.maybeResolveIn(Compiler.currentNS(), fs);
if(v instanceof Var)
{
return ((IFn) v).applyTo(RT.next(o));
}
throw new Exception("Can't resolve " + fs);
}
else
throw new IllegalArgumentException("Unsupported #= form");
}

```

Clojure has macros, so the function defining form, defn, expands into the basic def special form which creates the Var cells.

```
user=> (macroexpand '(defn foo[x] 10))
(def foo (clojure.core/fn ([x] 10)))
```

The defn has expanded to use the fn special form which defines anonymous functions, much like the lambda of Common Lisp.

There is incidentally a very nice shorthand way to define an anonymous function using the # reader macro.

```
user=> #(print %)
#<user $eval _ _815 $fn _ _817 [5]user $eval _ _815 $fn _ _817@19ecd80>
user=> (apply *1 [10])
10nil
```

With the value slots having been moved into vars, there are a number of tricks available. Vars are bound per thread much like special variables in Common Lisp. This gives an interesting action at a distance flavour.

```
user=> (def x 10)
#'user/x
user=> (defn printx[] (print x))
#'user/printx
user=> (printx)
10nil
user=> (binding [x 30] (printx))
30nil
user=> (defn doprintx[] (printx))
#'user/doprintx
user=> (doprintx)
10nil
```

```
user=> (binding [printx (fn [] (print "I've been changed"))] (doprintx))
I've been changednil
```

It means that it would be easy to write things like the trace macro of Common Lisp which would have an expansion something like:

```
user=> (let [olddefn @ #'printx]
(binding [printx
(fn []
(print "calling")
(olddefn)
(print "called"))]
(doprintx)))
calling10callednil
```

In the above, we used the @ reader macro, which expands into deref, to access the Var cell's contained value which we bound into the lexical variable olddefn.

Note that if you read Programming Clojure by Stuart Halloway, the download on his web site contains 1.1 alpha of Clojure in which the binding example doesn't work. Moreover, Clojure comes with an implementation of memoize. By use a binding form, we can locally optimise to use the memoized version of a function. This seems to me to be much better than the standard examples where the global definition is modified to memoize. Without a means of flushing the cache, this looks to me more like a memory leak.

```
user=> (defn fib [x] (if (< x 2) 1 (+ (fib (- x 1)) (fib (- x 2)))))
#'user/fib
```

```
user=> (time (fib 35))
"Elapsed time: 7977.511248 msecs"
14930352
user=> (time (binding [fib (memoize fib)] (fib 35)))
"Elapsed time: 0.792699 msecs"
14930352
```

Clojure has the usual Lisp methods for writing macros though comma (which is interpreted by the reader as whitespace, allowing a vector to be written as [1,2,3]) is replaced by . We can therefore write the classic erroneous Common Lisp style macro which evalautes the argument twice.

```
user=> (defmacro testmac [x] '(do (print x) (print x)))
 #'user/testmac
user=> (testmac 30)
3030nil
user=> (testmac (do (print "evaled") 50))
evaled50evaled50nil
```

Rather than having to use a method for generating unique variable names such as the common REBINDING macro, the Clojure reader creates such a new variable when the name has a # at the end.

```
user=> (defmacro testmac [x] '(let [new # x] (do (print new #) (print new #))))
 #'user/testmac
user=> (testmac (do (print "evaled") 50))
evaled5050nil
```

In Common Lisp, symbols have an associated property list. This is replaced in Clojure by meta data, a map that can be added to objects.

Maps in Clojure take the following form

```
user=> (def test-map {:a 1 :b 2 })
#'user/test-map
user=> (test-map :a)
1
user=> (test-map :b)
2
```

Our macro from earlier has meta data that we can access using the ^ reader macro.

```
user=> ^ #'testmac
{:macro true, :ns #<Namespace user>, :name testmac, :file "NO_SOURCE_PATH", :line 42,
:arglists ([x]) }
```

One other interesting observation on maps is that they are what destruct expands into.

```
user=> (defstruct mytype :a :b)
#'user/mytype
```

```
user=> (struct mytype 2 3)
{:a 2, :b 3 }
```

There's a mass of other interesting stuff, but I don't have time to type it up now.

1. <http://blog.software-acumen.com/>
  2. mailto:sun.misc.Launcher\$AppClassLoader@11b86e7
  3. mailto:user\$bar\_\_664@e8a021
  4. mailto:user\$bar\_\_664@e8a021
  5. mailto:user\$eval\_\_815\$fn\_\_817@19ecd80
- 

### **3.10.6 Keep the LINQ short (2009-10-23 20:01)**

[1]LINQ Unleashed for C # by Paul Kimmel

This was a fairly interesting read which went through the various changes to C # as the language moved between versions 2 and 3, covering in some detail the LINQ extensions. There were chapters in the various LINQ combinator and the syntax extensions in the form of the select statement which allows easy use of them.

There was fairly detailed coverage of LINQ to SQL and examples of its use in a couple of example applications. The text also covered an example LINQ to Active Directory provider as a way to show what you have to do to implement a LINQ provider.

The book was interesting, but some of the examples were too long, in that the code extended sometimes for 8 to 10 pages.

Unfortunately, a couple of things in the book were clearly wrong. For example, there's an example of shuffling cards which uses the rather concise syntax

```
const int numberOfCards = 52;
int[] cards = new int[numberOfCards];
var shuffler = cards.Select((num, index) =>
    new { Key = index, Random = random.Next() });
var result = Array.ConvertAll(shuffler.OrderBy(s => s.Random).ToArray(), s => s.Key);
```

The trouble is that the distribution of possible hands using this scheme isn't uniform. For example, with three cards, and using random numbers in the range 0-19, we get the distribution

[1,2,0]	1330
[0,2,1]	1330
[2,0,1]	1330
[2,1,0]	1140
[0,1,2]	1540
[1,0,2]	1330

In another chapter, the sieve schemes for generating prime numbers look very odd and not at all like the usual Sieve of Erastothenes.

There's also some code where we use the type of a default value by a runtime call of `GetType()` instead of using the generic type of the method.

ie instead of

```
static T ConvertTo<T>(object o, T defaultValue)
{
    return (T) Convert.ChangeType(o, defaultValue.GetType());
}
```

I don't see why we don't rather use

```
static T ConvertTo2<T>(object o, T defaultValue)
{
    return (T)Convert.ChangeType(o, typeof(T));
}
```

In summary, the book was a good way of seeing lots of examples of LINQ syntax and its uses, but I think I could have done just as well with smaller examples.

1. [http://www.amazon.co.uk/LINQ-Unleashed-C-Paul-Kimmel/dp/0672329832/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1256327891&sr=1-1](http://www.amazon.co.uk/LINQ-Unleashed-C-Paul-Kimmel/dp/0672329832/ref=sr_1_1?ie=UTF8&s=books&qid=1256327891&sr=1-1)

---

### 3.10.7 Faster, faster (2009-10-23 20:03)

[1]excl:ensuring-compiled-body

A great article on a new macro for Franz Lisp which ensures that the body is compiled while still interacting correctly with the interpreted dynamic environment. It shows how the code to be compiled can be wrapped so that the correct values can be plugged in from the dynamic interpreted environment. A great example of the power of Common Lisp.

1. [http://www.franz.com/support/tech\\_corner/ensure-comp101409.lhtml](http://www.franz.com/support/tech_corner/ensure-comp101409.lhtml)

---

## 3.11 November

### 3.11.1 That's the way you do it (2009-11-03 20:05)

I've been meaning to write lots more about Clojure, and will do at some point soon; as I delve deeper into the language, it gets more and more interesting. There is one thing that stopped me making progress for a while, so I'd better mention that.

I wanted to step through the Clojure repl (read-eval-print-loop), particularly the compiler, at the instruction level to get a feel for what is really going on under the covers. I dutifully downloaded the 1.0.0 zip, unpacked it onto my desktop and used ant to build it, but couldn't figure out how to get the jdb debugger to show me the source as I stepped through it.

It turns out (and I expect you knew this), that the jvm can be started in a mode where it allows connections from exterior debuggers. To get this to work you'd run the repl using:

```
"c:\Program Files\Java\jdk1.6.0_14\bin\java.exe" -Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=4571 -classpath clojure.jar clojure.lang.Repl
```

and then connect jdb to it using:

```
"c:\Program Files\Java\jdk1.6.0_14\bin\jdb.exe" -connect com.sun.jdi.SocketAttach:hostname=localhost,port=4571
```

You can then set a breakpoint at entry into the repl:

```
main[1] stop in clojure.lang.Repl.main
```

```
main[1] run
```

You can then use the "use" command to set the path for the source finding

```
main[1] use /users/clive/desktop/clojure-1.0.0/src/jvm  
main[1] list
```

```
16  
17 public class Repl {  
18  
330
```

```
19 public static void main(String[] args) throws Exception {  
20 => main.legacy _repl(args);  
21 }  
22 }
```

The cool thing is that the Clojure code is compiled with source location information

```
main[1] stop in clojure.core $apply __3243.dolInvoke
```

```
main[1] cont
```

For such locations, we need to set the source search path elsewhere

```
main[1] use /users/clive/desktop/clojure-1.0.0/src/clj  
main[1] list  
386 (defn apply  
387 "Applies fn f to the argument list formed by prepending args to argseq.  
"  
388 {:arglists '([f args* argseq]) }  
389 [ #^clojure.lang.IFn f & args]  
390 => (. f (applyTo (spread args)))  
391  
392 (defn vary-meta  
393 "Returns an object of the same type and value as obj, with  
394 (apply f (meta obj) args) as its metadata."  
395 [obj f & args]
```

---

### **3.11.2 That's something you can't do without (2009-11-12 20:32)**

[1]Essentials of Programming Languages by Friedman and Wand

A great book introducing the concepts of programming languages, including such things call-by-value/call-by-reference/call-by-name/call-by-need, type checking, type interface and modularization, all by means of interpreters for the various languages which are implemented in Scheme. The book covers various translations on the interpreters to make them more efficient, techniques like registerization (converting a series of tail recursion functions into a set of blocks connected via gotos with the arguments passed in global variables), CPS transformations and the use of trampoline techniques to move computation off the control stack into the heap. This latter technique was recently covered on the [2]B # blog and is used in Clojure to prevent stack overflow.

I also think that you don't understand a language until you implement it, and the idea of using Scheme interpreters to do the implementation is a powerful idea. Plenty of really good exercises too.

1. [http://www.amazon.co.uk/Essentials-Programming-Languages-third-Friedman/dp/0262062798/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1258055330&sr=1-1](http://www.amazon.co.uk/Essentials-Programming-Languages-third-Friedman/dp/0262062798/ref=sr_1_1?ie=UTF8&s=books&qid=1258055330&sr=1-1)
  2. <http://community.bartdesmet.net/blogs/bart/archive/2009/11/08/jumping-the-trampoline-in-c-stack-friendly-recursion.aspx>
- 

### **3.11.3 You can depend on this (2009-11-12 20:35)**

[1]Dependency Injection by Dhanji Prasanna

A great book on dependency injection with good coverage of Spring and Guice. It started from the beginning with a discussion of the motivation behind dependency injection, and then

covered the difference between the setter and constructor injection. It covered the various configuration styles which range from the fluid programming style of Guice to the XML based nature of Spring. The book had good coverage of scope and lifecycle management and then covered the container based support for interception and proxying. The book was mainly about java based containers, but there was only a small amount of code which could all be followed by a C # programmer.

- 
1. [http://www.amazon.co.uk/Dependency-Injection-Examples-Java-Ruby/dp/193398855X/ref=sr\\_1\\_1?ie=UTF8&qid=1258055825&sr=8-1-fkmr0](http://www.amazon.co.uk/Dependency-Injection-Examples-Java-Ruby/dp/193398855X/ref=sr_1_1?ie=UTF8&qid=1258055825&sr=8-1-fkmr0)

### **3.11.4 How many dwarves? (2009-11-12 20:36)**

[1]Coders At Work by Peter Seibel

Great interviews of "some of the top programmers" of our time. I really enjoyed the interviews of Brendan Eich, Douglas Crockford, and Simon Peyton-Jones. Peter Seibel has done a lot of programming in Common Lisp and he neatly got some of the interviewees talking about their connection with the language. Guy Steele had some interesting points to make.

Another person in the book is Ken Thompson, who has been involved in the new language that Google have just released. There's good coverage on this [2]blog.

- 
1. [http://www.amazon.co.uk/Coders-Work-Reflections-Craft-Programming/dp/1430219483/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1258057565&sr=1-1](http://www.amazon.co.uk/Coders-Work-Reflections-Craft-Programming/dp/1430219483/ref=sr_1_1?ie=UTF8&s=books&qid=1258057565&sr=1-1)
  2. [http://scienceblogs.com/goodmath/2009/11/googles\\_new\\_language\\_go.php](http://scienceblogs.com/goodmath/2009/11/googles_new_language_go.php)

### **3.11.5 What a lot of developers! (2009-11-27 20:57)**

After many years of reading about it, I eventually got to go to the recent [1]Microsoft PDC. Unfortunately, I didn't get to go as a delegate, but as an exhibitor on the Red Gate booth. It was amazing to see so many developers in one place, and we got to give lots of demonstrations of our product and field many interesting questions.

I also got to attend the two Keynotes. The first concerned Azure and Microsoft's commercial service which will be online in February. The second concerned Windows 7 and Silverlight 4. Silverlight seems to getting closer to a platform that supersedes technologies like ClickOnce for delivering applications onto corporate PCs.

I did get to attend two talks. [2]F # for Asynchronous and Parallel Programming by Luke Hoban and [3]Rx: Reactive Extensions for .NET. Both talks were very interesting. F # has been an interest of mine for a long time and it's good seeing it as a first class language in Visual Studio 2010. The talk covered asynchronous workflows with examples that downloaded data from the cloud, and moved onto the parallel extensions. Reactive functional programming has been an interesting area for some time, and seeing some of these ideas making their way into C # is very encouraging.

I have started to watch some of the other pdc talks which are available [4]here on video. The most interesting so far have been the talks by Stephen Toub on support for concurrency via the new Task class and extended ThreadPool ([5]ManyCore and .NET 4) and the [6]panel discussion of the future of programming.

1. <http://microsoftpdc.com/Videos>
2. <http://microsoftpdc.com/Sessions/FT20>
3. <http://microsoftpdc.com/Sessions/VTL04>
4. <http://microsoftpdc.com/Videos>
5. <http://microsoftpdc.com/Sessions/FT03>
6. <http://microsoftpdc.com/Sessions/FT52>

### **3.11.6 Oh, I say! (2009-11-29 10:45)**

[1]The Joy Of X: How algebra shapes our daily lives by Michael Willers

An interesting layman's introduction to algebra, including a brief history of algebra and full of interesting little problems together with their solutions.

1. [http://www.amazon.co.uk/gp/product/0550105239/ref=s9\\_sima\\_gw\\_s1\\_p14\\_i1?pf\\_rd\\_m=A3P5R0KL5A1OLE&pf\\_rd\\_s=central-1&pf\\_rd\\_r=1VHRJQA90KMRHNA4DVDR&pf\\_rd\\_t=101&pf\\_rd\\_p=](http://www.amazon.co.uk/gp/product/0550105239/ref=s9_sima_gw_s1_p14_i1?pf_rd_m=A3P5R0KL5A1OLE&pf_rd_s=central-1&pf_rd_r=1VHRJQA90KMRHNA4DVDR&pf_rd_t=101&pf_rd_p=)
- 

### **3.11.7 Google that (2009-11-29 10:50)**

[1]Planet Google: How one company is transforming our lives by Randall Stross

This book examines the company that currently dominates much of the web, looking at how it became so successful by the invention of the "algorithm" and the many technologies that it has attempted to develop (some successfully and some failures). An interesting analysis of where the company is going in the future.

1. [http://www.amazon.co.uk/Planet-Google-Company-Transforming-Lives/dp/1843549824/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1259491590&sr=1-1](http://www.amazon.co.uk/Planet-Google-Company-Transforming-Lives/dp/1843549824/ref=sr_1_1?ie=UTF8&s=books&qid=1259491590&sr=1-1)
- 

### **3.11.8 Burning bright (2009-11-29 10:58)**

[1]Java 2 v5.0 Tiger by Herbert Schildt

An interest in Clojure has got me spending more time with the Java platform. While I was in Cambridge library I came across this book, which is a brief introduction to the features introduced in the tiger release of the Java platform, in particular generics. Java generics differ a great deal from the generics of .Net and this book offered a very brief introduction to their major features, covering things like upper and lower bounds, wildcards, bridge methods and erasure. I'd have liked a lot more depth, but as a quick introduction, the book served its purpose.

- 
1. [http://www.amazon.co.uk/Java-v5-0-Tiger-New-Features/dp/0072258543/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1259491940&sr=1-1](http://www.amazon.co.uk/Java-v5-0-Tiger-New-Features/dp/0072258543/ref=sr_1_1?ie=UTF8&s=books&qid=1259491940&sr=1-1)

### **3.11.9 Negotiate only once (2009-11-30 20:00)**

The [1]Security Now podcast, episode #223, had a great discussion of a vulnerability in TLS. It was kind of difficult to get a real feel for the exploit from the verbal description. There's a pdf available [2]here which covers the exploit in more detail.

- 
1. <http://www.grc.com/securitynow.htm>
  2. [http://extendedsubset.com/Renegotiating\\_TLS.pdf](http://extendedsubset.com/Renegotiating_TLS.pdf)

## **3.12 December**

### **3.12.1 The eagle has landed (2009-12-14 19:37)**

[1]Finding Moonshine by Marcus Du Sautoy

Another book on symmetry, leading into group theory and progressing up to groups like the Monster. That side of the material is also covered by books such as Symmetry and the Monster. The really interesting part of the material was the autobiographical details about the author and his journey towards group theory. This material covers meetings with some of the interesting characters of the field such as [2]John H. Conway, with short interesting stories about them.

- 
1. [http://www.amazon.co.uk/Finding-Moonshine-Mathematicians-Journey-Symmetry/dp/0007214626/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1260819344&sr=8-1](http://www.amazon.co.uk/Finding-Moonshine-Mathematicians-Journey-Symmetry/dp/0007214626/ref=sr_1_1?ie=UTF8&s=books&qid=1260819344&sr=8-1)
  2. [http://en.wikipedia.org/wiki/John\\_Horton\\_Conway](http://en.wikipedia.org/wiki/John_Horton_Conway)
- 

### **3.12.2 Attack is the best defense (2009-12-14 19:40)**

[1]The art of computer virus research and Defense by Peter Szor

I was in Cambridge public library when I noticed this book, and I'm glad I took it out. Funnily enough, it also mentioned John H Conway, with reference to self-replicating structures, in the early chapters. The book covers all kinds of details about virus writing, propagation and detection with lots of details with a fair bit of x86 assembler on display. For example, there are many diagrams of the buffer overflow and format string attacks against the stack of a machine, and the book lists many methods that virus writers use to make their code undetectable. It covers techniques for cloaking the virus inside executable files, and covers propagation across a network.

A really interesting book, with perhaps a little too much information on DOS virus code.

- 
1. [http://www.amazon.co.uk/Art-Computer-Virus-Research-Defense/dp/0321304543/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1260819515&sr=1-1](http://www.amazon.co.uk/Art-Computer-Virus-Research-Defense/dp/0321304543/ref=sr_1_1?ie=UTF8&s=books&qid=1260819515&sr=1-1)
-

### 3.12.3 Can't go under it and can't go over it (2009-12-28 22:09)

There's a great trick in [1]Real World Functional Programming for getting construction of generic classes to work without the need for explicitly giving the types. If we have a class like the following:

```
class MyType<TX, TY>
{
    TX m_X;
    TY m_Y;
    public MyType(TX x, TY y)
    {
        m_X = x;
        m_Y = y;
    }
}
```

we'd like to be able to use it to assign a new instance to a variable that is implicitly typed. However,

```
var x = new MyType(2, "hello");
fails as C # requires that the generic type arguments are provided.
```

C # does allow the inference of types at the callsite of a generic method. Hence the trick is to define such a method, in say a static class.

```
static class MyType
{
    static public MyType<TX, TY> Create<TX, TY>(TX x, TY y)
    {
        return new MyType<TX, TY>(x, y);
    }
}
```

which we can then use in code like the following

```
var x = MyType.Create(2, "hello");
and hence avoid needing to give the types explicitly.
```

1. [http://www.amazon.co.uk/Real-World-Functional-Programming-Examples/dp/1933988924/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1262037560&sr=8-1](http://www.amazon.co.uk/Real-World-Functional-Programming-Examples/dp/1933988924/ref=sr_1_1?ie=UTF8&s=books&qid=1262037560&sr=8-1)

---

### 3.12.4 Bring your own boxes (2009-12-28 22:23)

The other day, I was looking through the code of the .NET framework code for EqualityComparer<T> in [1]Reflector. This type contains some rather interesting code for constructing a Default instance for each generic instantiation, and it took a while to dig out an [2]old blog post that describes the technique that is used.

```
private static EqualityComparer<T> CreateComparer()
{
    Type c = typeof(T);
    if (c == typeof(byte))
    {
        return (EqualityComparer<T>) new ByteEqualityComparer();
    }
    if (typeof(IEquatable<T>).IsAssignableFrom(c))
    {
        return (EqualityComparer<T>)
            typeof(GenericEqualityComparer<int>).TypeHandle.CreateInstanceForAnotherGeneric
            Parameter(c);
    }
    if (c.IsGenericType && (c.GetGenericTypeDefinition() ==
        typeof(Nullable<>)))
    {
        Type type2 = c.GetGenericArguments()[0];
        if (typeof(IEquatable<>).MakeGenericType(new Type[] { type2
    }).IsAssignableFrom(type2))
        {
            return (EqualityComparer<T>)
                typeof(NullableEqualityComparer<int>).TypeHandle.CreateInstanceForAnotherGeneri
                cParameter(type2);
        }
    }
    return new ObjectEqualityComparer<T>();
}
```

1. <http://www.red-gate.com/products/reflector/index.htm>
  2. <http://blogs.msdn.com/bclteam/archive/2005/03/15/396483.aspx>
-

# 2010

## 4.1 January

### 4.1.1 A journey through the maze of twisty passages (2010-01-03 12:14)

I've been playing around with Clojure for some time, but decided that over the holidays it would be good to delve more deeply into the internal workings of the system. Clojure is interesting in that the compiler is implemented in java, but a lot of the other code is implemented in Clojure itself; as we'll see the system loads this code dynamically when the environment is set up.

It seemed that the best way to start to understand the system, would be to see how the REPL, the read-eval-print loop, is implemented in the system.

Well, first the Repl.java file contains a main method for running the main loop

```
public static void main(String[] args) throws Exception {  
    main.legacy_repl(args);
```

legacy\_repl is defined elsewhere to be the code

```
public static void legacy_repl(String[] args) throws Exception {  
    REQUIRE.invoke(CLOJURE_MAIN);  
    LEGACY_REPL.invoke(RT.seq(args));  
}
```

The second statement, the invoke is just the standard way of calling into a Clojure defined Var which the system gets from the standard Clojure namespace.

```
final static private Var LEGACY_REPL = Var.intern(CLOJURE_MAIN_NS, Symbol.create("legacy-repl"));
```

This legacy-repl symbol is defined in main.clj

```
(defn- legacy-repl
  "Called by the clojure.lang.Repl.main stub to run a repl with args
  specified the old way"
  [args]
  (let [[inits [sep & args]] (split-with (complement # {"-"} ) args)]
    (repl-opt (concat ["-r"] args) (map vector (repeat "-i") inits))))
```

This eventually calls into the repl function that is defined in the same file, and then into the eval function that is defined in core.clj

```
(defn eval
  "Evaluates the form data structure (not text!) and returns the result."
  [form] (. clojure.lang.Compiler (eval form)))
```

which uses the eval method on the Compiler class. We'll take a look at that later, but let's first look at one of the other functions that were called earlier in the sequence.

What we need to look at is the call,  
REQUIRE.invoke(CLOJURE\_MAIN);

where

```
final static private Var REQUIRE = Var.intern(RT.CLOJURE_NS, Symbol.create("require"));
```

Require is defined in core.clj as a call into the load-libs function which eventually goes through to the function load which wraps the runtime function  
(clojure.lang.RT/load (.substring path 1))

This method loads a class corresponding to the module

```
try {
  Var.pushThreadBindings(
    RT.map(CURRENT_NS, CURRENT_NS.deref()),
    WARN_ON_REFLECTION, WARN_ON_REFLECTION.deref()));
  loaded = (loadClassForName(scriptbase.replace('/', '.') + LOADER_SUFFIX) != null);
}
finally {
  Var.popThreadBindings();
}
```

Where did the compiled code from require come from? There are a number of class files that are generated as part of the build.

```
<target name="compile-clojure" depends="compile-java"
description="Compile Clojure sources.">
<java classname="clojure.lang.Compile"
classpath=" ${build}: ${cljsrc}">
<sysproperty key="clojure.compile.path" value=" ${build}"/>
<arg value="clojure.core"/>
<arg value="clojure.main"/>
```

```

<arg value="clojure.set"/>
<arg value="clojure.xml"/>
<arg value="clojure.zip"/>
<arg value="clojure.inspector"/>
</java>
</target>

```

Require is in core \$ \$require \_\_5049.class

The RT.java file contains the method dolInit, which is responsible for loading much of the system into the memory.

```

static void dolInit() throws Exception {
load("clojure/core");
load("clojure/zip", false);
load("clojure/xml", false);
load("clojure/set", false);

```

```

Var.pushThreadBindings(
RT.map(CURRENT_NS, CURRENT_NS.deref()),
WARN_ON_REFLECTION, WARN_ON_REFLECTION.deref()));
try {
Symbol USER = Symbol.create("user");
Symbol CLOJURE = Symbol.create("clojure.core");

Var in_ns = var("clojure.core", "in-ns");
Var refer = var("clojure.core", "refer");
in_ns.invoke(USER);
refer.invoke(CLOJURE);
maybeLoadResourceScript("user.clj");
}
finally {
Var.popThreadBindings();
}
}

```

If we run the system using the command line

```

C:\Users\Clive\Desktop\clojure-1.0.0>java -cp classes clojure.lang.Repl
Exception in thread "main" java.lang.ExceptionInInitializerError
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Unknown Source)
at clojure.lang.RT.loadClassForName(RT.java:1499)
at clojure.lang.RT.load(RT.java:385)
at clojure.lang.RT.load(RT.java:367)
at clojure.lang.RT.dolInit(RT.java:402)
at clojure.lang.RT.<clinit>(RT.java:288)
at clojure.lang.Namespace.<init>(Namespace.java:32)
at clojure.lang.Namespace.findOrCreate(Namespace.java:117)
at clojure.main.<clinit>(main.java:21)

```

```
at clojure.lang.Repl.main(Repl.java:20)
Caused by: java.lang.NullPointerException
at java.util.Properties $LineReader.readLine(Unknown Source)
at java.util.Properties.load0(Unknown Source)
at java.util.Properties.load(Unknown Source)
at clojure.core $fn __5774.invoke(core.clj:4092)
at clojure.core __init.load(Unknown Source)
at clojure.core __init.<clinit>(Unknown Source)
... 11 more
```

we can see the dolnit being used to start the system, called as a static constructor on the RT class.

Removing some jar files to allow us to play with the code  
java -cp classes;srcclj clojure.lang.Repl

I modified main.clj to see if I could get it to recompile.... that didn't seem to happen. So running things under the java debugger:

```
C:UsersCliveDesktopclojure-1.0.0>java -Xdebug -Xrunjdwp:transport=dt
_socket,server=y,address=4571 -cp classes;srcclj clojure.lang.Repl
```

```
C:UsersCliveDesktopclojure-1.0.0>"c:Program FilesJavajdk1.6.0 _14binjdb.exe" -connect
com.sun.jdi.SocketAttach:hostname=localhost,port=4571
```

```
main[1] stop in clojure.lang.RT.lastModified
main[1] use /users/clive/desktop/clojure-1.0.0/src/jvm
main[1] list
```

Stepping through, we can see that the code in  
static public void load(String scriptbase, boolean failIfNotFound) throws Exception {

loads the core.clj file as a resource script if it is modified later than the class file.

We can test this by adding an extra function at the end.  
(defn extra [] "boo")

```
C:UsersCliveDesktopclojure-1.0.0>java -cp classes;srcclj clojure.lang.Repl
Clojure 1.0.0-
user=> (extra)
"boo"
```

The last modified is being tried on the init class corresponding to the file.

```
main[1] print url
```

```
url = "file:/C:/Users/Clive/Desktop/clojure-1.0.0/classes/clojure/core __init.class"
```

So we'll have to compile up clojure.core manually.

```
C:\Users\Clive\Desktop\clojure-1.0.0>java -Dclojure.compile.path=classes -cp classes;srcclj  
clojure.lang.Compile clojure.core  
Compiling clojure.core to classes
```

The main method of Compile.java basically compiles using the  
compile.invoke(Symbol.intern(lib));

where

```
private static final Var compile = RT.var("clojure.core", "compile");
```

This calls the

```
public static Object compile(Reader rdr, String sourcePath, String sourceName) throws Exception {  
in Compiler.java
```

Let's debug into it. We can simply touch the core.clj file and then

```
main[1] stop in clojure.lang.Compiler.compile
```

We eventually come around to the code that does the compilation

```
for(Object r = LispReader.read(pushbackReader, false, EOF, false); r != EOF;  
r = LispReader.read(pushbackReader, false, EOF, false))  
{  
LINE _AFTER.set(pushbackReader.getLineNumber());  
Expr expr = analyze(C.EVAL, r);  
fn.keywords = (IPersistentMap) KEYWORDS.deref();  
fn.vars = (IPersistentMap) VARS.deref();  
fn.constants = (PersistentVector) CONSTANTS.deref();  
expr.emit(C.EXPRESSION, fn, gen);  
expr.eval();  
LINE _BEFORE.set(pushbackReader.getLineNumber());  
}
```

Inside the LispReader

```
Step completed: "thread=main", clojure.lang.LispReader.read(), line=122 bci=0  
122 int ch = r.read();
```

```
main[1] list  
118 try
```

```

119 {
120 for(; ;)
121 {
122 => int ch = r.read();
123
124 while(isWhitespace(ch))
125 ch = r.read();
126
127 if(ch == -1)
main[1] step
>

```

and eventually we go into code that looks up the macro characters

```

>
Step completed: "thread=main", clojure.lang.LispReader.read(), line=142 bci=73
142 IFn macroFn = getMacro(ch);

```

```

main[1] list
138 return null;
139 return n;
140 }
141
142 => IFn macroFn = getMacro(ch);
143 if(macroFn != null)
144 {
145 Object ret = macroFn.invoke(r, (char) ch);
146 if(RT.suppressRead())
147 return null;

```

The getMacro looks up the characters in the macro array in LispReader.

```

static
{
macros['"'] = new StringReader();
macros[';'] = new CommentReader();
macros['"'] = new WrappingReader(QQUOTE);
macros['@'] = new WrappingReader(DEREF);//new DerefReader();
macros['^'] = new WrappingReader(META);
macros['"'] = new SyntaxQuoteReader();
macros['`'] = new UnquoteReader();
macros['('] = new ListReader();
macros[')'] = new UnmatchedDelimiterReader();
macros['['] = new VectorReader();
macros[''] = new UnmatchedDelimiterReader();
macros['{'] = new MapReader();
macros['}'] = new UnmatchedDelimiterReader();
macros['\'] = new CharacterReader();

```

```
macros[' %'] = new ArgReader();
macros[' #'] = new DispatchReader();
```

The first non-comment form we read is

```
main[1] print r
r = "(ns clojure.core)"
```

This is passed into  
return analyzeSeq(context, (ISeq) form, name);

This form is macroexpanded  
4,482 => Object me = macroexpand1(form);  
4,483 if(me != form)  
4,484 return analyze(context, me, name);

resulting in the form

```
main[1] print me
me = "(do (clojure.core/in-ns (quote clojure.core)))"
```

and we go around the analyze/analyseSeq code again. This time, do isn't a macro and hence we drop through into the code that checks for an inline function and then tries some other possibilities.

```
4,492 IParser p;
4,493 => if(op.equals(FN))
4,494 return FnExpr.parse(context, form, name);
4,495 else if((p = (IParser) specials.valAt(op)) != null)
4,496 return p.parse(context, form);
4,497 else
4,498 return InvokeExpr.parse(context, form);
```

specials contains the special forms, things that the compiler needs to know how to handle. This is the thing that has always impressed me about Lisp – the set of core functions that the compiler needs to handle is really small. Most control constructs can be implemented using macros and some underlying runtime functions.

```
static final public IPersistentMap specials = PersistentHashMap.create(
DEF, new DefExpr.Parser(),
LOOP, new LetExpr.Parser(),
RECUR, new RecurExpr.Parser(),
IF, new IfExpr.Parser(),
LET, new LetExpr.Parser(),
LETFN, new LetFnExpr.Parser(),
DO, new BodyExpr.Parser(),
```

```

FN, null,
QUOTE, new ConstantExpr.Parser(),
THE_VAR, new TheVarExpr.Parser(),
DOT, new HostExpr.Parser(),
ASSIGN, new AssignExpr.Parser(),
TRY, new TryExpr.Parser(),
THROW, new ThrowExpr.Parser(),
MONITOR_ENTER, new MonitorEnterExpr.Parser(),
MONITOR_EXIT, new MonitorExitExpr.Parser(),
CATCH, null,
FINALLY, null,
NEW, new NewExpr.Parser(),
_AMP_, null
);

```

In our case, Do is a special form that we need to handle,

```

4,496 return p.parse(context, form);
main[1] print context
context = "EVAL"
main[1] print form
form = "(do (clojure.core/in-ns (quote clojure.core)))"
main[1] print p
p = "clojure.lang.Compiler $BodyExpr $Parser@1530551"
main[1] next
3,854 static class Parser implements IParser {
3,855 public Expr parse(C context, Object frms) throws Exception {
3,856 => ISeq forms = (ISeq) frms;
3,857 if(Util.equals(RT.first(forms), DO))
3,858 forms = RT.next(forms);
3,859 PersistentVector exprs = PersistentVector.EMPTY;
3,860 for(; forms != null; forms = forms.next())
3,861 {

```

This causes us to go back around into analyze with

```

4,286 return analyze(context, form, null);
main[1] print context
context = "EVAL"
main[1] print form
form = "(clojure.core/in-ns (quote clojure.core))"

```

This time we need to invoke this as an InvokeExpr as it is simply a function call.

```

4,498 return InvokeExpr.parse(context, form);

```

We need to analyse this and generate an InvokeExpr node

```

static public Expr parse(C context, ISeq form) throws Exception {
if(context != C.EVAL)
context = C.EXPRESSION;
Expr fexpr = analyze(context, form.first());

```

```

PersistentVector args = PersistentVector.EMPTY;
for(ISeq s = RT.seq(form.next()); s != null; s = s.next())
{
  args = args.cons(analyze(context, s.first()));
}

return new InvokeExpr((String) SOURCE.deref(), (Integer) LINE.deref(), tagOf(form), fexpr,
args);
}

```

We first analyze the function symbol

2,767 Expr fexpr = analyze(context, form.first());

4,307 return analyzeSymbol((Symbol) form);  
main[1] print form  
form = "clojure.core/in-ns"

The form gets resolved to a Var, which is registered and then returned as a VarExpr

4,605 => Object o = resolve(sym);  
4,606 if(o instanceof Var)  
4,607 {  
4,608 Var v = (Var) o;  
4,609 if(isMacro(v) != null)  
4,610 throw new Exception("Can't take value of a macro: " + v);  
4,611 registerVar(v);  
4,612 return new VarExpr(v, tag);  
4,613 }

We'll need to look more at the context argument, but for now we have seen how a tree representation is built from the source.

After we have finished parsing, we get back up to the compile function.

main[1] list  
4,948 Expr expr = analyze(C.EVAL, r);  
4,949 fn.keywords = (IPersistentMap) KEYWORDS.deref();

4,950 fn.vars = (IPersistentMap) VARS.deref();  
4,951 fn.constants = (PersistentVector) CONSTANTS.deref();  
4,952 => expr.emit(C.EXPRESSION, fn, gen);  
4,953 expr.eval();  
4,954 LINE.\_BEFORE.set(pushbackReader.getLineNumber());

```

4,955 }
4,956 //end of load
4,957 gen.returnValue();
main[1] print fn.vars
fn.vars = " { #'clojure.core/in-ns 1, #'clojure.core/ns 0 }"
main[1] print fn.constants
fn.constants = "[ #'clojure.core/ns #'clojure.core/in-ns clojure.core]"
main[1] print fn.keywords
fn.keywords = " { }"

```

We now want to emit the code for the tree which is rooted at a BodyExpr. This will generate code emitting all but the last form in a STATEMENT context.

```

public void emit(C context, FnExpr fn, GeneratorAdapter gen) {
for(int i = 0; i < exprs.count() - 1; i++)
{
Expr e = (Expr) exprs.nth(i);
e.emit(C.STATEMENT, fn, gen);
}
Expr last = (Expr) exprs.nth(exprs.count() - 1);
last.emit(context, fn, gen);
}

```

In our case, there is a single node, so we'll step into that. It is of type InvokeExpr.

```

2,725 public void emit(C context, FnExpr fn, GeneratorAdapter gen) {
2,726 => gen.visitLineNumber(line, gen.mark());
2,727 fexpr.emit(C.EXPRESSION, fn, gen);
2,728 gen.checkCast(IFN_TYPE);

```

We emit the fexpr, which is a VarExpr.

```

429 public void emit(C context, FnExpr fn, GeneratorAdapter gen) {
430 => fn.emitVar(gen, var);
431 gen.invokeVirtual(VAR_TYPE, getMethod);

```

which emits the position of the Var in the vars map

```

3,525 public void emitVar(GeneratorAdapter gen, Var var) {
3,526 => Integer i = (Integer) vars.valAt(var);
3,527 emitConstant(gen, i);

```

```

main[1] print vars
vars = " { #'clojure.core/in-ns 1, #'clojure.core/ns 0 }"

```

and the invokeVirtual is emitted which accesses the get() method

```
1,329 invokeInsn(Opcodes.INVOKEVIRTUAL, owner, method);
main[1] print owner
owner = "Lclojure/lang/Var;"
main[1] print method
method = "get()Ljava/lang/Object;"
```

We then generate the arguments and finally invoke the fn.

```
2,750 gen.invokeInterface(IFN _TYPE, new Method("invoke", OBJECT _TYPE, ARG _TYPES[Math.min(MAX_POSITIONAL_ARITY + 1,
```

Back in the compile function we then need to evaluate to get any side effects to happen. We do this by evaluating the form in the current system. This is one difference between Clojure and Lisp systems I have worked on in the past. They would maintain a compile-time environment to avoid the need to side-effect the current environment. This had advantages in that it allowed cross-compilation of source code. Common Lisp has all sorts of interesting mechanisms in the eval-when special form to control what side-effects happen at which point.

```
4,953 => expr.eval();
```

```
3,876 Object ret = null;
3,877 for(Object o : exprs)
3,878 {
3,879 Expr e = (Expr) o;
3,880 => ret = e.eval();
3,881 }
3,882 return ret;
```

```
2,707 public Object eval() throws Exception {
2,708 try
2,709 {
2,710 => IFn fn = (IFn) fexpr.eval();
2,711 PersistentVector args = PersistentVector.EMPTY;
2,712 for(int i = 0; i < args.count(); i++)
2,713 args = args.cons(((Expr) args.nth(i)).eval());
2,714 return fn.applyTo(RT.seq(args));
2,715 }
```

After looping around and code generating the rest of the forms, we end at code for generating the rest of the class initialization code. By this point, we have a large number of vars that need to be set up, as well as keywords and constants. These are referenced by the code generated above which is generated into a method named load.

The constants, including the vars that are referenced, are set up as static fields of the class. The static initializer then pushes the namespace, calls the load method, and then pops the thread bindings.

We'll ignore the next forms in core.clj

```
4,948 Expr expr = analyze(C.EVAL, r);
main[1] print r
r = "(def unquote)"
```

```
main[1] print r
r = "(def unquote-splicing)"
```

```
main[1] print r
r = "(def list (. clojure.lang.PersistentList creator))"
```

Defs are handled by a special parser

```
4,495 else if((p = (IParser) specials.valAt(op)) != null)
```

```
main[1] print p
p = "clojure.lang.Compiler $DefExpr $Parser@ec0962"
```

This generates us a new DefExpr.

```
363 }
364 => IPersistentMap mm = sym.meta();
365 Object source_path = SOURCE_PATH.get();
366 source_path = source_path == null ? "NO_SOURCE_FILE" : source_path;
367 mm = (IPersistentMap) RT.assoc(mm, RT.LINE_KEY, LINE.get()).assoc(RT.FILE_KEY, source_path);
368 Expr meta = analyze(context == C.EVAL ? context : C.EXPRESSION, mm);
369 return new DefExpr((String) SOURCE.deref(), (Integer) LINE.deref(),
```

The debugger doesn't show us the other important lines

```
v, analyze(context == C.EVAL ? context : C.EXPRESSION, RT.third(form), v.sym.name),
meta, RT.count(form) == 3);
```

which eventually generates a StaticFieldExpr for the body

```
749 return new StaticFieldExpr(line, c, sym.name);
main[1] print form
form = "(. clojure.lang.PersistentList creator)"
```

We'll move on to emitting java code for the def when we get to the emit  
4,952 expr.emit(C.EXPRESSION, fn, gen);

which does

```
315 public void emit(C context, FnExpr fn, GeneratorAdapter gen) {  
316 => fn.emitVar(gen, var);  
317 if(initProvided)  
318 {  
319 gen.dup();  
320 init.emit(C.EXPRESSION, fn, gen);  
321 gen.invokeVirtual(VAR_TYPE, bindRootMethod);
```

which then emits a StaticFieldExpr

```
977 public void emit(C context, FnExpr fn, GeneratorAdapter gen) {  
978 => gen.visitLineNumber(line, gen.mark());  
979  
980 gen.getStatic(Type.getType(c), fieldName, Type.getType(field.getType()));  
981 //if(context != C.STATEMENT)  
982 HostExpr.emitBoxReturn(fn, gen, field.getType());  
983 if(context == C.STATEMENT)
```

Now we need to look at how functions are emitted. Continuing for a bit we get to the compilation of

```
4,948 Expr expr = analyze(C.EVAL, r);  
main[1] print r  
r = "(def cons (fn* cons [x seq] (. clojure.lang.RT (cons x seq))))"
```

The analyse code generates a DefExpr into which it puts the result of analysing

```
Step completed: "thread=main", clojure.lang.Compiler.analyze(), line=4,293 bci=0  
4,293 if(form instanceof LazySeq)  
main[1] print form  
form = "(fn* cons [x seq] (. clojure.lang.RT (cons x seq)))"
```

which we parse as an FnExpr

```
4,493 if(op.equals(FN))  
4,494 => return FnExpr.parse(context, form, name);  
4,495 else if((p = (IParser) specials.valAt(op)) != null)  
4,496 return p.parse(context, form);  
4,497 else  
4,498 return InvokeExpr.parse(context, form);
```

The code generates a FnExpr and then processes each method

```
Step completed: "thread=main", clojure.lang.Compiler $FnExpr.parse(), line=2,963bci=420  
2,963 FnMethod f = FnMethod.parse(fn, (ISeq) RT.first(s));  
main[1] print s  
s = "(([x seq] (. clojure.lang.RT (cons x seq))))"
```

This parse method

```
3,637 private static FnMethod parse(FnExpr fn, ISeq form) throws Exception {  
3,638 //([args] body...)  
3,639 => IPersistentVector parms = (IPersistentVector) RT.first(form);  
3,640 ISeq body = RT.next(form);  
3,641 try  
3,642 {  
3,643 FnMethod method = new FnMethod(fn, (FnMethod) METHOD.deref());  
3,644 method.line = (Integer) LINE.deref();  
main[1] print form  
form = "([x seq] (. clojure.lang.RT (cons x seq)))"
```

generates a new FnMethod, sets up the argument variables and then parses the body.

```
3,695 LOOP _LOCALS.set(argLocals);  
3,696 method.argLocals = argLocals;  
3,697 => method.body = (new BodyExpr.Parser()).parse(C.RETURN, body);  
3,698 return method;
```

This has the name

```
main[1] print fn.name  
fn.name = "clojure.core $cons __4"
```

We can step through the compile method of this to see how the code is generated.

It is, however, easier to use a java decompiler to have a look at the generated code. We were compiling core.clj, and we saw that this would generate a class file core \_\_init.class. Decompiling this using [1]jd-gui we see the load method has the form.

```
public static void load()  
{  
const __3.setMeta((IPersistentMap)RT.map(new Object[] { const __4, "clojure/core.clj", const __5, const __6 }));  
const __7.setMeta((IPersistentMap)RT.map(new Object[] { const __4, "clojure/core.clj", const __5, const __8 }));  
Var tmp100 __97 = const __9; tmp100 __97  
.bindRoot(PersistentList.creator);  
  
tmp100 __97  
.setMeta((IPersistentMap)RT.map(new Object[] { const __4, "clojure/core.clj", const __5, const __10, const __11, const __12, const __13,  
"Creates a new list containing the items." }));  
Var tmp173 __170 = const __14;  
tmp173 __170.bindRoot(new core.cons __2206());  
tmp173 __170.setMeta((IPersistentMap)RT.map(new Object[] { const __4, "clojure/core.clj", const __5, const __15, const __11, const __16,  
const __13, "Returns a new seq where x is the first element and seq isn the rest." }));
```

The code for cons \_\_2206 is

```
public class core $cons __2206 extends AFunction
{
    public Object invoke(Object x, Object seq)
        throws Exception
    {
        x = null; seq = null; return RT.cons(x, seq);
    }
}
```

That code looks wrong of course... we don't really blast the arguments before the call to the RT.cons function. The decompiler is not decompiling correctly. We can see this using javap, which shows that we load the incoming arguments on to the stack, before blasting the old values, and it is these stack values that are passed into the RT.cons method.

```
public java.lang.Object invoke(java.lang.Object, java.lang.Object) throws java.lang.Exception;
Code:
0:  aload _1
1:  aload _2
2:  aconst _null
3:  astore _1
4:  aconst _null
5:  astore _2
6:      invokestatic #20;      //Method      clojure/lang/RT.cons:(Ljava/lang/Object;-
Ljava/lang/Object;)Lclojure/lang/ ISeq;
9:  areturn
```

jd cannot decompile the static constructor which has the following code for setting up const \_\_14, which is then assigned to the Var, tmp173 \_\_170, into which an instance of the function class is assigned.

```
// 246: ldc_w 6584
// 249: ldc_w 6651
// 252: invokestatic 6590 clojure/lang/RT:var (Ljava/lang/String;Ljava/lang/String;)Lc-
lojure/lang/Var;
// 255: checkcast 12 clojure/lang/Var
// 258: putstatic 89 clojure/core __init:const __14 Lclojure/lang/Var;
```

Using javap to decompile it,

```
C:\Users\Clive\Desktop\clojure-1.0.0\classes\clojure>"c:\Program Files\Java\jdk1.6.0 _14\bin\javap.exe" -c core __init > foo.txt
```

We see the following

```
246: ldc_w #6584; //String clojure.core
249: ldc_w #6651; //String cons
252: invokestatic #6590;      //Method      clojure/lang/RT.var:(Ljava/lang/String;L-
java/lang/String;)Lclojure/lang/V ar;
```

```
255: checkcast #12; //class clojure/lang/Var  
258: putstatic #89; //Field const __14:Lclojure/lang/Var;
```

with the runtime method having the code

```
static public Var var(String ns, String name) {  
    return Var.intern	Namespace.findOrCreate(Symbol.intern(null, ns),  
        name));  
}
```

Hence, the instance of the class representing the function is placed into the Var that is found using this method in the runtime.

We need to have a look at nested lambda expressions, but there's not time to do that now.

It was an interesting exercise, and I now have some idea of what is happening under the surface. Having worked on several Lisp systems in the past, it was slightly disappointing that more of the compiler wasn't written in Clojure; keeping the architecture-specific code small in those systems, allowed the system to be ported much more easily to other platforms.

1. <http://java.decompiler.free.fr/?q=jdgui#downloads>

---

#### 4.1.2 Sorry, we're closed (over) (2010-01-04 16:45)

The next thing to look at is how Clojure handles closures, and the typical test example for that is

```
(ns clojure.test)  
(defn f [x] (fn [y] (* x y)))
```

Putting that into a file, test.clj, next to the core.clj file and compiling it using  
java -Dclojure.compile.path=classes -cp classes;srcclj clojure.lang.Compile clojure.test

This results in three class files. The top level file, test \_\_init.class is the usual load file which is responsible for setting up the binding of the various things we define.

```
public class test __init  
{  
    public static final Var const __0 = (Var)RT.var("clojure.core", "ns");  
    public static final Var const __1 = (Var)RT.var("clojure.core", "in-ns");  
    public static final AFn const __2 = (AFn)Symbol.create(null, "clojure.test");  
    public static final Var const __3 = (Var)RT.var("clojure.core", "refer");  
    public static final AFn const __4 = (AFn)Symbol.create(null, "clojure.core");  
    public static final Var const __5 = (Var)RT.var("clojure.core", "defn");  
    public static final Var const __6 = (Var)RT.var("clojure.test", "f");
```

```

public static final Object const __7 = Keyword.intern(Symbol.create(null, "file"));
public static final Object const __8 = Keyword.intern(Symbol.create(null, "line"));
public static final Object const __9 = Integer.valueOf(3);
public static final Object const __10 = Keyword.intern(Symbol.create(null, "arglists"));
public static final Object const __11;
public static final Var const __12;

public static void load()
{
  ((IFn)const __1.get()).invoke(const __2);
  Var tmp38 __35 = const __6;
  tmp38 __35.bindRoot(new test.f __1());
  tmp38 __35.setMeta((IPersistentMap)RT.map(new Object[] { const __7, "clojure/test.clj", const __8, const __9, const __10, const __11 }));
}

// ERROR //
static
{
// jd fails to decompile this part of the code
}
}

```

The function, f, is defined in test \$f \_\_1

```

public class test $f __1 extends AFunction
{
  public static final Var const __0 = (Var)RT.var("clojure.core", "fn");
  public Object invoke(Object x) throws Exception { return new fn __3(x); }
}

```

And the inner function is defined in

```

public class test $f __1 $fn __3 extends AFunction
{
  public static final Var const __0 = (Var)RT.var("clojure.core", "*");
  public Object x;
  public test $f __1 $fn __3(Object paramObject)
  {
    this.x = paramObject; }
  public Object invoke(Object y) throws Exception { y = null; return Numbers.multiply(this.x, y); }
}

```

Remember that the setting of y to null in the invoke function is a decompiler error. Hence we see that the closure is represented by an instance of a class, which has fields corresponding to the closed over variables. This is a fairly standard representation for the case where the locals are immutable and hence we don't need to home the variable just once in a class corresponding to the lexical level.

Local bindings for the FnExpr are maintained in a field named closes in the FnExpr instance.  
 IPersistentMap closes = PersistentHashMap.EMPTY;

This is assigned in the static closeOver method  
 static void closeOver(LocalBinding b, FnMethod method) {  
 if(b != null & & method != null)

```

{
if(RT.get(method.locals, b) == null)
{
method.fn.closes = (IPersistentMap) RT.assoc(method.fn.closes, b, b);
closeOver(b, method.parent);
}
else if(IN_CATCH_FINALLY.deref() != null)
{
method.localsUsedInCatchFinally = (PersistentHashSet) method.localsUsedInCatchFinally.cons(b.i-
dx);
}
}
}
}

```

The compile method of FnExpr then takes care of adding the extra arguments allowing the closed over values to be passed down into the closure instance constructor.

---

#### **4.1.3 How tight is the binding? (2010-01-05 21:51)**

Clojure has a notion of binding that is close to the behaviour of special variables of Common Lisp. Using def, and associated macros such as defn, one can set up a location that is associated with a symbol and which can take thread specific values.

Define a Var associated with the symbol boo.

```
user=> (def boo 10)
#'user/boo
```

Evaluating the symbol boo, looks for the associated Var and then gets the value of the Var.

```
user=> boo
10
```

Define a function, bound to the symbol foo which returns the value of the Var associated with boo.

```
user=> (defn foo [] boo)
#'user/foo
```

Note that defn is just a macro that expands into a use of def

```
user=> (macroexpand '(defn foo[] boo))
(def foo (clojure.core/fn ([] boo)))
```

Define a function that calls foo which in turn returns the value of the Var associated with boo.

```
user=> (defn bar [] (foo))
#'user/bar
user=> (bar)
10
```

Now we can use the binding macro to rebind the value of the Var cell associated with boo during the evaluation of bar.

```
user=> (binding [boo 30] (bar))  
30
```

This is completely different from what a let binding does.

```
user=> (let [boo 30] (bar))  
10
```

But, Var are not just used for values like integers, but are also used to hold the function values too.

```
user=> (binding [foo (fn [] 50)] (bar))  
50
```

So how does this all work then? Well, the foo function looks something like this:

```
public class test $foo __1 extends AFunction  
{  
    public static final Var const __0 = (Var)RT.var("clojure.test", "boo");  
    public Object invoke() throws Exception { return const __0.get();}  
}
```

As this shows, accessing the value is achieved by calling the get() method of the Var object.

Binding is a macro that is responsible for setting up new mappings.

```
user=> (macroexpand '(binding [boo 30] (bar)))  
(let*  
  []  
  (clojure.core/push-thread-bindings (clojure.core/hash-map #'boo 30))  
  (try (bar) (finally (clojure.core/pop-thread-bindings))))
```

The Var class has a static ThreadLocal field that contains a chain of Frames for that particular thread.

```
static ThreadLocal<Frame> dvals = new ThreadLocal<Frame>() {  
    protected Frame initialValue() {  
        return new Frame();  
    }  
};
```

A Frame is an object with three fields.

Associative bindings; // All bindings in the chain of Frames – if a Var is bound its latest value for the thread is found here

Associative frameBindings; // The bindings set for the current frame

Frame prev; // The next Frame in the chain

Pushing new bindings involves getting the current bindings, merging in the new bindings and then adding a Frame containing this information to the thread local variable. The AtomicInteger in the field count is used to optimise the case when there are no bindings for the given Var on any thread.

```
public static void pushThreadBindings(Associative bindings) {  
    Frame f = dvals.get();  
    Associative bmap = f.bindings;
```

```

for(ISeq bs = bindings.seq(); bs != null; bs = bs.next())
{
  IMapEntry e = (IMapEntry) bs.first();
  Var v = (Var) e.key();
  v.validate(v.getValidator(), e.val());
  v.count.incrementAndGet();
  bmap = bmap.assoc(v, new Box(e.val()));
}
dvals.set(new Frame(bindings, bmap, f));
}

```

Note that the value stored in the map is of type `Box`, an instance with a single value field. This is required as `Var` instances support a `set` function which changes the value of the last binding.

```

user=> (def foo 10)
 #'user/foo
user=> foo
10
user=> (defn getFoo[] foo)
 #'user/getFoo
user=> (getFoo)
10
user=> (binding [foo 30] (var-set #'foo 50) (getFoo))
50
user=> foo
10

```

The `Var` `get` method is the same as the `deref` method

```

final public Object deref() {
  Box b = getThreadBinding();
  if(b != null)
    return b.val;
  if(hasRoot())
    return root;
  throw new IllegalStateException(String.format("Var %s/ %s is unbound.", ns, sym));
}

```

This uses `getThreadBinding` to find the latest frame

```

final Box getThreadBinding() {
  if(count.get() > 0)
  {
    IMapEntry e = dvals.get().bindings.entryAt(this);
    if(e != null)
      return (Box) e.val();
  }
  return null;
}

```

If there is no binding on the thread, then we check for a root binding, stored in the field `root`. The `dvals` value itself is stored in the `root` field to signal that there is no root binding. otherwise the `root` field contains the root value itself.

```

final public boolean hasRoot() {

```

```
return root != dvals;
}
```

In the code to set up the bindings, there was also a call to validate. Vars can have a validation function associated with them that can be used to verify values that are being set into them. For example, we can ensure that the value in the Var associated with myFunc is always a function; attempting to bind it to an integer causes the binding to throw an exception.

```
user=> (defn myFunc [])
#'user/myFunc
user=> (set-validator! #'myFunc fn?)
nil
user=> (def myFunc 10)
#<CompilerException java.lang.IllegalStateException: Invalid reference state (NO _SOURCE
_FILE:153)>
user=> myFunc
#<user $myFunc _ _4974 user $myFunc _ _4974@19902f7>
```

Note that we can get rid of the validation function

```
user=> (set-validator! #'myFunc nil)
nil
user=> (def myFunc 10)
#'user/myFunc
```

This validation logic is inherited by Var from the ARef class.

There are a number of utility functions for setting the root binding of a Var without affecting the values seen by individual threads and for unbinding the root binding. There are also functions for dealing the metadata associated with the Var. This metadata is used for flagging whether the Var represents a function or a macro for example.

---

#### 4.1.4 Call my Agent will you? (2010-01-07 21:01)

Clojure contains some interesting concurrency mechanisms aside from the normal java locks mechanism. My favourite has got to be the bounded version of software transactional memory which we'll get to in later post. There's a good [1]retrospective, by Joe Duffy, on some early work on getting software transactional memory into the CLR, though the post details how they opted for an unbounded implemented (all objects are affected by the transactions) which contrasts with the Clojure version where there is a specific object type that interacts with the transactions (a bounded implementation).

In order to see some code, we'll consider a data structure which is a vector of two integers. Pretend they are bank balances if you will, and we need to maintain an invariant that the total of the two integers doesn't change when we do a transfer operation of one unit from the first element to the second. The Vector type in Clojure is functional, so we can't just change the first element of the vector when we start the transfer, but instead need to generate a new

vector. This gives us the problem of where we put the current vector representing the current state. Clojure offers us a type called an Atom for handling this.

An Atom guards a value.

```
user=> (def balances (atom [1000, 0]))  
 #'user/balances  
user=> balances  
#<Atom@1286b10: [1000 0]>  
user=> @balances  
[1000 0]
```

More importantly, it supports a swap! operation. This takes the current value, applies a function to it to get a potential new value, and then tries to atomically assign this new value into the Atom. If the value has changed between the read and the assignment, then the system will try again, going around the read/generate/assign cycle until the new value is successfully set. Because of this behaviour, you'd usually want the function to be pure (with no side-effects).

Let's define a couple of operations, action1 and action2 that we'll run concurrently. The functions that generate new values, update1 and update2, output a message when they are being called. I've added some sleep statements to simulate long running actions and to try to ensure that we need to re-execute one of the updates.

```
(defn update1[balances]  
(println "update1")  
(let [[x y] balances]  
(. Thread (sleep 1000))  
[ (- x 1) (+ y 1)]))
```

```
(defn update2[balances]  
(println "update2")  
(let [[x y] balances]  
[ (- x 1) (+ y 1)]))
```

```
(defn action1[]  
(swap! balances update1))
```

```
(defn action2[]  
(. Thread (sleep 30))  
(swap! balances update2))
```

We can use pcalls to run these two actions simultaneously.

```
user=> (pcalls action1 action2)  
update1  
update2  
update1  
([998 2] [999 1])
```

We see that the first update function starts before the second update function, but this second function is quicker and assigns its result to the Atom, forcing the first update to run on the new value. Note how the invariant is maintained on the vector.

Internally the Atom is implemented as an AtomicReference  
final AtomicReference state;

```
public Atom(Object state) {  
    this.state = new AtomicReference(state);  
}
```

The deref method is used to access the value of the Atom,  
user=> '@balances  
(clojure.core/deref balances)

```
public Object deref() {  
    return state.get();  
}
```

And there are several overloads of the swap method for using the update function.

```
public Object swap(IFn f) throws Exception {  
    for(; ;)  
    {  
        Object v = deref();  
        Object newv = f.invoke(v);  
        validate(newv);  
        if(state.compareAndSet(v, newv))  
        {  
            notifyWatches(v, newv);  
            return newv;  
        }  
    }  
}
```

Given that we have easy access to java methods and classes, we could opt to use the normal java monitors to avoid the possibility of the swap! re-running the update function. We could create an instance of Object and use that as a lock to guard our data structure.

```
(def lock (Object.))
```

```
(defn lockedAction1[]  
(locking lock  
(action1)))
```

```
(defn lockedAction2[]  
(locking lock
```

```
(action2))
```

The locking macro expands into code that uses monitors to manipulate the lock on the object.

```
user=> (macroexpand '(locking lock something))
(let* [lockee __2725 __auto __lock]
  (try (monitor-enter lockee __2725 __auto __)
    something
  (finally (monitor-exit lockee __2725 __auto __))))
```

When we run these versions of our functions, access to the Atom is serialized.

```
user=> (pcalls lockedAction1 lockedAction2)
update1
update2
([999 1] [998 2])
```

However, Clojure provides another means of guarding the data, which guarantees that only a single update function is running at any moment. This is called an Agent. When you send a message, which is an updater functions plus some arguments, to an Agent, the Agent makes sure that the updaters are serialized so you don't need to do any locking.

We first create an Agent responsible for our two balances.

```
user=> (def balances (agent [1000, 0]))
#'user/balances
```

We can use the deref function, most easily denoted using the @ reader macro, to get the data from the Agent.

```
user=> @balances
[1000 0]
```

And now we can send-off the update1 and update2 actions to the Agent. The Agent will queue them and run them one at a time on some other thread.

```
user=> (pcalls #(send-off balances update1) #(send-off balances update2))
( #<Agent@15291cd: update1
[1000 0]> #<Agent@15291cd: [1000 0]>)
user=> update2
```

Note that in the above there are messages coming from the background threads that get mixed with the messages from the REPL (read-eval-print-loop). The balances have now been updated appropriately.

```
balances
#<Agent@15291cd: [998 2]>
```

The Agent class is declared in Agent.java and its constructor is called when you use the agent function defined in core.clj.

```
(defn agent
```

```
([state] (new clojure.lang.Agent state))
([state & options]
  (setup-reference (agent state) options)))
```

An Agent has the following fields.

```
volatile Object state;
AtomicReference<IPersistentStack> q = new AtomicReference(PersistentQueue.EMPTY);
volatile ISeq errors = null;
final static ThreadLocal<IPersistentVector> nested = new ThreadLocal<IPersistentVector>();
```

state is the state that the Agent is guarding. It is set by the constructor, which also allows the use of a validator function to check the invariants on this state.

```
public Agent(Object state, IPersistentMap meta) throws Exception {
    super(meta);
    setState(state);
}
```

```
boolean setState(Object newState) throws Exception {
    validate(newState);
    boolean ret = state != newState;
    state = newState;
    return ret;
}
```

errors is a collection of the errors that happened during the execution of the update functions. This collection is used to prevent access to the data if there are uncleared errors.

```
public Object deref() throws Exception {
    if(errors != null)
    {
        throw new Exception("Agent has errors", (Exception) RT.first(errors));
    }
    return state;
}
```

```
public ISeq getErrors() {
    return errors;
}
```

```
public void clearErrors() {
    errors = null;
}
```

The latter two functions have a Clojure interface defined in core.clj.

```
(defn agent-errors
  "Returns a sequence of the exceptions thrown during asynchronous
  actions of the agent."
  [ #^clojure.lang.Agent a] (. a (getErrors)))
```

```
(defn clear-agent-errors
  "Clears any exceptions thrown during asynchronous actions of the
  agent, allowing subsequent actions to occur."
  [ #^clojure.lang.Agent a] (. a (clearErrors)))
```

As we saw above, update functions are sent to Agents. This can be done in two ways, using either send or send-off. The interface functions are defined in core.clj.

```
(defn send
  [ #^clojure.lang.Agent a f & args]
  (. a (dispatch f args false)))
```

```
(defn send-off
  [ #^clojure.lang.Agent a f & args]
  (. a (dispatch f args true)))
```

These functions add instances of the nested Action class to the q. The Action records the Agent to which a given update function and args were sent, together with a value named solo that reflects which thread pool to run the action on.

```
public Object dispatch(IFn fn, ISeq args, boolean solo) {
if(errors != null)
{
throw new RuntimeException("Agent has errors", (Exception) RT.first(errors));
}
Action action = new Action(this, fn, args, solo);
dispatchAction(action);
return this;
}
```

dispatchAction needs to behave differently if we are running inside a transaction (the first condition) or if a send happens inside the execution of an update function (the second case). In the normal case, the Action enqueues itself on the Agent to which it was sent.

```
static void dispatchAction(Action action) {
LockingTransaction trans = LockingTransaction.getRunning();
if(trans != null)
trans.enqueue(action);
else if(nested.get() != null)
{
nested.set(nested.get().cons(action));
}
else
action.agent.enqueue(action);
}
```

Enqueue is fairly straightforward.

```
void enqueue(Action action) {
boolean queued = false;
IPersistentStack prior = null;
while(!queued)
{
```

```

prior = q.get();
queued = q.compareAndSet(prior, (IPersistentStack) prior.cons(action));
}
if(prior.count() == 0)
action.execute();
}

```

Execute on an Action determines which thread pool to use for running the update function.

```

void execute() {
if(solo)
soloExecutor.execute(this);
else
pooledExecutor.execute(this);
}

```

In the solo case, when send-off is used in Clojure, we use an expandable threadpool, whereas in the send case we use a fixed size pool.

```

final public static ExecutorService pooledExecutor =
Executors.newFixedThreadPool(2 + Runtime.getRuntime().availableProcessors());
final public static ExecutorService soloExecutor = Executors.newCachedThreadPool();

```

A worker thread ends up being passed an Action and executes the doRun method. The action code binds a variable \*agent\* to the currently executing agent and then sets the nested field ready to receive any nested calls to send. The update function is then called, and if there are no errors the new state is assigned. If there were errors during the execute of the update function, the error is logged and we don't releasePendingSends that have accumulated in the nested field. In the success alternative, the nested collection is traversed and the nested sends are queued to the relevant agents. We then pop the action off the agent's queue, and if the agent has more work to do we execute the next Action.

```

static void doRun(Action action) {
try
{
Var.pushThreadBindings(RT.map(RT.AGENT, action.agent));
nested.set(PersistentVector.EMPTY);

boolean hadError = false;
try
{
Object oldval = action.agent.state;
Object newval = action.fn.applyTo(RT.cons(action.agent.state, action.args));
action.agent.setState(newval);
action.agent.notifyWatches(oldval,newval);
}
catch(Throwable e)
{
action.agent.errors = RT.cons(e, action.agent.errors);
hadError = true;
}
}

```

```

if(!hadError)
{
releasePendingSends();
}

boolean popped = false;
IPersistentStack next = null;
while(!popped)
{
IPersistentStack prior = action.agent.q.get();
next = prior.pop();
popped = action.agent.q.compareAndSet(prior, next);
}
if(next.count() > 0)
((Action) next.peek()).execute();
}
finally
{
nested.set(null);
Var.popThreadBindings();
}
}

```

Note how the use of the .count() to trigger the execution of Actions means that only a single Action will be running on a given Agent at once.

Another important thing we can do with an Agent is wait for it to finish processing its action queue. This is done with the await function together with a version, await-for, which times out and await1 which is optimised for a single Agent. This queues a send to each agent and then tracks their completion. The function uses \*agent\*, which is bound by the execution code when we are inside agent processing on the current thread, to stop await being called inside an update action. This code also uses the io! macro which throws an exception if we are inside a transaction.

```

(defn await
"Blocks the current thread (indefinitely!) until all actions
dispatched thus far, from this thread or agent, to the agent(s) have
occurred."
[ & agents]
(io! "await in transaction"
(when *agent*
(throw (new Exception "Can't await in agent action")))
(let [latch (new java.util.concurrent.CountDownLatch (count agents))
count-down (fn [agent] (. latch (countDown)) agent)]
(doseq [agent agents]
(send agent count-down))
(. latch (await))))
```

#### 4.1.5 What do you base that on? (2010-01-09 14:00)

It's quite interesting to look at the various interfaces that the Clojure implementation objects support, as it gives quite a clue as to what is happening inside the implementation.

The lowest level abstract class is called Obj. This supports the IObj interface which extends IMeta. These interfaces have the following definition and support the association of metadata information with an object. meta() accesses the metadata and withMeta takes some new metadata and makes a new instance with this metadata. This method is abstract in the Obj class.

```
public interface IMeta {  
    IPersistentMap meta();  
}  
public interface IObj extends IMeta {  
    public IObj withMeta(IPersistentMap meta);  
}
```

Meta data is used by the system for many things. The most important example is that information is associated with a Var to record information such as whether it defines a macro as well as location information for its definition. The example below shows the metadata associated with the Var bound to defn. Note that the reader macro #' can be used to get the Var associated with a symbol and ^ is shorthand for the meta function (which is defined in core.clj).

```
user=> (var defn)  
#'clojure.core/defn  
user=> #'defn  
#'clojure.core/defn  
user=> (meta #'defn)  
{:macro true, :ns #<Namespace clojure.core>, :name defn, :file "clojure/core.clj", :line 189,  
:doc "Same as (def name (fn [params*] exprs*)) or (defn name (fn ([params*] exprs*+)) with  
any doc-string or attrs addedn to the var metadata", :arglists ([name doc-string? attr-map?  
[params*] body] [name doc-string? attr-map? ([params*] body) + attr-map?]) }  
user=> ^ #'defn  
{:macro true, :ns #<Namespace clojure.core>, :name defn, :file "clojure/core.clj", :line 189,  
:doc "Same as (def name (fn [params*] exprs*)) or (defn name (fn ([params*] exprs*+)) with  
any doc-string or attrs addedn to the var metadata", :arglists ([name doc-string? attr-map?  
[params*] body] [name doc-string? attr-map? ([params*] body) + attr-map?]) }
```

Because the metadata is a map, and applying a map to a value looks up the value as a key in the map, this means we can easily check the Var for being a macro or get the source location information using:

```
user=> (^ #'defn :macro)  
true
```

```
user=> (^ #'defn :file)
"clojure/core.clj"
user=> (^ #'defn :line)
189
```

The Clojure function with-meta can be used to call the withMeta method. This is defined in core.clj.

```
(def
#^ {:arglists '([ #^clojure.lang.IObj obj m])
:doc "Returns an object of the same type and value as obj, with
map m as its metadata." }
with-meta (fn with-meta [ #^clojure.lang.IObj x m]
(. x (withMeta m))))
```

```
user=> (def testVec [1 2 3])
#'user/testVec
user=> (meta testVec)
nil
user=> (def testVec2 (with-meta testVec {:a 1 :b 2 }))
#'user/testVec2
user=> (meta testVec2)
{:a 1, :b 2 }
user=> testVec2
[1 2 3]
```

The next interface to look at is IFn which represents something that can be called as function.

```
public interface IFn extends Callable, Runnable {
public Object invoke() throws Exception;
public Object invoke(Object arg1) throws Exception;
public Object invoke(Object arg1, Object arg2) throws Exception;
public Object invoke(Object arg1, Object arg2, Object arg3) throws Exception;
public Object invoke(Object arg1, Object arg2, Object arg3, Object arg4) throws Exception;
public Object invoke(Object arg1, Object arg2, Object arg3, Object arg4, Object arg5) throws
Exception;
.....
public Object invoke(Object arg1, Object arg2, Object arg3, Object arg4, Object arg5, Object
arg6, Object arg7,
Object arg8, Object arg9, Object arg10, Object arg11, Object arg12, Object arg13, Object
arg14,
Object arg15, Object arg16, Object arg17, Object arg18, Object arg19, Object arg20,
Object... args)
throws Exception;
public Object applyTo(ISeq arglist) throws Exception;
}
```

The idea is that performance is optimised by having specialised function versions that reflect the number of arguments that the function expects. So for a simple function that expects two arguments, we'd implement the overload with two arguments and throw an arity exception in the other overloads. The behaviour of throwing an arity exception for all overloads is implemented in the AFn abstract class. This means it is easy to implement a new function by

inheriting from this class and overriding the relevant overloads corresponding to the arity of the function.

Some functions can take different numbers of arguments. For these functions several of the overloads would be implemented.

```
user=> (+ 1 2 3 4)
```

```
10
```

Clojure supports a means of applying a function to an arbitrary number of arguments, via the function apply . This is implemented by using the applyTo method of the IFn interface.

```
user=> (defn foo[x y] (+ x y))
```

```
#'user/foo
```

```
user=> (apply foo [1 4])
```

```
5
```

AFn implements the applyTo by counting the number of arguments and calling back into the appropriate overload of invoke. AFn also implements run() and call() to support the Runnable and Callable interfaces, where these simply call the invoke() overload.

Clojure functions, defined via the fn special form, are instances of classes that extend AFunction, AFunction is a class which extends AFn, implements the marker (empty) interface Fn, and defines a compare method to allow it to implement the Comparator interface. This compare simply uses the 2 argument invoke overload and then checks the resulting value.

```
public int compare(Object o1, Object o2) {  
    try {  
        Object o = invoke(o1, o2);  
  
        if(o instanceof Boolean)  
        {  
            if(RT.booleanCast(o))  
                return -1;  
            return RT.booleanCast(invoke(o2,o1))? 1 : 0;  
        }  
    }  
}
```

```
Number n = (Number) o;  
return n.intValue();  
}  
catch(Exception e)  
{  
    throw new RuntimeException(e);  
}  
}
```

Putting this all together, we find that the following Clojure function, defined in the namespace clojure.test,

```
(defn foo[x y] (+ x y))
```

translates into

```
public class test $foo __1 extends AFunction
{
    public static final Var const __0 = (Var)RT.var("clojure.core", "+");

    public Object invoke(Object x, Object y) throws Exception { x = null; y = null; return Numbers.add(x, y);
}
}
```

(the null assignments are actually carried out after x and y have been pushed on to the stack so they do get passed through to the +. The decompiler is just wrong here!)

And the metadata for the function is set up by the load method in the test \_\_init file for

```
public class test __init
{
    public static final Var const __0 = (Var)RT.var("clojure.core", "ns");
    public static final Var const __1 = (Var)RT.var("clojure.core", "in-ns");
    public static final AFn const __2 = (AFn)Symbol.create(null, "clojure.test");
    public static final Var const __3 = (Var)RT.var("clojure.core", "refer");
    public static final AFn const __4 = (AFn)Symbol.create(null, "clojure.core");
    public static final Var const __5 = (Var)RT.var("clojure.core", "defn");
    public static final Var const __6 = (Var)RT.var("clojure.test", "foo");
    public static final Object const __7 = Keyword.intern(Symbol.create(null, "file"));
    public static final Object const __8 = Keyword.intern(Symbol.create(null, "line"));
    public static final Object const __9 = Integer.valueOf(3);
    public static final Object const __10 = Keyword.intern(Symbol.create(null, "arglists"));
    public static final Object const __11;
    public static final Var const __12;

    public static void load()
    {
        ((IFn)const __1.get()).invoke(const __2);
        Var tmp38 __35 = const __6;
        tmp38 __35.bindRoot(new test.foo __1());
        tmp38 __35.setMeta((IPersistentMap)RT.map(new Object[] { const __7, "clojure/test.clj", const __8, const __9, const __10, const __11 }));
    }

    // Extra stuff removed
}

user=> (instance? clojure.lang.IFn foo)
true
user=> (instance? clojure.lang.Fn foo)
true
```

```
user=> (instance? clojure.lang.IObj foo)
true
user=> (instance? clojure.lang.Obj foo)
true
```

It's interesting that things like Var also implement IFn so they can be applied to things. In particular, Var implements the no argument invoke as a lookup on its own value and an invoke on that.

```
public Object invoke() throws Exception {
    return fn().invoke();
}
final public IFn fn() {
    return (IFn) deref();
}
```

```
user=> (defn bar[] 10)
#'user/bar
user=> ( #'bar)
10
user=> (var bar)
#'user/bar
user=> ((var bar))
10
```

---

#### 4.1.6 Let's compose some arguments (2010-01-10 09:27)

Clojure has some good support for functional programming. We've already looked at anonymous functions.

```
user=> (fn [x y] (* x y))
#<user $eval __181 $fn __183 user $eval __181 $fn __183@e61fd1>
user=> (*1 2 3)
6
```

Here we used \*1 which is bound to the last result in the read-eval-print loop, \*2 is bound to the result before that and \*3 is bound to the one before that.

Functions can be written to take a variable number of arguments by binding an argument to a sequence of all arguments after a given point.

```
user=> (fn [x y & z] z)
#<user $eval __232 $fn __234 user $eval __232 $fn __234@cf66b>
user=> (*1 1)
java.lang.IllegalArgumentException: Wrong number of args passed to: user $eval-232 $fn
```

```
(NO_SOURCE_FILE:0)
user=> (*1 1 2)
nil
user=> (*2 1 2 3)
(3)
user=> (*3 1 2 3 4 5)
(3 4 5)
```

There is good support for dealing with higher order functions. The first, partial, partially applies a function to some arguments producing another function that can be applied to the additional arguments.

```
user=> (defn addAll[x y z] (+ x y z))
#'user/addAll
user=> (partial addAll 1)
#<core $partial __2880 $fn __2882 clojure.core $partial __2880 $fn __2882@1884174>
user=> (*1 2 3)
6
user=> (partial addAll 2 3)
#<core $partial __2880 $fn __2886 clojure.core $partial __2880 $fn __2886@baa466>
user=> (*1 4)
9
```

Partial is defined in core.clj as

```
(defn partial
([f arg1]
(fn [& args] (apply f arg1 args)))
([f arg1 arg2]
(fn [& args] (apply f arg1 arg2 args)))
([f arg1 arg2 arg3]
(fn [& args] (apply f arg1 arg2 arg3 args)))
([f arg1 arg2 arg3 & more]
(fn [& args] (apply f arg1 arg2 arg3 (concat more args))))))
```

comp composes the functions, applying the rightmost first and then applying the next function to the result.

```
user=> (comp (fn [x] (- x 1)) (partial * 2))
#<core $comp __2874 $fn __2876 clojure.core $comp __2874 $fn __2876@134b07e>
user=> (*1 5)
9
```

comp is also defined in core.clj

```
(defn comp
[& fs]
(let [fs (reverse fs)]
(fn [& args]
(loop [ret (apply (first fs) args) fs (next fs)]
(if fs
(recur ((first fs) ret) (next fs))
ret)))))
```

One other possibly useful function is eval which takes a Clojure form and evaluates it. It does this by compiling it and running the result. Eval is useful in rare occasions, but often it is better not to use it.

```
user=> (eval '(defn bar[x] x))
#'user/bar
user=> (eval '(bar 20))
20
```

This is implemented by calling into the code for the Compiler.

```
(defn eval
[form] (. clojure.lang.Compiler (eval form)))
```

---

#### 4.1.7 What's next in the sequence? (2010-01-10 21:09)

Clojure has a really nice abstraction that works with many of the supplied datatypes. The instances of the datatypes can be accessed as sequences. We'll see later that there is a good range of functions for working with such sequences, in order to search, filter and combine them in interesting ways.

First we ought to have a quick look at the underlying java datatypes. The first, Seqable, is an interface supported by types that can provide a sequence representation of the data that they contain.

```
public interface Seqable {
  ISeq seq();
}
```

Most of Clojure's datatypes are persistent collections. These have a count of the number of elements that they contain, a means for getting an empty collection of the type and a way to add another element to the collection, though this is functional so returns the new collection without modifying the original. There's also a function for checking if the collections are equivalent.

```
public interface IPersistentCollection extends Seqable {
  int count();
  IPersistentCollection cons(Object o);
  IPersistentCollection empty();
  boolean equiv(Object o);
}
```

equiv is the method used by Clojure's = function, defined in core.clj as

```
(defn =
{:tag Boolean
:inline (fn [x y] '(. clojure.lang.Util equiv x y))}
```

```
:inline-arithies # {2 } }
([x] true)
([x y] (clojure.lang.Util/equiv x y))
([x y & more]
(if (= x y)
(if (next more)
(recur y (first more) (next more))
(= y (first more)))
false)))
```

with the Util.equiv method defined as

```
static public boolean equiv(Object k1, Object k2) {
if(k1 == k2)
return true;
if(k1 != null)
{
if(k1 instanceof Number)
return Numbers.equiv(k1, k2);
else if(k1 instanceof IPersistentCollection & & k2 instanceof IPersistentCollection)
return ((IPersistentCollection)k1).equiv(k2);
return k1.equals(k2);
}
return false;
}
```

There's a marker interface defined, accessed by the Clojure sequential? function.

```
public interface Sequential {
```

```
}
```

Lastly, there's an interface that allows access to the elements of the sequence without any kind of positional indexing.

```
public interface ISeq extends IPersistentCollection, Sequential {
Object first();
ISeq next();
ISeq more();
ISeq cons(Object o);
}
```

first returns the first element of the sequence. next and more return a sequence containing all but the first element, in the same order. Cons adds an element to the start of the sequence. The type of the returned sequence is not guaranteed to be the same as the sequence to which the function is applied. more is surfaced into Clojure as the rest function.

```
(def
#^ {:arglists '([coll])
:tag clojure.lang.ISeq
:doc "Returns a possibly empty seq of the items after the first. Calls seq on its argument." }
rest (fn rest [x] (. clojure.lang.RT (more x))))
```

The difference between rest and next is that the latter returns nil for an empty sequence

whereas the former returns an empty sequence.

```
user=> (next [])
nil
user=> (rest [])
()
```

We can use doseq to run across the elements of a number of Clojure's main datatypes.

```
user=> (class '(1 2 3 4))
clojure.lang.PersistentList
user=> (doseq [x '(1 2 3 4)] (println x))
1
2
3
4
nil
user=> (class [1 2 3 4])
clojure.lang.LazilyPersistentVector
user=> (doseq [x [1 2 3 4]] (println x))
1
2
3
4
nil
user=> (class # {1 2 3 4 })
clojure.lang.PersistentHashSet
user=> (doseq [x # {1 2 3 4 }] (println x))
1
2
3
4
nil
user=> (class {1 2 3 4 })
clojure.lang.PersistentArrayMap
user=> (doseq [x {1 2 3 4}] (println x))
[1 2]
[3 4]
nil
```

There is a for macro supporting the same syntax as doseq which collects the resulting values into a sequence instead of executing them for side-effects.

```
user=> (for [x '(1 2 3) :when (>= x 2) :let [y (* x 2)]] y)
(4 6)
user=> (for [x '(1 2 3) :while (>= x 2) :let [y (* x 2)]] y)
()
user=> (for [x '(1 2 3) :while (<= x 2) :let [y (* x 2)]] y)
(2 4)
```

Many of the sequence manipulation functions return an element of type LazySeq. The Java type takes an IFn and stores it until the sequence is accessed, at which point the function is applied to no arguments and the result is stored and is accessed when the LazySeq is accessed. If an exception happens, then, as you would expect, it is not caught.

```
user=> (def a (lazy-seq (println "called") '(1 2 3)))
```

```
#'user/a
user=> (first a)
called
1
user=> (first a)
1
```

This type supports both ISeq and List

```
public final class LazySeq extends Obj implements ISeq, List {
with the majority of operations calling the seq() method to realise the sequence, by calling the
captured function, before going on to do any work on the elements.
```

This function is available in Clojure via the seq function

```
(def
#^ {:arglists '([coll])
:doc "Returns a seq on the collection. If the collection is
empty, returns nil. (seq nil) returns nil. seq also works on
Strings, native Java arrays (of reference types) and any objects
that implement Iterable."
:tag clojure.lang.ISeq }
seq (fn seq [coll] (. clojure.lang.RT (seq coll))))
```

which uses a method defined in the RT.java class

```
static public ISeq seq(Object coll) {
if(coll instanceof ASel)
return (ASel) coll;
else if(coll instanceof LazySeq)
return ((LazySeq) coll).seq();
else
return seqFrom(coll);
}
```

```
user=> (def a (lazy-seq (print "called") '(1 2 3)))
#'user/a
user=> (def b (seq a))
called #'user/b
```

Note the use of the def in the above to avoid the evaluation of the sequence by the printer that the read-eval-print loop uses.

```
user=> (lazy-seq (print "called") '(1 2 3))
(called1 2 3)
```

There are many sequence manipulation functions, amongst them:

```
user=> (filter #(> % 2) '(1 2 3))
(3)
user=> (remove #(> % 2) '(1 2 3))
(1 2)
user=> (map #(* % 2) '(1 2 3))
(2 4 6)
user=> (map (fn [x y] (* x y)) '(1 2 3) '( 4 5 6))
(4 10 18)
user=> (mapcat #(list % (+ % 1)) '( 1 2 3))
(1 2 2 3 3 4)
```

```

user=> (concat '(1 2 3) [4 5 6])
(1 2 3 4 5 6)
user=> (concat '(1 2 3) [4 5 6] # {7 8 9 })
(1 2 3 4 5 6 7 8 9)
user=> (take 3 '(a b c d e f))
(a b c)
user=> (take-while #(< % 3) '(1 2 3 4 5 6))
(1 2)
user=> (drop 2 '(a b c d e f))
(c d e f)
user=> (drop-last 2 '(a b c d e f))
(a b c d)
user=> (drop-while #(< % 3) '(1 2 3 4 5 6))
(3 4 5 6)
user=> (take 10 (cycle '(1 2 3)))
(1 2 3 1 2 3 1 2 3 1)
user=> (split-at 3 '(a b c d e f g))
[(a b c) (d e f g)]
user=> (split-with #(< % 3) '(1 2 3 4 5 6))
[(1 2) (3 4 5 6)]
user=> (take 10 (repeat 1))
(1 1 1 1 1 1 1 1 1 1)
user=> (repeat 1 10)
(10)
(user=> (take 4 (iterate #(* % 2) 1)))
(1 2 4 8)
user=> (range 1 10)
(1 2 3 4 5 6 7 8 9)
user=> (partition 2 '(1 2 3 4 5 6 7 8 9 10))
((1 2) (3 4) (5 6) (7 8) (9 10))
user=> (partition 2 3 '(1 2 3 4 5 6 7 8 9 10))
((1 2) (4 5) (7 8))

```

In order to force the evaluation of the members of a lazy sequence, there are two functions doall and dorun. dorun calls next until the end of the sequence is reached, hence causing the evaluation of all of the elements. doall calls dorun and then returns the original sequence.

```

user=> (def a
(concat (lazy-seq (do (print "a") (list 1)))
(lazy-seq (do (print "b") (list 2))))))
#'user/a
user=> (dorun a)
abnil

```

#### 4.1.8 What's the name of my space? (2010-01-13 21:04)

The next thing we need to look at are symbols, keywords and namespaces. Symbols and keywords are used to name things, and the Clojure versions are a bit different from how they work in Common Lisp. In Common Lisp, there is a notion of a package, which is a lot like a Clojure namespace, and there are symbols which have a name and are maybe affiliated with a package. The Clojure version of a symbol has a name and may have a namespace, but the namespace may not exist even if it is present in the symbol.

We can make symbols using the symbol function

```
user=> (symbol "foo" "bar")
```

```
foo/bar
```

```
user=> (symbol "bar")
```

```
bar
```

And use the namespace and name functions to deconstruct them

```
user=> (namespace (symbol "foo" "bar"))
```

```
"foo"
```

```
user=> (name (symbol "foo" "bar"))
```

```
"bar"
```

```
user=> (namespace (symbol "bar"))
```

```
nil
```

```
user=> (name (symbol "bar"))
```

```
"bar"
```

Rather than using the function symbol to make the symbol, the reader automatically makes the symbol when it reads it.

```
user=> (name 'bar)
```

```
"bar"
```

```
user=> (namespace 'bar)
```

```
nil
```

```
user=> (name 'foo/bar)
```

```
"bar"
```

```
user=> (namespace 'foo/bar)
```

```
"foo"
```

In Common Lisp, Keywords are symbol with a particular home package - the package that owns the symbol is the package named Keyword. In Clojure, a Keyword is an instance of a class which contains a symbol.

```
user=> (keyword "foo" "bar")
```

```
:foo/bar
```

```
user=> (namespace :foo/bar)
```

```
"foo"
```

```
user=> (name :foo/bat)
```

```
"bat"
```

```
user=> (keyword "bar")
```

```
:bar
```

```
user=> (namespace :bar)
```

```
nil
```

```
user=> (name :bar)
```

```
"bar"
```

Keywords are useful because they evaluate to themselves, whereas symbols are evaluated by

looking them up and evaluating the result of this lookup.

```
user=> :boo
:boo
user=> boo
java.lang.Exception: Unable to resolve symbol: boo in this context (NO_SOURCE_FILE:0)
```

Just before we look at any more code, remember than the String class in java has an intern method that converts a string to a canonical instance of the same string.

```
user=> (. (new String "a") (intern))
"a"
user=> (new String "a")
"a"
user=> (identical? *1 *2)
false
user=> (new String "a")
"a"
user=> (new String "a")
"a"
user=> (identical? (new String "a") "a")
false
user=> (. (new String "a") (intern))
"a"
user=> (. (new String "a") (intern))
"a"
user=> (identical? *1 *2)
true
```

The Symbol class always interns the strings used for the namespace and the name to make later comparison identity comparison instead of equality comparison.

```
static public Symbol intern(String ns, String name) {
    return new Symbol(ns == null ? null : ns.intern(), name.intern());
}

static public Symbol intern(String nsname) {
    int i = nsname.lastIndexOf('/');
    if(i == -1)
        return new Symbol(null, nsname.intern());
    else
        return new Symbol(nsname.substring(0, i).intern(), nsname.substring(i + 1).intern());
}
```

Symbols have metadata.

```
user=> (def x 'a)
 #'user/x
user=> x
a
user=> (meta x)
nil
user=> (with-meta 'a {:a 2 })
a
user=> (def y *1)
 #'user/y
```

```
user=> (meta y)
{:a 2 }
```

And compare equal if their name and namespace components are the same.

```
user=> (= 'a 'a)
true
user=> (identical? 'a 'a)
false
```

In this example, the reader is making a new symbol every time it reads one of the “a” characters.

When the Clojure reader reads something from a file, perhaps prior to compiling it, the forms are read as lists of symbols.

```
user=> (def form '(defn foo [x y] (+ x y)))
 #'user/form
user=> (first form)
defn
user=> (namespace (first form))
nil
user=> (name (first form))
"defn"
user=> (namespace (second form))
nil
user=> (name (second form))
"foo"
```

Namespaces are the next thing we need to understand in order to see how the system evaluates the form contained in the Var named form. The first symbol, “defn” will be resolved in the current namespace to see if it denotes a Var.

```
user=> *ns*
#<Namespace user>
user=> (first form)
defn
user=> (resolve (first form))
#'clojure.core/defn
user=> (ns-resolve *ns* (first form))
#'clojure.core/defn
```

The Java class for Namespace contains the following fields.

```
public class Namespace extends AReference {
final public Symbol name;
final AtomicReference<IPersistentMap> mappings = new AtomicReference<IPersistentMap>();
final AtomicReference<IPersistentMap> aliases = new AtomicReference<IPersistentMap>();
final static ConcurrentHashMap<Symbol, Namespace> namespaces = new ConcurrentHashMap<Symbol, Namespace>();
```

The namespace is named by a Symbol and there is a global map of symbols to namespaces. Each namespace has a group of mappings and some aliases.

When a namespace is constructed in the java code, the constructor deals with metadata, sets the name of the namespace, sets the aliases to an empty map and initializes the mappings to a default set.

```

Namespace(Symbol name) {
super(name.meta());
this.name = name;
mappings.set(RT.DEFAULT_IMPORTS);
aliases.set(RT.map());
}

```

The mappings of a namespace, accessed by the Clojure function ns-map, maps from symbols to Var or Class instances.

```

user=> (ns-map (find-ns 'user))
{sorted-map #'clojure.core/sorted-map, read-line #'clojure.core/read-line, re-pattern
 #'clojure.core/re-pattern, keyword? #'clojure.core/keyword?, val #'clojure
.core/val, ProcessBuilder java.lang.ProcessBuilder, Enum java.lang.Enum, SuppressWarnings
java.lang.SuppressWarnings, *compile-path* #'clojure.core/*compile-path*, par #'user/par,
max-key #'clojure.core/max-key, list* #'clojure.core/list*, ns-aliases #'clojure.core/ns-aliases,
the-ns #'clojure.core/the-ns, == #'clojure.core/==, longs #'clojure.core/longs, special-
form-anchor #'clojure.core/special-form-anchor, Throwable java.lang.Throwable, InterruptedException
java.lang.InterruptedException, instance? #'clojure.core/instance?, syntax-
symbol-anchor #'clojure.core/syntax-symbol-anchor, Thread $UncaughtExceptionHandler
java.lang.Thread $UncaughtExceptionHandler, RuntimeException java.lang.RuntimeException,
..... }

```

RT.DEFAULT\_IMPORTS is a mappings consisting of a large number of predefined Java classes.

```

final static IPersistentMap DEFAULT_IMPORTS = map(
Symbol.create("Boolean"), Boolean.class,
Symbol.create("Byte"), Byte.class,
Symbol.create("Character"), Character.class,
Symbol.create("Class"), Class.class,
Symbol.create("ClassLoader"), ClassLoader.class,
Symbol.create("Compiler"), Compiler.class,
Symbol.create("Double"), Double.class,
Symbol.create("Enum"), Enum.class,
Symbol.create("Float"), Float.class,
..... }

```

Hence, all newly created namespaces start with these mappings.

```

user=> (find-ns 'myNamespace)
nil
user=> (create-ns 'myNamespace)
#<Namespace myNamespace>
user=> (find-ns 'myNamespace)
#<Namespace myNamespace>
user=> (ns-resolve (find-ns 'myNamespace) 'Boolean)
java.lang.Boolean
user=> (ns-resolve (find-ns 'myNamespace) 'boolean)
nil

```

The Namespace class supports methods all, remove, find, findOrCreate for accessing a sequence of all known namespaces and for adding and removing from this set.

```

user=> (all-ns)
( #<Namespace clojure.set> #<Namespace user> #<Namespace clojure.main> #<Namespace clojure.core> #<Namespace clojure.zip> #<Namespace clojure.xml>
user=> (create-ns 'foo)
#<Namespace foo>
user=> (all-ns)
( #<Namespace clojure.set> #<Namespace user> #<Namespace clojure.main> #<Namespace clojure.core> #<Namespace clojure.zip> #<Namespace foo> #<Namespace clojure.xml>
user=> (remove-ns 'foo)
#<Namespace foo>
user=> (all-ns)
( #<Namespace clojure.set> #<Namespace user> #<Namespace clojure.main> #<Namespace clojure.core> #<Namespace clojure.zip> #<Namespace clojure.xml>
user=> (find-ns 'clojure.core)
#<Namespace clojure.core>
user=> (find-ns 'foo)
nil

```

Symbols can be interned into a namespace in order to make a named Var binding.

```

user=> (create-ns 'foo)
#<Namespace foo>
user=> (intern (find-ns 'foo) 'bar)
#'foo/bar

```

This uses one of the intern methods on Var

```

public static Var intern(Symbol nsName, Symbol sym) {
    Namespace ns = Namespace.findOrCreate(nsName);
    return intern(ns, sym);
}

```

which calls back into the intern method on the Namespace

```

public static Var intern(Namespace ns, Symbol sym) {
    return ns.intern(sym);
}

```

which does the actual work

```

public Var intern(Symbol sym) {
    if(sym.ns != null)
    {
        throw new IllegalArgumentException("Can't intern namespace-qualified symbol");
    }
}

```

```

IPersistentMap map = getMappings();
Object o;
Var v = null;
while((o = map.valAt(sym)) == null)
{
    if(v == null)
        v = new Var(this, sym);
    IPersistentMap newMap = map.assoc(sym, v);
    mappings.compareAndSet(map, newMap);
    map = getMappings();
}

```

```

}

if(o instanceof Var & & ((Var) o).ns == this)
return (Var) o;
throw new IllegalStateException(sym + " already refers to: " + o + " in namespace: " + name);
}

```

Lookup of a symbol in a namespace is via the resolve function (which will use the current namespace, that in the \*ns\* Var)

user=> (resolve 'foo/bar)

#'foo/bar

The functions implementing this are in core.clj

(defn resolve

"same as (ns-resolve \*ns\* symbol)"

[sym] (ns-resolve \*ns\* sym))

(defn ns-resolve

"Returns the var or Class to which a symbol will be resolved in the namespace, else nil. Note that if the symbol is fully qualified, the var/Class to which it resolves need not be present in the namespace."

[ns sym]

(clojure.lang.Compiler/maybeResolveIn (the-ns ns) sym))

Symbols from one namespace can be made visible in another.

user=> (ns-resolve 'foo 'mySym)

nil

user=> (create-ns 'other)

#<Namespace other>

user=> (intern 'other 'mySym)

#'other/mySym

user=> (binding [\*ns\* (find-ns 'foo)] (refer 'other))

nil

user=> (ns-resolve 'foo 'mySym)

#'other/mySym

user=> (ns-resolve 'foo 'other/mySym)

#'other/mySym

Internally this calls the refer method of the Namespace object

public Var refer(Symbol sym, Var var) {

return (Var) reference(sym, var);

}

which calls the reference method

Object reference(Symbol sym, Object val) {

if(sym.ns != null)

{

throw new IllegalArgumentException("Can't intern namespace-qualified symbol");

}

IPersistentMap map = getMappings();

Object o;

while((o = map.valAt(sym)) == null)

{

IPersistentMap newMap = map.assoc(sym, val);

```
mappings.compareAndSet(map, newMap);
map = getMappings();
}
if(o == val)
return o;
throw new IllegalStateException(sym + " already refers to: " + o + " in namespace: " + name);
}
```

Import can be used to map java classes into a namespace. ns-unmap can be used to remove any existing mapping.

```
user=> Thread
java.lang.Thread
user=> (ns-unmap *ns* 'Thread)
nil
user=> Thread
java.lang.Exception: Unable to resolve symbol: Thread in this context (NO _SOURCE
(FILE:0)
user=> (import '(java.lang Thread))
nil
user=> Thread
java.lang.Thread
```

One can also set up aliases for a given namespace to allow it to be accessed via another symbol when that symbol is used as the namespace part of the lookup symbol. There are also function ns-publics and ns-imports for getting subsets of the main namespace map.

---

#### 4.1.9 Get Real! (2010-01-13 21:06)

I posted a [1]review of the book Real-World Functional Programming to my work blog. An excellent book which is well worth a read.

---

1. <http://www.simple-talk.com/community/blogs/clivet/archive/2010/01/03/85227.aspx>

---

#### 4.1.10 What sort of structure is that? (2010-01-16 09:34)

Common Lisp has a type called a structure. This is often implemented as some kind of vector with the slots of the vector being accessed via named readers and writers. Structure types

support single inheritance with a derived structure type being implemented as a vector with more slots than the type from which it derives and with these extra slots being positioned after the slots for the subtype. The readers and writers defined on the subtype also work on the derived type. Typically a structure accessor is compiled to a highly efficient access into the values vector of the structure.

The Clojure structure is fairly similar. The structure is defined by a Def, which is a Java type that contains a sequence of keys and a map from each key to the index of that key in a value vector.

```
final ISeq keys;
final IPersistentMap keyslots;
```

The StructMap itself, in the Java class PersistentStructMap, contains the following fields.

```
final Def def;
final Object[] vals;
final IPersistentMap ext;
```

The Def is the class shown above. The vals is the values vector for the structure instance. The ext field is used to contain to hold value for slots associated with the instance which aren't in the struct map - if we define a struct map to have slots a,b and c, we can still maintain a value for a slot named d. However, this extra slot is not stored as efficiently as the others.

We can define a struct map and make an instance of it. If we don't supply all of the slot values, then they default to nil.

```
user=> (defstruct myStructType :a :b :c)
#'user/myStructType
user=> (def instance1 (struct myStructType 1 2 3))
#'user/instance1
user=> instance1
{:a 1, :b 2, :c 3 }
user=> (def instance2 (struct myStructType 1 2))
#'user/instance2
user=> instance2
{:a 1, :b 2, :c nil }
```

We can set the slots by supplying individual values.

```
user=> (struct-map myStructType :b 20)
{:a nil, :b 20, :c nil }
```

We can access values in the struct map.

```
user=> instance2
{:a 1, :b 2, :c nil }
user=> (instance1 :a)
1
user=> (:a instance1)
1
user=> (instance1 :c)
3
user=> (:c instance1)
3
```

And we can have it maintain the values for slots that aren't part of the original struct map.

```
user=> (def instance3 (struct-map myStructType :d 39))
```

```
#'user/instance3
user=> instance3
{:a nil, :b nil, :c nil, :d 39 }
```

In this last case, we see that the a,b and c slots are held in a vector, but the d slot is held in the ext mapping field.

```
main[1] print value.vals
value.vals = instance of java.lang.Object[3] (id=1473)
main[1] print value.ext
value.ext = " {:d 39 }"
main[1] print value.def.keys
value.def.keys = "(:a :b :c)"
main[1] print value.def.keysslots
value.def.keysslots = " {:a 0, :c 2, :b 1 }"
```

We can use the accessor function to create a function that knows which index in the vector the slot value resides. This returns a fn that should access the value efficiently.

```
user=> (accessor myStructType :b)
#< clojure.lang.PersistentStructMap $1@67064>
user=> (*1 instance1)
2
```

Though this accessor still checks the type of the structure.

```
user=> (accessor myStructType :b)
#< clojure.lang.PersistentStructMap $1@e0420b>
user=> (def a *1)
#'user/a
user=> (defstruct other :a :b :c)
#'user/other
user=> (a (struct other))
java.lang.Exception: Accessor/struct mismatch (NO _SOURCE _FILE:0)
```

And, of course, all of the usual operations work on the struct-map

```
user=> (assoc instance3 :e 30)
{:a nil, :b nil, :c nil, :d 39, :e 30 }
user=> (assoc *1 :a "bar")
{:a "bar", :b nil, :c nil, :d 39, :e 30 }
```

---

#### 4.1.11 Crash bang! (2010-01-16 09:34)

There's a great talk from the JVM Language Summit that has just been published on InfoQ. Entitled [1]"A Crash Course in Modern Hardware", it covers the architecture of modern processors, emphasising that today performance is not governed by the total number of instructions a program executes, but rather by the number of instructions between the cache misses when accessing data.

1. <http://www.infoq.com/presentations/click-crash-course-modern-hardware>

---

#### 4.1.12 I thought I'd just done that (2010-01-17 21:39)

Clojure is great in that its really easy to access the underlying java platform. Standard functions are compiled into classes supporting Runnable

```
user=> (instance? Runnable (fn [] 10))
true
```

and hence one can easily start a new Thread using such a function as method to run on the new thread.

```
user=> (.start (new Thread (fn [] (pr "Running"))))
nil
user=> 1
"Running"1
```

This post will attempt to dig into the implementation of the Software Transaction Memory (hereafter referred to as STM) in Clojure. Clojure provides a storage location called a Ref that will interact with any running transaction, allowing the same Ref to be seen as containing different values when running from different threads, and with the system merging a set of consistent changes when a transaction commits. Indeed, the system will even retry the transaction if a value is read but is changed by some other thread before the current transaction successfully commits.

We'll be working using the usual bank account example. We'll have two bank accounts, with a transaction transferring money between them. We'll have two separate threads trying to do this action, and will use a sleep to ensure that one of the transactions needs to restart.

```
user=> (def account1 (ref 1000))
#'user/account1
user=> account1
#<Ref@157b39f: 1000>
user=> @account1
1000
user=> (def account2 (ref 0))
#'user/account2
user=> account2
#<Ref@169df00: 0>
user=> @account2
0
```

We'll use the following function to transfer an amount of money. I've added some delays so that we can later try to get some contention between the transactions.

```
(defn transfer [amount delay1 delay2]
  (sync
```

```
nil
(. Thread (sleep delay1))
(let [acc1 @account1 acc2 @account2]
(ref-set account1 (- acc1 amount))
(. Thread (sleep delay2))
(ref-set account2 (+ acc2 amount))))
```

Now we can try running a couple of transactions in parallel, with some delays inserted to make things interesting.

```
(do (.start (new Thread [] (transfer 1 0 2000))))
(.start (new Thread [] (transfer 1 1000 0))))
```

We get the following breakpoints hit in the debugger. The breakpoints are set at the location where the closure representing the body of the transaction is called and the line after the value is returned by this closure.

```
Breakpoint hit: "thread=Thread-41", clojure.lang.LockingTransaction.run(), line=
```

```
236 bci=77
```

```
236 ret = fn.call();
```

```
Thread-41[1] cont
```

```
>
```

```
Breakpoint hit: "thread=Thread-42", clojure.lang.LockingTransaction.run(), line=
```

```
236 bci=77
```

```
236 ret = fn.call();
```

```
Thread-42[1] cont
```

```
>
```

```
Breakpoint hit: "thread=Thread-42", clojure.lang.LockingTransaction.run(), line=
```

```
236 bci=77
```

```
236 ret = fn.call();
```

```
Thread-42[1] cont
```

```
>
```

```
Breakpoint hit: "thread=Thread-41", clojure.lang.LockingTransaction.run(), line=
```

```
238 bci=84
```

```
238 if(info.status.compareAndSet(RUNNING, COMMITTING))
```

```
Thread-41[1] cont
```

```
>
```

```
Breakpoint hit: "thread=Thread-42", clojure.lang.LockingTransaction.run(), line=
```

```
236 bci=77
```

```
236 ret = fn.call();
```

```
Thread-42[1] cont
```

```
>
```

```
Breakpoint hit: "thread=Thread-42", clojure.lang.LockingTransaction.run(), line=
```

```
238 bci=84
```

```
238 if(info.status.compareAndSet(RUNNING, COMMITTING))
```

The lines on which the breakpoints are set are inside

```
main[1] where
```

```
[1] clojure.lang.LockingTransaction.run (LockingTransaction.java:236)
```

```
232 startPoint = readPoint;
233 startTime = System.nanoTime();
234 }
235 info = new Info(RUNNING, startPoint);
```

```

236 => ret = fn.call();
237 //make sure no one has killed us before this point, and can't from now on
238 if(info.status.compareAndSet(RUNNING, COMMITTING))
239 {
240 for(Map.Entry<Ref, ArrayList<CFn>> e : commutes.entrySet())
241 {

```

From the breakpoints, we see that one of the threads executed fn.call() once and the other executed it three times. We'll investigate why this has happened. We expect this to be because one of the transactions tried to commit but found that another transaction had committed first. Therefore, the system aborted it and retried. In our example, one thread called the body closure three times.

So what happens when one of the transactions, the code inside the sync block, is executed. Well, the sync macro looks like:

```

(defmacro sync
[flags-ignored-for-now & body]
'(. clojure.lang.LockingTransaction
(runInTransaction (fn [] [1] @body))))

```

The flags-ignored-for-now is expected to be nil – the macro really ought to check for that... I often forget to supply it and then wonder why the first form isn't executed.

Let's first step through the case where only a single transaction is running at a time.

```

user=> (.start (new Thread (fn [] (transfer 1 0 2000))))
nil

```

This goes into the runInTransaction code in LockingTransaction.

```

static public Object runInTransaction(Callable fn) throws Exception {
LockingTransaction t = transaction.get();
if(t == null)
transaction.set(t = new LockingTransaction());
if(t.info != null)
return fn.call();
return t.run(fn);
}

```

This looks for any existing transaction on the current thread, by accessing a ThreadLocal containing a LockingTransaction.

```

final static ThreadLocal<LockingTransaction> transaction = new ThreadLocal<LockingTransaction>();

```

If there is an existing transaction on the thread, we simply get on with calling the closure representing the body of the sync block; we'll look at the meaning of the info field later. Otherwise we go into the run method.

The run method is going to loop around trying to execute the transaction up to a certain retry limit. The first thing we do after getting some local variables ready is to take an integer value representing the current point in time that the transaction starts.

```

229 getReadPoint();

final private static AtomicLong lastPoint = new AtomicLong();

void getReadPoint() {
readPoint = lastPoint.incrementAndGet();
}

```

We'll now go on to make an Info object. This has the following fields; one describing the current status of the transaction and the other recording the start point in time.

```
235 info = new Info(RUNNING, startPoint);
```

```
final AtomicInteger status;  
final long startPoint;
```

We then call the closure representing the body of the sync block.

```
236 ret = fn.call();
```

The code is soon going to jump into code to access the value of account1. The binding of this Var is fetched and then we jump into the deref method of the Ref object.

```
96 public Object deref() {  
97 => LockingTransaction t = LockingTransaction.getRunning();  
98 if(t == null)  
99 return currentValue();  
100 return t.doGet(this);  
101 }
```

And the final return of this accesses the doGet method of the LockingTransaction. This first checks that the transaction hasn't been aborted, using the info object associated with the transaction.

```
351 if(!info.running())
```

If the transaction isn't running, then we throw an instance of RetryEx which signals to the runInTransaction that the transaction needs to be restarted. A single instance of this class is made and stored as a static field in the LockingTransaction class.

Next we see if we have set the value from this transaction. If we have done this previously as part of this transaction, the value will be added to a HashMap called vals,

```
final HashMap<Ref, Object> vals = new HashMap<Ref, Object>();
```

If we have then this is the value that the transaction must see.

```
353 if(vals.containsKey(ref))
```

```
354 return vals.get(ref);
```

Each Ref has a lock that we now take to allow us to do some modifications

```
final ReentrantReadWriteLock lock;
```

We take this lock,

```
357 => ref.lock.readLock().lock();
```

A Ref contains a field

```
TVal tvals;
```

These TVal instances form a chain of the value of the Ref across the various transactions.

```
public static class TVal {
```

```
Object val;
```

```
long point;
```

```
long msecs;
```

```
TVal prior;
```

```
TVal next;
```

```
TVal(Object val, long point, long msecs, TVal prior) {
```

```
this.val = val;
```

```
this.point = point;
```

```

this.msecs = msecs;
this.prior = prior;
this.next = prior.next;
this.prior.next = this;
this.next.prior = this;
}

TVal(Object val, long point, long msecs) {
this.val = val;
this.point = point;
this.msecs = msecs;
this.next = this;
this.prior = this;
}
}

```

We fetch the head of this chain,

360 => Ref.TVal ver = ref.tvals;

and then loop to find the value of the Ref as it was at the point when the transaction started.

361 do

362 {

363 if(ver.point <= readPoint)

364 return ver.val;

365 } while((ver = ver.prior) != ref.tvals);

On the way out, we unlock the Ref,

369 ref.lock.readLock().unlock();

And if we didn't find the readpoint in the list, we fail the transaction.

372 ref.faults.incrementAndGet();

373 throw retryex;

We have now successfully read the value of the Ref, as seen from the current transaction.

Soon we get into the code for setting the Ref. The code for setting jumps through into the transaction

144 return LockingTransaction.getEx().doSet(this, val);

We check that a transaction is running, and that we haven't used a commute operation on the Ref before. A commute is a means of setting a value in a way that interacts better with other running transactions.

378 => if(!info.running())

379 throw retryex;

380 if(commutes.containsKey(ref))

381 throw new IllegalStateException("Can't set after commute");

The transaction maintains a HashSet of all the Refs that have been set.

final HashSet<Ref> sets = new HashSet<Ref>();

We add the Ref to the sets collection to say that this transaction has set the Ref, and then we call the lock method. After the lock method returns, we'll add the latest value we have set into the vals collection. This was accessed by the doGet code and ensures that the transaction always sees the latest value that it set when it reads from a Ref.

if(!sets.contains(ref))

{

```
sets.add(ref);
lock(ref);
}
vals.put(ref, val);
```

The lock method inside LockingTransaction is quite long so let's step through that. First we take the writelock on the Ref.

```
113 ref.lock.writeLock().lock();
```

If another transaction has committed to the Ref, then a new TVal will have been added to the Ref. We can detect this and abort the current transaction.

```
114 if(ref.tvals != null & & ref.tvals.point > readPoint)
```

```
115 throw retryex;
```

The tinfo field of a Ref is used to record the writing transaction that "owns" it. This field may be set later in this lock method. For now, we check that there isn't a different owner. If there is, we potentially try to barge the other transaction, and if that fails, then we clear state using the stop method and throw the retry exception.

```
if(refinfo != null & & refinfo != info & & refinfo.running())
```

```
{
```

```
if(!barge(refinfo))
```

```
{
```

```
ref.lock.writeLock().unlock();
```

```
unlocked = true;
```

```
//stop prior to blocking
```

```
stop(RETRY);
```

```
synchronized(refinfo)
```

```
{
```

```
if(refinfo.running())
```

```
{
```

```
try
```

```
{
```

```
refinfo.wait(LOCK_WAIT_MSECS);
```

```
}
```

```
catch(InterruptedException e)
```

```
{
```

```
}
```

```
}
```

```
}
```

```
throw retryex;
```

```
}
```

```
}
```

stop basically cleans up

```
void stop(int status) {
```

```
if(info != null)
```

```
{
```

```
synchronized(info)
```

```
{
```

```
info.status.set(status);
```

```
info.notifyAll();
```

```
}
```

```

info = null;
vals.clear();
sets.clear();
commutes.clear();
//actions.clear();
}
}

```

The barge method checks to see if we have been running for a certain amount of time and if we started before the other transaction. If these conditions are met, we might well have done lots of useful work, so we should get the other transaction to retry and we should be allowed to continue.

```

private boolean barge(Info refinfo) {
boolean barged = false;
//if this transaction is older
// try to abort the other
if(bargeTimeElapsed() & & startPoint < refinfo.startPoint)
{
synchronized(refinfo)
{
barged = refinfo.status.compareAndSet(RUNNING, KILLED);
if(barged)
refinfo.notifyAll();
}
}
return barged;
}

```

If the lock method didn't need to barge, or if it successfully barged another transaction away, we set ourselves as the owner

143 ref.info = info;

That concludes the code for setting the Ref. We now need to go back to the code in run that tries to commit the work when the transaction finishes.

Thread-44[1] stop in clojure.lang.LockingTransaction:236

Set breakpoint clojure.lang.LockingTransaction:236

Thread-44[1] stop in clojure.lang.LockingTransaction:238

Set breakpoint clojure.lang.LockingTransaction:238

We get back to the code just after the call into the closure that executes the body of the sync.

236 ret = fn.call();

237 //make sure no one has killed us before this point, and  
can't from now on

238 => if(info.status.compareAndSet(RUNNING, COMMITTING))

239 {

The code first checks for the validity of the commutes set. We'll cover the notion of "commutes" in a later entry.

for(Map.Entry<Ref, ArrayList<CFn>> e : commutes.entrySet())

Next we take locks on all of the referenced Refs

for(Ref ref : sets)

{

```

if(!commutes.containsKey(ref))
{
    ref.lock.writeLock().lock();
    locked.add(ref);
}
}

```

And check all of the validation functions

```

for(Map.Entry<Ref, Object> e : vals.entrySet())
{
    Ref ref = e.getKey();
    ref.validate(ref.getValidator(), e.getValue());
}

```

After this we are ready to set the Refs to the newly committed values.

```

long msecs = System.currentTimeMillis();
long commitPoint = getCommitPoint();
for(Map.Entry<Ref, Object> e : vals.entrySet())
{
    Ref ref = e.getKey();
    Object oldval = ref.tvals == null ? null : ref.tvals.val;
    Object newval = e.getValue();
    if(ref.tvals == null)
    {
        ref.tvals = new Ref.TVal(newval, commitPoint, msecs);
    }
    else if(ref.faults.get() > 0)
    {
        ref.tvals = new Ref.TVal(newval, commitPoint, msecs, ref.tvals);
        ref.faults.set(0);
    }
    else
    {
        ref.tvals = ref.tvals.next;
        ref.tvals.val = newval;
        ref.tvals.point = commitPoint;
        ref.tvals.msecs = msecs;
    }
    if(ref.getWatches().count() > 0)
        notify.add(new Notify(ref, oldval, newval));
}
done = true;
info.status.set(COMMITTED);
}

```

Note that there is one other interesting point. A transaction will only be retried a certain number of times.

```
public static final int RETRY_LIMIT = 10000;
```

If we fail to commit after running it that many times, an exception is thrown.

```
throw new Exception("Transaction failed after reaching retry limit");
```

This kind of bound is really hard to test for, as there are many scenarios when it won't be hit

during testing, but it might occur very occasionally out in the field.

In the finally clause in the run method, there is also code to unlock the Ref locks we took out earlier, and code to try to carry out any agent send messages that were sent during the transaction – these get caught and batched until the transaction is successful.

```
if(done) //re-dispatch out of transaction
{
for(Notify n : notify)
{
n.ref.notifyWatches(n.oldval, n.newval);
}
for(Agent.Action action : actions)
{
Agent.dispatchAction(action);
}
}
```

Watches are functions that can be called when something changes. There were various calls to handle these in the code above.

We can now add some breakpoints to the parts of the code that throw the RetryEx, and run the original example.

```
> stop in clojure.lang.LockingTransaction:115
Set breakpoint clojure.lang.LockingTransaction:115
> stop in clojure.lang.LockingTransaction:140
Set breakpoint clojure.lang.LockingTransaction:140
```

```
user=> (do (.start (new Thread (fn [] (transfer 1 0 2000))))
(.start (new Thread (fn [] (transfer 1 1000 0)))))
```

Then we see the transaction being restarted as we expected.

```
Breakpoint hit: "thread=Thread-56", clojure.lang.LockingTransaction.lock(), line
=140 bci=126
140 throw retryex;
```

There are several points we need to make. First, the body of the sync really needs to have no side effects as it may be executed a large number of times before the transaction successfully commits. Clojure provides the handle io! macro which can be used to guard the body and which throws an exception if execution is attempted while a transaction is running.

```
(defmacro io!
[ & body]
(let [message (when (string? (first body)) (first body))
body (if message (next body) body)]
'(if (clojure.lang.LockingTransaction/isRunning)
(throw (new IllegalStateException (or message "I/O in transaction")))
(do @body))))
```

There are some other transaction related functions. Ensure, alter and commute can be used to interact with Refs during a transaction. We'll look at them in more detail later.

---

#### 4.1.13 Simple is not always easy (2010-01-21 19:55)

There was a good question on one of the Clojure lists the other day. Someone asked the equivalent of why the following code returned the usual value of 3, and didn't print the message.

```
user=> (binding [+ (fn [ & args] (println "called"))] (+ 1 2))
3
```

The answer isn't as easy as it might appear. There are several reasons why it is not possible to intercept a function using the binding mechanism.

The first is these is inlining. The compiler looks for metadata under the tags :inline and :inline-arities. If :inline-arities contains the integer corresponding to the number of arguments to which the Var is applied or it is not present or nil, then the function associated with :inline is called with those arguments.

```
user=> (defn myFunc
{:inline (fn [x y & z] '(println "inlined")) :inline-arities # {2 3} }
[ & args]
(println "Normal myFunc called"))
#'user/myFunc
user=> (myFunc)
Normal myFunc called
nil
user=> (myFunc 1)
Normal myFunc called
nil
user=> (myFunc 1 2)
inlined
nil
user=> (myFunc 1 2 3)
inlined
nil
user=> (myFunc 1 2 3 4)
Normal myFunc called
nil
user=> (defn boo [] (myFunc 1 2))
#'user/boo
user=> (boo)
inlined
nil
```

Notice how the inline function needs to return the code that the compiler goes on to process. Hence the body, in the example above, needs to be quoted so that we return the code and don't evaluate it at compile time.

```
user=> (defn myFunc
{:inline (fn [x y & z] '(println "inlined" ' x ' y ' z)) :inline-arities # {2 3} }
```

```
[ & args]
(println "Normal myFunc called")
 #'user/myFunc
user=> (myFunc 1 #'boo 20)
inlined 1 (var boo) (20)
nil
user=> (myFunc 1 #'boo (* 20 50))
inlined 1 (var boo) ((* 20 50))
nil
```

We can write some functions to walk over all of the Vars in all of the namespaces and filter out those which aren't tagged as inline.

```
user=> (defn is-inline [sym] (:inline (meta sym)))
 #'user/is-inline
user=> (defn filter-ns [ns]
(filter is-inline
(map #(ns-resolve ns %)
(map first (ns-map ns)))))

 #'user/filter-ns
user=> (set (mapcat filter-ns (all-ns)))
# { #'clojure.core/= #'clojure.core/float #'clojure.core/neg?    #'clojure.core/long #'clojure.core// #'clojure.core/== #'clojure.core/unchecked-multiply #'clojure.core/unchecked-remainder #'clojure.core/+ #'clojure.core/* #'clojure.core/bit-xor #'clojure.core/floats #'clojure.core/>= #'clojure.core/bit-and #'clojure.core/unchecked-dec      #'clojure.core/int-array      #'clojure.core/unchecked-divide #'clojure.core/unchecked-inc #'clojure.core/longs #'clojure.core/bit-or #'clojure.core/acl-one #'clojure.core/ints      #'clojure.core/unchecked-negate      #'clojure.core/<= #'clojure.core/unchecked-subtract #'clojure.core/alength #'clojure.core/aset #'clojure.core/byte #'clojure.core/aget #'clojure.core/pos?   #'clojure.core/- #'clojure.core/long-array #'clojure.core/int #'clojure.core/< #'clojure.core/boolean #'clojure.core/zero? #'clojure.core/compare #'user/myFunc #'clojure.core/dec #'clojure.core/float-array #'clojure.core/double-array #'clojure.core/> #'clojure.core/char #'clojure.core/unchecked-add      #'clojure.core/inc      #'clojure.core/double      #'clojure.core/doubles #'clojure.core/bit-not #'clojure.core/short #'clojure.core/num }
user=> (doseq [var *1]
(println var (:inline-arithies (meta var))))
 #'clojure.core/= # {2 }
 #'clojure.core/float nil
 #'clojure.core/neg? nil
 #'clojure.core/long nil
 #'clojure.core// # {2 }
 #'clojure.core/== # {2 }
 #'clojure.core/unchecked-multiply nil
 #'clojure.core/unchecked-remainder nil
 #'clojure.core/+ # {2 }
 #'clojure.core/* # {2 }
 #'clojure.core/bit-xor nil
 #'clojure.core/floats nil
 #'clojure.core/>= # {2 }
 #'clojure.core/bit-and nil
```

```

#'clojure.core/unchecked-dec nil
#'clojure.core/int-array # {1 2 }
#'clojure.core/unchecked-divide nil
#'clojure.core/unchecked-inc nil
#'clojure.core/longs nil
#'clojure.core/bit-or nil
#'clojure.core/aclone nil
#'clojure.core/ints nil
#'clojure.core/unchecked-negate nil
#'clojure.core/<= # {2 }
#'clojure.core/unchecked-subtract nil
#'clojure.core/alength nil
#'clojure.core/aset # {3 }
#'clojure.core/byte nil
#'clojure.core/aget # {2 }
#'clojure.core/pos? nil
#'clojure.core/- # {1 2 }
#'clojure.core/long-array # {1 2 }
#'clojure.core/int nil
#'clojure.core/< # {2 }
#'clojure.core/boolean nil
#'clojure.core/zero? nil
#'clojure.core/compare nil
#'user/myFunc # {2 3 }
#'clojure.core/dec nil
#'clojure.core/float-array # {1 2 }
#'clojure.core/double-array # {1 2 }
#'clojure.core/> # {2 }
#'clojure.core/char nil
#'clojure.core/unchecked-add nil
#'clojure.core/inc nil
#'clojure.core/double nil
#'clojure.core/doubles nil
#'clojure.core/bit-not nil
#'clojure.core/short nil
#'clojure.core/num nil
nil

```

We can see that the inlining arity for + is {2 }, so a three argument call ought to allow intercepting by binding, and indeed it does.

```

user=> (binding [+ (fn [ & args] (println "called"))] (+ 1 2 3))
called
nil

```

The compiler deals with inline functions in the analyzeSeq method, when it is processing a form

```

Object op = RT.first(form);
if(op == null)
throw new IllegalArgumentException("Can't call nil");
IFn inline = isInline(op, RT.count(RT.next(form)));

```

```
if(inline != null)
return analyze(context, inline.applyTo(RT.next(form)));
```

The second way a Var can avoid interception is by being a macro definition. Vars are flagged as being macros by the presence of metadata. For example, let is a macro and hence contains a :macro marker in its metadata.

```
user=> (meta #'let)
{:macro true, :ns #<Namespace clojure.core>, :name let, :file "clojure/core.clj", :line 2592,
:arglists ([bindings & body]), :doc "Evaluates the exprs in a lexical context in which the symbols inn the binding-forms are bound to their respective init-exprs or partsn therein." }
```

The compiler checks for macros just before it processes inline functions.

```
Object me = macroexpand1(form);
if(me != form)
return analyze(context, me, name);
```

Again, the macro (possibly) returns a modified form that the compiler will continue to process so the body will often need to be quoted.

```
user=> (defn myFunc
{:macro true }
[ & args]
(println "Macro function being called" args)
'(println "Generated macro code called"))
#'user/myFunc
user=> (defn boo[] (myFunc (* 10 20)))
Macro function being called ((* 10 20))
#'user/boo
user=> (boo)
Generated macro code called
nil
```

Going back to our original question, the following macros are present in the base system, and so you can't intercept these with binding.

```
user=> (defn macro-filter-ns [ns]
(filter (comp :macro meta)
(map #(ns-resolve ns %)
(map first (ns-map ns))))))
#'user/macro-filter-ns
user=> (set (mapcat macro-filter-ns (all-ns)))
# { #'clojure.core/when-first #'clojure.core/.. #'clojure.core/areduce #'clojure.core/letfn
#'clojure.core/doc #'clojure.core/amap #'clojure.core/with-out-str #'clojure.core/and #'clojure.core/-> #'clojure.main/with-bindings #'clojure.core/defmulti #'clojure.core/loop #'clojure.core/gen-interface #'clojure.core/defn #'clojure.core/delay #'clojure.core/definline #'clojure.core/memfn #'clojure.core/condp #'clojure.core/io! #'clojure.core/doseq #'clojure.core/dosync #'clojure.core/sync #'clojure.core/or #'clojure.core/with-in-str #'clojure.core/defonce #'clojure.core/let #'clojure.core/if-let #'clojure.core/pvalues #'clojure.core/ns #'clojure.core/defmacro #'clojure.core/assert-args #'clojure.core/when #'clojure.core/if-not #'clojure.core/do #'clojure.core/add-doc #'clojure.core/cond #'clojure.core/future #'clojure.core/with-open #'clojure.core/gen-class #'clojure.core/binding #'clojure.core/with-local-vars #'clojure.core/proxy-super #'clojure.core/
```

```

declare #'clojure.core/comment #'clojure.core/while #'clojure.core/proxy #'clojure.core/defn-
#'clojure.core/refer-clojure #'clojure.core/def-aset #'clojure.cor
e/defmethod #'user/myFunc #'clojure.core/lazy-cat #'clojure.core/defstruct #'clo-
jure.core/time #'clojure.core/assert #'clojure.core/when-let #'clojure.core/lazy
-seq #'clojure.core/for #'clojure.core/with-precision #'clojure.core/dotimes #'clo-
jure.core/locking #'clojure.core/fn #'clojure.core/when-not } Note that the defmacro macro
is the best way to define macros rather than using the expanded syntax above - defmacro
expands into code which calls the setMacro java method on the Var class to adjust the
metadata for :macro to true.
user=> (defmacro myMacro [] (println "expand") '(print "expanded"))
 #'user/myMacro
user=> (defn boo[] (myMacro))
expand
 #'user/boo
user=> (boo)
expandednil

```

Of course, when I said we couldn't use binding to catch uses, I meant in the sense of binding myMacro at runtime and hoping the rebound function will be used during the execution of boo. I can rebind the macro function and this will indeed be called when the compiler needs to use it to calculate the macro expansion.

```

user=> (binding [myMacro (fn [] '(print "expanded but different"))]
(eval '(defn boo[] (myMacro))))
 #'user/boo
user=> (boo)
expanded but differentnil

```

There is another set of things that the compiler treats specially - the special forms. As an example of one of these, we can look at let. Let is a macro that expands into let\*, but when we look up let\* we find that it has no Var binding.

```

user=> (macroexpand '(let [x 3] 4))
(let* [x 3] 4)
user=> (resolve 'let*)
nil
user=> (resolve 'let)
 #'clojure.core/let

```

The compiler has a set list of symbols that are treated specially.

```

static final public IPersistentMap specials = PersistentHashMap.create(
DEF, new DefExpr.Parser(),
LOOP, new LetExpr.Parser(),
RECUR, new RecurExpr.Parser(),
IF, new IfExpr.Parser(),
LET, new LetExpr.Parser(),
LETFN, new LetFnExpr.Parser(),
DO, new BodyExpr.Parser(),
QUOTE, new ConstantExpr.Parser(),
THE_VAR, new TheVarExpr.Parser(),
DOT, new HostExpr.Parser(),
ASSIGN, new AssignExpr.Parser(),
TRY, new TryExpr.Parser(),

```

```
THROW, new ThrowExpr.Parser(),  
MONITOR_ENTER, new MonitorEnterExpr.Parser(),  
MONITOR_EXIT, new MonitorExitExpr.Parser(),  
NEW, new NewExpr.Parser(),  
.....  
);
```

If we attempt to define one of these special forms, our definition will be ignored, and the system definition will be used instead.

```
user=> (defn let* [] (print "boo"))  
 #'user/let*  
user=> (let [x 3] 4)  
4  
user=> (resolve 'let*)  
 #'user/let*  
user=> (*1)  
boonil  
user=> (let* [x 3] 4)  
4
```

I'd be interested to hear of any more!



#### 4.1.14 Where is my mind? (2010-01-24 15:28)

I have just finished reading [1]The Mathematician's Brain: A Personal Tour Through the Essentials of Mathematics and Some of the Great Minds Behind Them by David Ruelle. This was an oddly intriguing book covering several stories about the author's life in Mathematics, together with his reflections on the process of mathematical discovery. The mix made a very interesting read.

1. [http://www.amazon.co.uk/Mathematicians-Brain-Personal-Essentials-Mathematics/dp/0691129827/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1264346573&sr=8-1](http://www.amazon.co.uk/Mathematicians-Brain-Personal-Essentials-Mathematics/dp/0691129827/ref=sr_1_1?ie=UTF8&s=books&qid=1264346573&sr=8-1)



#### 4.1.15 Laziness is good for you (2010-01-24 15:29)

Clojure has a nice little type called LazySeq which supports the ISeq interface, and which is accessed using the lazy-seq function in core.clj.

```
(defmacro lazy-seq
  "Takes a body of expressions that returns an ISeq or nil, and yields
  a Seqable object that will invoke the body only the first time seq
  is called, and will cache the result and return it on all subsequent
  seq calls."
  [& body]
  (list 'new 'clojure.lang.LazySeq (list* ' #^ {:once true } fn* [] body)))
```

The LazySeq type contains only two fields.

```
private IFn fn;
private ISeq s;
```

When we construct the instance we provide the generator function, and this function is used by the seq method:

```
final synchronized public ISeq seq() {
  if(fn != null)
  {
    try
    {
      s = RT.seq(fn.invoke());
      fn = null;
    }
    catch(Exception e)
    {
      throw new RuntimeException(e);
    }
  }
  return s;
}
```

to generate a sequence when it is needed in the implementation of the ISeq methods. For example, next,

```
public ISeq next() {
  seq();
  if(s == null)
  return null;
  return s.next();
}
```

Note that the seq method is synchronized, so it is thread safe, and if the method is successful, the generator is guaranteed to only be called once.

My favourite use for such a function, is to implement a sequence of all of the primes.

```
(def primes
  (letfn [(filter-by [seq divisors]
            (let [next (first seq)]
              (if (some # (= (rem next %) 0) divisors)
                  (recur (rest seq) divisors)
                  (lazy-seq
                    (cons next (filter-by (rest seq) (cons next divisors))))))]
                (filter-by (iterate (partial + 1) 2) nil)))
        user=> (take 10 primes)
        (2 3 5 7 11 13 17 19 23 29)
```

```

user=> (take 50 primes)
(2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109
113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229)

By the way lazy-seq is implemented, the above function effectively caches the calculations
that it has done so far. The trick for seeing this, is to add a print statement within the body.
This caching improves the performance, but the overhead in memory could be too much for
some scenarios.

(def primes
(letfn [(filter-by [seq divisors]
(print "called")
(let [next (first seq)]
(if (some # (= (rem next %) 0) divisors)
(recur (rest seq) divisors)
(lazy-seq
(cons next (filter-by (rest seq) (cons next divisors))))))]
(filter-by (iterate (partial + 1) 2) nil)))

user=> (take 10 primes)
(calledcalledcalled2 calledcalled3 calledcalledcalledcalled5 calledcalled7 calledcalledcalled-
called11 calledcalled13 calledcalledcalledcalled17 calledcalledcall
edcalledcalledcalled19 calledcalled23 29)
user=> (take 10 primes)
(2 3 5 7 11 13 17 19 23 29)
user=> (take 12 primes)
(2 3 5 7 11 13 17 19 23 calledcalledcalledcalled29 calledcalledcalledcalled31 37)
user=> (take 12 primes)
(2 3 5 7 11 13 17 19 23 29 31 37)

```

There's another type of object, Delay, that has the same kind of properties. A Delay, constructed using the delay macro, wraps a form and evaluates the form the first time the Delay is forced, either by calling the force function or by dereferencing it. Subsequent calls of these functions return the cached value.

```

user=> (def a (delay (do (print "called") 30)))
#'user/a
user=> @a
called30
user=> @a
30
user=> (def a (delay (do (print "called") 30)))
#'user/a
user=> (force a)
called30
user=> (force a)
30

```

#### **4.1.16 Change of heart (2010-01-24 21:10)**

There are videos of two really good videos of sessions with Mark Russinovich from PDC 09 online [1]here and [2]here. He goes into detail about the changes they have made to the Windows kernel whilst working towards Windows 7. I thought it was particularly interesting that ETW is used in the OS to allow monitoring of faulting applications and user mode scheduling looks like a powerful new feature.

- 
1. <http://microsoftpdc.com/Sessions/P09-20>
  2. <http://microsoftpdc.com/Sessions/CL29>

#### **4.1.17 I'll be back (2010-01-24 22:01)**

Recursion is one of the really nice techniques in computer science which allows proof to follow implementation. Tail recursion, and self tail recursion in particular, are often patterns that compilers go to a lot of effort to implement efficiently. Without such optimisations, it is all too easy to write something that works for test cases and then fails with a stack overflow in a production system.

In Clojure, if we compile the function

```
(defn checkEven [x]
  (letfn [(step [y status]
            (if (= x y)
                status
                (step (+ y 1) (not status))))]
    (step 0 true)))
```

We see that it works nicely for small values:

```
user=> (test.foo/checkEven 1)
false
user=> (test.foo/checkEven 2)
true
user=> (test.foo/checkEven 3)
false
```

But then fails with a stack overflow for large values.

```
user=> (test.foo/checkEven 30000)
java.lang.StackOverflowError (NO_SOURCE_FILE:0)
```

If we compile the code and look at the resulting classes, we find that the nested step function is compiled into an instance of a class foo \$checkEven \_ \_149 \$step \_ \_151 (when compiled in a namespace foo), and the invoke method on this class has the following java definition.

```
public java.lang.Object invoke(java.lang.Object, java.lang.Object) throws java.lang.Exception;
Code:
```

```

0: aload_0
1: getfield #50; //Field x:Ljava/lang/Object;
4: aload_1
5: invokestatic #60; //Method clojure/lang/Util.equiv:(Ljava/lang/Object;Ljava/lang/Object;)Z
8: ifeq 16
11: aload_2
12: goto 51
15: pop
16: aload_0
17: checkcast #62; //class clojure/lang/IFn
20: aload_1
21: getstatic #39; //Field const _2:Ljava/lang/Object;
24: invokestatic #68; //Method clojure/lang/Numbers.add:(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Number;
27: getstatic #43; //Field const _3:Lclojure/lang/Var;
30: invokevirtual #72; //Method clojure/lang/Var.get:()Ljava/lang/Object;
33: checkcast #62; //class clojure/lang/IFn
36: aload_2
37: invokeinterface #75, 2; //InterfaceMethod clojure/lang/IFn.invoke:(Ljava/lang/Object;)Ljava/lang/Object;
42: aconst_null
43: astore_1
44: aconst_null
45: astore_2
46: invokeinterface #77, 3; //InterfaceMethod clojure/lang/IFn.invoke:(Ljava/lang/Object;)Ljava/lang/Object;
51: areturn

```

The self tail call in the source happens on line 46, where we see that it isn't implemented as a tail call in the byte code.

Clojure has a way of telling the compiler to use a self tail call. This is done by using the recur special form. In our example above, we can rewrite it to.

```

(defn checkEven [x]
  (letfn [(step [y status]
            (if (= x y)
                status
                (recur (+ y 1) (not status))))]
    (step 0 true)))

```

At which point the invoke method is changed to

public java.lang.Object invoke(java.lang.Object, java.lang.Object) throws java.lang.Exception;  
Code:

```

0: aload_0
1: getfield #50; //Field x:Ljava/lang/Object;
4: aload_1
5: invokestatic #60; //Method clojure/lang/Util.equiv:(Ljava/lang/Object;Ljava/lang/Object;)Z
8: ifeq 16
11: aload_2
12: goto 43
15: pop
16: aload_1

```

```

17: getstatic #39; //Field const __2:Ljava/lang/Object;
20:     invokestatic #66; //Method clojure/lang/Numbers.add:(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Number;
23: getstatic #43; //Field const __3:Lclosure/lang/Var;
26: invokevirtual #70; //Method clojure/lang/Var.get:()Ljava/lang/Object;
29: checkcast #72; //class clojure/lang/IFn
32: aload _2
33: invokeinterface #75, 2; //InterfaceMethod clojure/lang/IFn.invoke:(Ljava/lang/Object;)Ljava/lang/Object;
38: astore _2
39: astore _1
40: goto 0
43: areturn

```

The tail call has been turned into a goto allowing the method to execute in constant stack space.

```

user=> (test.foo/checkEven 1)
false
user=> (test.foo/checkEven 2)
true
user=> (test.foo/checkEven 3)
false
user=> (test.foo/checkEven 30000)
true

```

The recur form can be used to tail call into the nearest lexical function, and is also used by the loop special form to allow looping with regard to a set of local bindings. Hence the following is much like a call to the above function with an argument of 31.

```

(let [x 31]
(loop [y 0 status true]
(if (= x y)
status
(recur (+ y 1) (not status))))))

```

Which compiles to the code

```
public java.lang.Object invoke() throws java.lang.Exception;
```

Code:

```

0: getstatic #39; //Field const __1:Ljava/lang/Object;
3: astore _1
4: getstatic #45; //Field const __3:Ljava/lang/Object;
7: astore _2
8: getstatic #72; //Field java/lang/Boolean.TRUE:Ljava/lang/Boolean;
11: astore _3
12: aload _1
13: aload _2
14: invokestatic #78; //Method clojure/lang/Util.equiv:(Ljava/lang/Object;Ljava/lang/Object;)Z
17: ifeq 25
20: aload _3
21: goto 52
24: pop
25: aload _2
26: getstatic #55; //Field const __6:Ljava/lang/Object;

```

```

29: invokestatic #84; //Method clojure/lang/Numbers.add:(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Number;
32: getstatic #59; //Field const _7:Lclojure/lang/Var;
35: invokevirtual #87; //Method clojure/lang/Var.get:()Ljava/lang/Object;
38: checkcast #89; //class clojure/lang/IFn
41: aload _3
42: invokeinterface #92, 2; //InterfaceMethod clojure/lang/IFn.invoke:(Ljava/lang/Object;)Ljava/lang/Object;
47: astore _3
48: astore _2
49: goto 12
52: areturn

```

So we've seen a couple of methods for dealing with functions that are self recursive, and that we can use a looping construct instead of defining a function. However, we haven't seen how to get a couple of mutually recursive functions to execute in constant stack space. Clojure offers the trampoline function for doing this.

(defn trampoline

"trampoline can be used to convert algorithms requiring mutual recursion without stack consumption. Calls f with supplied args, if any. If f returns a fn, calls that fn with no arguments, and continues to repeat, until the return value is not a fn, then returns that non-fn value. Note that if you want to return a fn as a final value, you must wrap it in some data structure and unpack it after trampoline returns."

```

([f]
(let [ret (f)]
(if (fn? ret)
(recur ret)
ret)))
([f & args]
(trampoline #(apply f args))))

```

The idea behind trampoline is that we have a single function sitting on the stack. This calls another function which either returns a value, in which case this value is returned, or a function is returned, in which case the function is called. This is a [1]typical implementation technique in languages that compile via CPS transformations.

We could take an example of mutually recursive functions,

```

(defn checkEven [x]
(letfn [(step-even [x]
(if (= x 0)
true
(step-odd (- x 1))))
(step-odd [x]
(if (= x 0)
false
(step-even (- x 1))))]
(step-even x)))

```

```

user=> (checkEven 10)
true

```

```
user=> (checkEven 11)
false
user=> (checkEven 10000)
java.lang.StackOverflowError (NO_SOURCE_FILE:0)
```

and transform it into the following which doesn't cause stack overflow.

```
(defn checkEven [x]
(letfn [(step-even [x]
(if (= x 0)
true
#(step-odd (- x 1))))
(step-odd [x]
(if (= x 0)
false
#(step-even (- x 1))))]
(trampoline step-even x)))
```

```
user=> (checkEven 10)
true
user=> (checkEven 11)
false
user=> (checkEven 10000)
true
```

One could imagine the compiler automatically doing the transformation to use recur, though it would make the compiler more complicated and would change the semantics of the program - the self call couldn't be intercepted using a binding of the var corresponding to the function. trampoline is a very nice mechanism with many uses, and the tidy way of defining anonymous functions in Clojure using the # reader macro makes it easy to use it for optimising mutually recursive functions.

---

1. [http://clivetong.spaces.live.com/blog/cns!3F21DF299C355E7F!248.entry?&\\_c02\\_vws=1](http://clivetong.spaces.live.com/blog/cns!3F21DF299C355E7F!248.entry?&_c02_vws=1)

#### 4.1.18 De-structure the structure (2010-01-25 22:32)

Clojure has a nice mechanism that you can use in loop and let bindings. I'm talking here about destructuring, a primitive form of pattern matching which is found in many functional programming languages.

```
user=> (let [x [1 2 3]]
(let [[a b c] x [d] x]
(list a b c d)))
(1 2 3 1)
user=> (let [x [1 2 3]]
(loop [[a b c] x [d] x]
```

```
(list a b c d))  
(1 2 3 1)
```

The really cool thing is that this destructuring is implemented in Clojure itself.

Let is essentially implemented as

```
(defmacro let  
[bindings & body]  
'(let* (destructure bindings) [1] @body))
```

and loop builds on the destructuring of let

```
(defmacro loop  
[bindings & body]  
(let [db (destructure bindings)]  
(if (= db bindings)  
'(loop* bindings @body)  
(let [vs (take-nth 2 (drop 1 bindings))  
bs (take-nth 2 bindings)  
gs (map (fn [b] (if (symbol? b) b (gensym))) bs)  
bfs ...]  
'(let bfs  
(loop* (vec (interleave gs gs))  
(let (vec (interleave bs gs))  
[2] @body)))))))
```

which both use the destructure function to produce the code for the binding.

```
user=> (destructure '[[a b c] x [d] x])  
[vec __205 x a (clojure.core/nth vec __205 0 nil) b (clojure.core/nth vec __205 1 nil) c  
(clojure.core/nth vec __205 2 nil) vec __206 x d (clojure.core/nth vec __206 0 nil)]
```

And there are two special symbols that the code handles. & binds the next variable to the sequence of the rest of the arguments after all of the required arguments (those to the left of the &).

```
user=> (destructure '[[a & b] x])  
[vec __213 x a (clojure.core/nth vec __213 0 nil) b (clojure.core/nthnext vec __213 1)]  
user=> (let [[a & b] '(1 2 3 4)] (list a b))  
(1 (2 3 4))
```

And :as gives us a sequence representing the full sequence of arguments even after we have split them into parts

```
user=> (destructure '[[a & b :as c] x])  
[vec __221 x a (clojure.core/nth vec __221 0 nil) b (clojure.core/nthnext vec __221 1) c vec __221]  
user=> (let [[a & b :as c] '(1 2 3 4)] (list a b c))  
(1 (2 3 4) (1 2 3 4))
```

The argument list of a fn also supports the & symbol for getting a sequence of the remaining arguments.

```
user=> ((fn [a & b] (list a b)) 1 2 3 4)  
(1 (2 3 4))
```

But doesn't it handle any destructuring. The code for processing these arguments is the code in the parse method in the FnMethod class in Compiler.java. This simply has a loop that maintains

a state for the current argument which can be required, or rest if it follows the & symbol. Nothing else is treated specially.

1. mailto:~@body
  2. mailto:~@body
- 

#### 4.1.19 Back to the future (to get its value) (2010-01-26 21:38)

Everyone wants to exploit concurrency these days, in order to get the best out of the new multi-core processors that are just around the corner. One of the patterns for concurrency is fork-join parallelism, where a computation is launched on another thread and the main thread comes back for the value at a later point in time.

Java offers a type called a future for handling this kind of parallelism, and this type is surfaced into Clojure via the future macro. Future takes a series of forms and executes them as a future. In does this by wrapping the forms with a fn, and then submits this to the ExecutorService in the static soloExecutor field of the Agent class.

```
(defmacro future
  [& body] '(future-call (fn [] @body)))
```

This returns a future which is wrapped for appropriate interfaces to make it accessible using the @/deref function.

```
(defn future-call
  [ #^Callable f]
  (let [fut (.submit clojure.lang.Agent/soloExecutor f)]
    (proxy [clojure.lang.IDeref java.util.concurrent.Future] []
      (deref [] (.get fut))
      (get [] (.get fut))
      ([timeout unit] (.get fut timeout unit)))
      (isCancelled [] (.isCancelled fut))
      (isDone [] (.isDone fut))
      (cancel [interrupt?] (.cancel fut interrupt?)))))
```

This operation will block if the computation isn't yet finished. Hence in the example below we assigned the future via a def so that the printer didn't wait until the value was available in order to print it.

```
user=> (def a (future (. Thread (sleep 10000)) 20))
#'user/a
user=> (. a isDone)
false
user=> @a
20
user=> (. a isDone)
true
```

```
Futures can be cancelled
user=> (def a (future (. Thread (sleep 10000)) 20))
#'user/a
user=> (. a (cancel true))
true
user=> (. a isDone)
true
user=> @a
java.util.concurrent.CancellationException (NO_SOURCE_FILE:0)
```

Futures are used to implement some other higher order operations that allow parallel execution. pmap takes a function and some sequences and applies the function to elements of the sequences(s), generating a lazy-seq which uses to calculate the values in parallel.

```
user=> (pmap (fn [x y] (+ x y)) '(1 2) '(3 4))
(4 6)
user=> (pmap (fn [x y] (+ x y)) '(1 2 3 4 5 6) '(3 4 5 6 7 8))
(4 6 8 10 12 14)
```

By printing inside the function we can see the calculation as it proceeds – the results will now be a list of nil values as returned by println.

```
user=> (pmap (fn [x] (println x)) '(1 2 3 4 5 6 7))
(3
45
2
1

nil6
nil nil 7
nil nil nil nil)
```

pmap is used to implement the function pcalls and the macro pvalues which call a sequence of functions and a group of values respectively.

```
user=> (pcalls (fn[] (println 1)) (fn[] (println 2)))
(21

nil nil)
user=> (pvalues (print 1) (print 2))
1(2nil nil)
```

---

#### 4.1.20 Macro mayhem (2010-01-28 07:11)

Macros are one of those things that set Lisp based languages apart from the rest, so Clojure has them too. A macro is a function that is called by the compiler when it is compiling code, and this function then returns more code which the compiler sets about compiling instead of the original.

A simple example would be something like the following:

```
user=> (defmacro myMacro[] (println "expansion") '(print x))
```

#'user/myMacro Our use of the defmacro macro makes things look complicated. However, if we expand the definition we made, we see that defmacro expands into the normal defn which simply defines a function, then goes on to call the setMacro method on the Var, and finally returns the new Var.

```
user=> (macroexpand '(defmacro myMacro[] (println "expansion") '(print x)))
```

```
(do (clojure.core/defn myMacro [] (println "expansion") (quote (print x))) (. (var myMacro) (setMacro)) (var myMacro))
```

The setMacro method simply adds the :macro metadata to the Var

```
user=> ^ #'myMacro
```

```
{:macro true, :ns #<Namespace user>, :name myMacro, :file "NO_SOURCE_PATH", :line 127, :arglists ([])}
```

We can call this macro function and see that all it does is return the expansion.

```
user=> ( #'myMacro)
```

```
expansion
```

```
(print x)
```

When the compiler processes a form, one of checks it does is to see if the first element of any sequence is a macro. In this case, it expands the macro and continues processing on the result of the expansion. Hence the use of a macro looks like a function call, but the effect is very different. In the following we see that the “expansion” gets printed when the compiler processes the code, and the resulting form, which prints 20, is patched into the definition.

```
user=> (defn macroUser[] (let [x 20] (myMacro) (myMacro)))
```

```
expansion
```

```
expansion
```

```
#'user/macroUser
```

```
user=> (macroUser)
```

```
2020nil
```

Of course, if we are going to be returning code, we need a templating method for generating chunks of source code. This is what the syntax quote offers us. Syntax quote behaves a lot like quote, but we'll see in a moment that it allows value substitution. The main different is that the symbols in the generated body have their namespace set to the current namespace.

```
user=> '(foo 2)
```

```
(foo 2)
```

```
user=> '(foo 2)
```

```
(user/foo 2)
```

The main advantage of syntax quote is that it allow substitution. The use of ` causes the following form to be evaluated, with the result being substituted into the generated form. Typically the substitution happens of a variable using syntax like `x, but any form can follow the `, so we could use something like `(print x) to get the side-effect. The @ variant expands the following form to evaluate to a list and splices the list into the generated code.

```
user=> (let [x 2] '(foo `x))
```

```
(user/foo 2)
```

```
user=> (let [x 2] '(foo `x `x `x))
```

```
(user/foo 2 2 2)
```

```

user=> (let [x '(2 3 4)] '(foo 1 @x 5))
(user/foo 1 2 3 4 5)

user=> (let [x '(2 3 4)] '(foo (println x) x @x))
(2 3 4)
(user/foo nil (2 3 4) 2 3 4)

```

The syntax quote actually generates code to produce the expansion, so that side-effects can happen. Quoting the result allows us to see what the expansion is.

```

user=> (let [x 2] "(foo x)
(clojure.core/seq (clojure.core(concat (clojure.core/list (quote user/foo)) (clojure.core/list x)))

```

The big advantage of macros is that they allow us to write constructs that don't evaluate arguments in the standard order. If we want to implement something like if,

```
(if (= 1 2) (print "a") (print "b"))
```

then we want to evaluate the condition and only one of the branches. If "if" is a standard Clojure function, then the arguments are all going to be evaluated before the code for "if" runs (assuming no exceptions). Using a macro we can expand to use existing special forms to avoid this.

One gotcha in Common Lisp is that you have to be careful to avoid macros in the expanded form shadowing variables from the outer lexical scope. So, for example, if you write a macro to time something

```
(defmacro get-execution-time [& body]
'(let [before (. System (nanoTime))]
(do @body)
(- (. System (nanoTime)) before)))
```

you'd get incorrect behaviour if you tried to

```
(let [before 10]
(get-execution-time
(print before)))
```

as this would expand to

```
(let [before 10]
(let [before (. System (nanoTime))]
(do (print before))
(- (. System (nanoTime)) before)))
```

and hence the (print before) would print the wrong value, as the before in the print gets redirected to point to the start time instead of the outer definition.

The way that Clojure's syntax quote namespace qualifies the symbols, and the rule that namespace qualified symbols cannot be locally bound, means that the macro as given above fails in Clojure.

```

user=> (defmacro get-execution-time [& body]
'(let [before (. System (nanoTime))]
(do @body)
(- (. System (nanoTime)) before)))
#'user/get-execution-time
user=> (get-execution-time (. Thread (sleep 100)))
java.lang.Exception: Can't let qualified name: user/before (NO_SOURCE_FILE:152)

```

Inside the syntax quote, we can generate new non-namespaced symbols by post-fixing a symbol with a hash. This causes the reader to generate a new symbol which is substituted for the hash-postfixed version. Most importantly, within the same syntax quoted form, the same symbol can be used multiple times.

```
user=> '(a # a #)
(a __116 __auto __ a __116 __auto __)
```

We can use this to easily fix our macro.

```
user=> (defmacro get-execution-time [ & body]
'(let [before # (. System (nanoTime))]
(do @body)
(- (. System (nanoTime)) before #)))
 #'user/get-execution-time
user=> (get-execution-time (. Thread (sleep 100)))
99666927
user=> (get-execution-time (. Thread (sleep 101)))
100825314
```

All of this is handled in the LispReader.java code, in particular the two reader macros

```
macros[""""] = new SyntaxQuoteReader();
macros[' '] = new UnquoteReader();
```

SyntaxQuoteReader sets up a map for mapping the hash post-fixed symbols to the generated symbol ("gensym") that it maps to. The reader is called recursively when we start processing the """ character in the SyntaxQuoteReader. This recursive read will return @ as uses of the symbols

```
static Symbol UNQUOTE = Symbol.create("clojure.core", "unquote");
static Symbol UNQUOTE_SPLICING = Symbol.create("clojure.core", "unquote-splicing");
```

and the code in SyntaxQuoteReader walks the returned form to get rid of uses of these symbols, converting the code into a sequence of calls to do the actual construction as we saw in the example above.

Suggested practice is that macros are deterministic and hygenic – they do not have side effects.

Also, that macros that look like normal function calls should evaluate their arguments in the normal order, and should evaluate the arguments only once. Hence a macro like

```
user=> (defmacro broken[x y] '(do y x x))
 #'user/broken
```

is bad because it doesn't behave much like a normal function with regard to argument evaluation. We can see this by using a side-effecting function as an argument.

```
user=> (broken (print 1) (print 2))
```

```
21nil
```

where a normal function would give

```
user=> (defn foo[x y])
 #'user/foo
user=> (foo (print 1) (print 2))
12nil
```

The macro should be corrected to

```
user=> (defmacro brokenNot[x y]
'(let [x # x y # y]
```

```
(do y # x # x #))
#'user/brokenNot
user=> (brokenNot (print 1) (print 2))
12nil
user=> (macroexpand '(brokenNot (print 1) (print 2)))
(let* [x __185 __auto __ (print 1) y __186 __auto __ (print 2)] (do y __186 __auto __ x __185
__auto __ x __185 __auto __))
```

The above also highlights that the best way to debug the macro expansion is often to call macroexpand and macroexpand-1. Macroexpand-1 checks the first element of the sequence to see if it is macro and does the expansion if it is. macroexpand calls macroexpand-1 until there is no more expansion possible. Both are useful when debugging macros.

```
user=> (macroexpand '(brokenNot (print 1) (print 2)))
(let* [x __185 __auto __ (print 1) y __186 __auto __ (print 2)] (do y __186 __auto __ x __185
__auto __ x __185 __auto __))
```

```
user=> (macroexpand-1 '(brokenNot (print 1) (print 2)))
(clojure.core/let [x __185 __auto __ (print 1) y __186 __auto __ (print 2)] (do y __186 __auto
__ x __185 __auto __ x __185 __auto __))
user=> (macroexpand-1 *1)
(let* [x __185 __auto __ (print 1) y __186 __auto __ (print 2)] (do y __186 __auto __ x __185
__auto __ x __185 __auto __))
```

Macros are often used to implement internal DSLs and as such can be a very powerful aid to abstraction.

---

#### 4.1.21 Off you go (2010-01-30 14:46)

Clojure has really good integration with the java platform. It's really easy to instantiate java classes and call their methods. Another very useful Clojure construct is the proxy macro which allows the easy construction of a Clojure object that can be called from Java. The constructed proxy can inherit from an existing class and can implement a set of interfaces. Methods can be defined to override those of any base class, and interface methods may be implemented too. By default, interface methods default to throwing an UnsupportedOperationException.

Clojure functions are based on the AFunction abstract class which defines invoke methods that throw an exception representing a wrong number of arguments, with the intention that a user function will implement the overload which has the appropriate number of arguments.

Hence we could define an instance inheriting from this abstract class and then call it.

```
user=> ((proxy [clojure.lang.AFunction] []))
java.lang.IllegalArgumentException: Wrong number of args passed to: AFunction (NO_SOURCE
_FILE:0)
```

We can start defining some of the overloaded methods.

```
user=> ((proxy [clojure.lang.AFunction] []
```

```
(invoke [x]
(prinln "Called with " x)
123))
20)
Called with 20
123
```

And can define multiple overloads at the same time.

```
user=> ((proxy [clojure.lang.AFunction] []
(invoke
([x]
(prinln "Called with " x)
47)
([ x y]
(prinln "Called with two args" x y) 33)))
20)
Called with 20
47
```

```
user=> ((proxy [clojure.lang.AFunction] []
(invoke
([x]
(prinln "Called with " x)
47)
([ x y]
(prinln "Called with two args" x y) 33)))
20 30)
Called with two args 20 30
33
```

A this variable is bound during the execution of the method, allowing us to call back into the proxy object.

```
user=> ((proxy [clojure.lang.AFunction] []
(invoke
([x]
(prinln "Called with " x)
47)
([ x y]
(prinln "Called with two args" x y)
(. this (invoke x))
(. this (invoke y))))))
1 2)
Called with two args 1 2
Called with 1
Called with 2
47
```

It's really neat that we can use this to check which underlying methods are called by the Clojure functions.

```
user=> (next (proxy [clojure.lang.ISeq] []
(next [] (list 30))))
(30)
```

There is a related macro, gen-class, that can be used to generate a loadable class which can potentially load Clojure (the runtime) and even define a main method so that the class can be used to start an application. gen-class can be added into the expansion of the ns macro if :gen-class arguments are provided. The usual expansion for gen-class uses the \*compile-files\* Var to only do work if we are in the middle of compiling something. The body of the expansion uses a function, generate-class, to generate the bytecode for a class which is then written using the clojure.lang.Compiler/writeClassFile function.

generate-class takes many options. We could use it as below to generate a class file, test/bar.class on the class path.

```
clojure.core=> (generate-class {:name 'test.bar :init 'startup :main true })
["test/bar" #<byte[] [B@1c64ed8>]
clojure.core=> (let [[cname bytecode] *1]
(clojure.lang.Compiler/writeClassFile cname bytecode))
nil
```

The generated class has the following definition.

```
public class bar
{
private static final Var startup __var = Var.internPrivate("clojure.core", "-startup");
private static final Var main __var = Var.internPrivate("clojure.core", "-main");
private static final Var hashCode __var = Var.internPrivate("clojure.core", "-hashCode");
private static final Var clone __var = Var.internPrivate("clojure.core", "-clone");
private static final Var equals __var = Var.internPrivate("clojure.core", "-equals");
private static final Var toString __var = Var.internPrivate("clojure.core", "-toString");

static
{
RT.var("clojure.core", "load").invoke("/clojure/core");
}

public String toString() ...
public boolean equals(Object paramObject) ...
public Object clone() ...
public int hashCode() ...
public static void main(String[] paramArrayOfString)...
}
```

As you can see, a prefix is added to the method names when constructing the Clojure names. These are searched for in a namespace which defaults to one named after the class file. The main function has the following definition:

public static void main(java.lang.String[]);

Code:

```
0: getstatic #29; //Field main __var:Lclojure/lang/Var;
3: dup
4: invokevirtual #65; //Method clojure/lang/Var.isBound:()Z
7: ifeq 16
10: invokevirtual #69; //Method clojure/lang/Var.get:()Ljava/lang/Object;
13: goto 18
16: pop
17: aconst _null
```

```

18: dup
19: ifnull 38
22: checkcast #56; //class clojure/lang/IFn
25: aload _0
26: invokestatic #120; //Method clojure/lang/RT.seq:(Ljava/lang/Object;)Lclojure/lang/ISeq;
29: invokeinterface #124, 2; //InterfaceMethod clojure/lang/IFn.applyTo:(Lclojure/lang/ISeq;)Ljava/lang/Object;
34: pop
35: goto 48
38: new #79; //class java/lang/UnsupportedOperationException
41: dup
42: ldc #126; //String clojure.core/-main not defined
44: invokespecial #84; //Method java/lang/UnsupportedOperationException."<
init>":(Ljava/lang/String;)V
47: athrow
48: return

```

We can put all of this together to get a very simple application. In a file helloworld.clj, we can define a namespace and a main method.

```

(ns test.helloworld
(:gen-class))
(defn -main[]
(prinln "Hello World!"))

```

We can then compile this using the compile function.

```

user=> (compile 'test.helloworld)
test.helloworld

```

We get the following classes generated

```

30/01/2010 14:05 914 helloworld $ _main __339.class
30/01/2010 14:05 1,783 helloworld.class
30/01/2010 14:05 2,260 helloworld __init.class

```

And finally, we can run the resulting classes.

```

C:UsersCliveDesktopclojure-1.0.0>java -cp classes;src/clj test.helloworld
Hello World!

```

We can push this a bit further to generate a windowing application.

```

(ns test.helloworld
(:gen-class)
(:import (javax.swing JButton JPanel JFrame)))
(defn start-app []
(let [button (JButton. "Press me!")
panel ( JPanel.)
frame ( JFrame. "Hello App")]
(. button addActionListener
(proxy [java.awt.event.ActionListener] []
(actionPerformed [event]

```

```
(println "Don't press that again!"))
(. panel setOpaque true)
(. panel add button)
(. frame setContentPane panel)
(. frame setSize 300 100)
(. frame setVisible true))

(defn -main[]
(println "Hello World!")
(start-app))
```

We can test this at the REPL using

```
user=> (load-file "src/clj/test/helloworld.clj")
#'test.helloworld/-main
user=> (*1)
Hello World!
```

and [1]the window appears. We can test things interactively, and reload until we get things working. Finally, we can compile to get a set of classes for the final standalone application. This last step is very neat when compared with the classic style of Common Lisp environment. In those environments, the debugging experience is better as you develop the application inside the Common Lisp environment, but application delivery becomes harder as it is hard to pull the application out of the environment.

```
user=> (compile 'test.helloworld)
test.helloworld
```

```
30/01/2010 14:30 1,029 helloworld $start _app __176 $fn __178.class
30/01/2010 14:30 2,458 helloworld $start _app __176.class
30/01/2010 14:30 1,025 helloworld $ _main __183.class
30/01/2010 14:05 914 helloworld $ _main __339.class
30/01/2010 14:30 1,783 helloworld.class
30/01/2010 14:30 2,858 helloworld __init.class
6 File(s) 10,067 bytes
```

```
C:UsersCliveDesktopclojure-1.0.0>java -cp classes;src/clj test.helloworld
Hello World !
```

---

1. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Pictures/clojureapp.jpg>

## 4.2 February

### 4.2.1 There are no bugs here! (2010-02-07 22:38)

[1]Advanced Windows Debugging by Mario Hewardt and Daniel Pravat

I came across this book by chance in Cambridge library. Unfortunately, I didn't get time to read it too thoroughly in the time I could have it on loan, but the parts I did get to read were exceptionally good.

Over the years I've used several low level debuggers for debugging windows at the machine code level (and have also done lots of debugging on Unix systems to using adb and gdb). I used cdb in my early career working on Lisp systems, and have used, more recently, windbg and the excellent SOS extension for debugging .NET. This book covered everything I've ever learned in this area, organising the material logically and explaining all sorts of related concepts. It then moved on to cover common scenarios and how they could be debugged, offering background details on the implementation of parts of Windows as a means of mapping out a strategy for the debugging.

In particular, the book had chapters on

- (1) Basic debugger commands and how to set up symbols files to make it easier to debug code.
- (2) Debugger internals covering how the operating system sends events to the debugger, how the debugger interacts with Windows' structured exceptions and how the debugger inserts break instructions into the instruction stream to regain control at certain points. It went on to talk about thread suspension and resumption, and the related notions of freezing and thawing.
- (3) The Windows calling conventions together with a detailed look at the many forms of stack corruption.
- (4) Windows heaps and the built in allocation mechanisms together with a detailed look at debugging heap corruption.
- (5) Security and how to look at the security context of a process or a thread.
- (6) Troubleshooting communications failures including LPC
- (7) Debugging resource leaks such as the failure to free handles and memory.
- (8) Discussion of synchronisation and debugging deadlocks.
- (9) Writing debugger extension commands and how to take various kinds of memory dump.
- (10) Great chapters on the changes to support 64 bit operating systems and the changes made with the introduction of Vista.

A fantastic book covering both the technical means and the strategy for debugging many kinds of system failure.

- 
1. [http://www.amazon.co.uk/Advanced-Windows-Debugging-Administering-Addison-Wesley/dp/0321374460/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1265494079&sr=8-1](http://www.amazon.co.uk/Advanced-Windows-Debugging-Administering-Addison-Wesley/dp/0321374460/ref=sr_1_1?ie=UTF8&s=books&qid=1265494079&sr=8-1)

#### **4.2.2 Exceptionally good (2010-02-07 22:39)**

Clojure interacts really well with the JVM. Exceptions in Clojure are simply implemented as standard Java exceptions, and there are special forms for throwing exceptions and for doing the normal try-catch-finally exception handling pattern.

Nomal Java exceptions can be thrown by most expressions

```
user=> (/ 1 0)
java.lang.ArithmaticException: Divide by zero (NO_SOURCE_FILE:0)
```

And we can also easily create instances of Exception classes and throw these using the throw special form.

```
user=> (throw (Exception.))
java.lang.Exception (NO_SOURCE_FILE:0)
```

The REPL also catches exceptions when it evaluates the form it has just read. This code binds the exception into a Var so that it can be examined.

```
(try
(let [input (read request-prompt request-exit)]
(or ( # {request-prompt request-exit} ) input)
(let [value (eval input)]
(print value)
(set! *3 *2)
(set! *2 *1)
(set! *1 value)))
(catch Throwable e
(caught e)
(set! *e e)))
```

This gives the behaviour

```
user=> (throw (Exception.))
java.lang.Exception (NO_SOURCE_FILE:0)
user=> (bean *e)
{:stackTrace #<StackTraceElement[] [Ljava.lang.StackTraceElement;@9ba134>, :message "java.lang.Exception (NO_SOURCE_FILE:0)", :localizedMessage "java.lang.Exception (NO_SOURCE_FILE:0)", :class clojure.lang.Compiler $CompilerException, :cause #<Exception java.lang.Exception> }
```

try-catch-finally takes the form with the usual semantics. The first clause is found which handles the class of the exception, and this clause is used to process the exception, binding a local variable to the handled exception. Any finally clause is executed on exit from the scope.

```
(try
(/ 1 0)
(catch ArithmaticException e (println "Arithmatic exception"))
(catch Throwable e (println "Throwable"))
	finally
	(println "Done"))
```

---

### 4.2.3 This and that (2010-02-07 22:39)

I was listening to the [1]Java Posse the other day, in particular the special episode covering the announcement that closures would be making their way into the Java language version

7. During the conversation, the guest pointed out that some of the closure proposals had alternatives for the meaning of “this” inside the closure body. In both Java and C # the closure instances are represented using instances of compiler generated classes, into which the closed over code is injected as a method. In some of the Java proposals, this abstraction would be leaky in that the code could see this special class as the type of its “this” pointer.

We can see how the C # compiler has to go to some effort for getting a “this” pointer inside the closure to refer to the instance at the time of closure construction. First, let’s take a very simple Action returning method in a class called Program.

```
class Program
{
    Action Makelt()
    {
        return () => { Console.WriteLine(this); };
    }
}
```

and look at how it is called in the following code in a Main method.

```
Program xxx = new Program();
Action yyy = xxx.Makelt();
yyy();
```

The IL for the Makelt method is

```
.method private hidebysig instance class [[2]System.Core][[3]System.Action [4]Makelt() cil
managed
{
    .maxstack 3
    .locals init (
        [0] class [[5]System.Core][6]System.Action CS $1 $0000)
    L_0000: nop
    L_0001: ldarg.0
    L_0002: ldftn instance [7]void [8]ConsoleApplication1.Program::[9]<Makelt>b __0()
    L_0008: newobj instance [10]void [[11]System.Core][12]System.Action::[13].ctor([14]object ,
    [15]native int)
    L_000d: stloc.0
    L_000e: br.s L_0010
    L_0010: ldloc.0
    L_0011: ret
}
```

Action is a subclass of MulticastDelegate, and the instance is constructed on line L \_0008 using the value resulting from Idarg.0. This will be the “this” value for the method call, which in our example will be an instance of Program.

The closure code of the Action is compiled as follows. Note that “this” is simply compiled as a reference to argument 0, the target instance of the instance method call.

```
.method private hidebysig instance [16]void [17]<Makelt>b __0() cil managed
{
    .custom instance [18]void [[19]mscorlib][20]System.Runtime.CompilerServices-
    s.CompilerGenerat edAttribute::[21].ctor()
```

```

.maxstack 8
L_0000: nop
L_0001: ldarg.0
L_0002: call [22]void [[23]mscorlib][24]System.Console::[25]WriteLine([26]obje ct)
L_0007: nop
L_0008: ret
}

```

Now let's have a look at a closure that actually closes over some values.

```

Action Makelt2()
{
int x = 30;
return () => { Console.WriteLine(this); Console.WriteLine(x); };
}

```

executed by the code

```

yyy = xxx.Makelt2();
yyy();

```

In this case we are closing over the value in the local variable x. This means that the compiler needs to generate a class for holding the values that are closed over.

```
.class auto ansi sealed nested private beforefieldinit [27]<>c _ _DisplayClass2 extends
[[28]mscorlib][29]System.Object
```

Fields are added to this class for holding the closed over value, and there is now a need to keep hold of the “this” pointer for the instance in which the closure was constructed – the “this” pointer at execution time will be an instance of this new class, so all uses of “this” in the body need to redirect via the added field.

```
.field public class [30]ConsoleApplication1.Program <>4 __this
.field public [31]int32 x
```

For example, the first call to WriteLine in the body needs to use this extra “this” pointer. This implementation technique is really clever, though I kind of feel that everything would have been easier if the notion of a closure was built into the virtual machine from the start.

```

.method public hidebysig instance [32]void [33]<MakeIt2>b__1() cil managed
{
    .maxstack 8
    L_0000: nop
    L_0001: ldarg.0
    L_0002: ldfld class [34]ConsoleApplication1.Program
    [35]ConsoleApplication1.Program/[36]<>c__DisplayClass2::[37]<>4__this
    L_0007: call [38]void
    [[39]mscorlib][40]System.Console::[41]WriteLine([42]object)
        L_000c: nop
        L_000d: ldarg.0
        L_000e: ldfld [43]int32
    [44]ConsoleApplication1.Program/[45]<>c__DisplayClass2::[46]x
        L_0013: call [47]void
    [[48]mscorlib][49]System.Console::[50]WriteLine([51]int32)
        L_0018: nop
}
```

```

    L_0019: ret
}

.method private hidebysig instance class [[52]System.Core][53]System.Action
[54]MakeIt2() cil managed
{
    .maxstack 3
    .locals init (
        [0] class [55]ConsoleApplication1.Program/[56]<>c__DisplayClass2
CS$<>8__locals3,
        [1] class [[57]System.Core][58]System.Action CS$1$0000)
    L_0000: newobj instance [59]void
[60]ConsoleApplication1.Program/[61]<>c__DisplayClass2::[62].ctor()
    L_0005: stloc.0
    L_0006: ldloc.0
    L_0007: ldarg.0
    L_0008: stfld class [63]ConsoleApplication1.Program
[64]ConsoleApplication1.Program/[65]<>c__DisplayClass2::[66]<>4__this
    L_000d: nop
    L_000e: ldloc.0
    L_000f: ldc.i4.s 30
    L_0011: stfld [67]int32
[68]ConsoleApplication1.Program/[69]<>c__DisplayClass2::[70]x
    L_0016: ldloc.0
    L_0017: ldftn instance [71]void
[72]ConsoleApplication1.Program/[73]<>c__DisplayClass2::[74]<MakeIt2>b__1()
    L_001d: newobj instance [75]void
[[76]System.Core][77]System.Action::[78].ctor([79]object, [80]native int)
    L_0022: stloc.1
    L_0023: br.s L_0025
    L_0025: ldloc.1
    L_0026: ret
}

```

1. [http://javaposse.com/index.php?post\\_id=577886](http://javaposse.com/index.php?post_id=577886)
2. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089>
3. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089/System.Action>
4. [http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program/MakeIt\(\):System.Action](http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program/MakeIt():System.Action)
5. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089>
6. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089/System.Action>
7. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Void>
8. [http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program/MakeIt\(\):System.Action](http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program/MakeIt():System.Action)

on1.Program  
9. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program/<MakeIt>b\_\_0()  
10. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Void  
11. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089  
12. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089/System.Action  
13. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089/System.Action/.ctor(Object,IntPtr)  
14. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Object  
15. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.IntPtr  
16. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Void  
17. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program/<MakeIt>b\_\_0()  
18. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Void  
19. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089  
20. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Runtime.CompilerServices.CompilerGeneratedAttribute  
21. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Runtime.CompilerServices.CompilerGeneratedAttribute/.ctor  
22. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Void  
23. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089  
24. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Console  
25. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Console.WriteLine(Object)  
26. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Object  
27. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c\_\_DisplayClass2  
28. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089  
29. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Object  
30. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program  
31. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.IntPtr  
32. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Void  
33. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c\_\_DisplayClass2.<MakeIt2>b\_\_1()  
34. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program  
35. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica

tion1.Program  
36. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c\_\_DisplayClass2  
37. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c\_\_DisplayClass2/<>4\_\_this:ConsoleApplic  
38. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Vo  
id  
39. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089  
40. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Co  
nsole  
41. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Co  
nsole/WriteLine(Object)  
42. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.O  
bject  
43. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.In  
t32  
44. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica  
tion1.Program  
45. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica  
tion1.Program.<>c\_\_DisplayClass2  
46. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica  
tion1.Program.<>c\_\_DisplayClass2/x:Int32  
47. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Vo  
id  
48. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089  
49. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Co  
nsole  
50. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Co  
nsole/WriteLine(Int32)  
51. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.In  
t32  
52. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089  
53. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089/Syste  
m.Action  
54. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica  
tion1.Program/MakeIt2():System.Action  
55. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica  
tion1.Program  
56. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica  
tion1.Program.<>c\_\_DisplayClass2  
57. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089  
58. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089/Syste  
m.Action  
59. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Vo  
id  
60. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica  
tion1.Program  
61. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica  
tion1.Program.<>c\_\_DisplayClass2  
62. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplica

```

tion1.Program.<>c__DisplayClass2.ctor()
63. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program
64. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program
65. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c__DisplayClass2
66. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c__DisplayClass2<>4__this:ConsoleApplic
67. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.In
t32
68. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program
69. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c__DisplayClass2
70. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c__DisplayClass2/x:Int32
71. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Vo
id
72. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program
73. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c__DisplayClass2
74. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://ConsoleApplication3:1.0.0.0/ConsoleApplication1.Program.<>c__DisplayClass2<>MakeIt2>b__1()
75. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Vo
id
76. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089
77. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089/Syste
m.Action
78. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Core:3.5.0.0:b77a5c561934e089/Syste
m.Action.ctor(Object,IntPtr)
79. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.O
bject
80. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.In
tPtr

```

---

#### 4.2.4 There's more than one way to do that (2010-02-09 22:14)

Clojure has a simple way of defining multi-methods. If you've seen multi-methods before in Common Lisp, these are a very simplified version, without the ability to call next-method, without any kind of topological sorting of applicable methods being used to find the most specific and without the notion of an effective method. As an aside, the definitive book on the design of Clos, [1]The Art of the Metaobject Protocol, is one of the best computing book I've read, taking

the reader through the Common Lisp object system and explaining the design decisions that were required along the way.

Clojure's system for calling multimethods certainly isn't as beautiful as Clos, but it is quite neat and simple, and we'll take a look at it here.

In order to work out which method is closest to a set of arguments, we need the notion of a hierarchy which is realised in an `isA` function together with the notion of one type dominating another. In the implementation in `MultiFn`

```
private boolean isA(Object x, Object y) throws Exception {  
    return RT.booleanCast(isa.invoke(hierarchy.deref(), x, y));  
}  
  
private boolean dominates(Object x, Object y) throws Exception {  
    return prefers(x, y) || isA(x, y);  
}
```

where `prefers` is defined as

```
private boolean prefers(Object x, Object y) throws Exception {  
    IPersistentSet xprefs = (IPersistentSet) getPreferTable().valAt(x);  
    if(xprefs != null & & xprefs.contains(y))  
        return true;  
    for(ICollection ps = RT.seq(parents.invoke(y)); ps != null; ps = ps.next())  
    {  
        if(prefers(x, ps.first()))  
            return true;  
    }  
    for(ICollection ps = RT.seq(parents.invoke(x)); ps != null; ps = ps.next())  
    {  
        if(prefers(ps.first(), y))  
            return true;  
    }  
    return false;  
}
```

This code uses the clojure builtin `parents` function, for checking the relationship between things. The prefer table is a table that can be set using the functions `prefer-method`, and accessed using `prefers`, and is used for breaking ties when there are multiple methods that are equally related to the given item. We have an example of this at the end of this post.

We can access the global hierarchy using the `parents` and `ancestors` functions.

```
foo=> (parents (class a))  
# {java.lang.Object }  
foo=> (ancestors (class a))  
# {java.lang.Object }  
foo=> (parents (class b))  
# {clive.test.A }  
foo=> (ancestors (class b))  
# {java.lang.Object clive.test.A }
```

The `isa` function defaults to the Clojure `isa?` function. This has the following comment in `core.clj`

```
(defn isa?
  "Returns true if (= child parent), or child is directly or indirectly derived from
  parent, either via a Java type inheritance relationship or a
  relationship established via derive. h must be a hierarchy obtained
  from make-hierarchy, if not supplied defaults to the global
  hierarchy"
  ([child parent] (isa? global-hierarchy child parent))
  ([h child parent] ...))
```

There are a series of functions including parents, ancestors, descendants that use a hierarchy to relate various objects, with the user being allowed to define extra relationships using derive and underive.

MultiFn itself tries to cache a lot of the calls to these relationship functions to speed up subsequent calls. It also caches the results of its search to find the particular method that matches a set of arguments.

In the following, we will assume the following Java class definitions have been compiled and are placed on the class path.

```
package clive.test;
public class A { }
public class B extends A { }
```

We can import these classes into the current session.

```
user=> (import 'clive.test.A)
nil
user=> (import 'clive.test.B)
nil
```

And then define instances of these classes.

```
user=> (def a (clive.test.A.))
#'user/a
user=> (def b (clive.test.B.))
#'user/b
```

We can do some simple tests on the isa? function.

```
user=> (isa? (class a) clive.test.A)
true
user=> (isa? (class a) clive.test.B)
false
user=> (isa? (class b) clive.test.A)
true
user=> (isa? (class b) clive.test.B)
true
```

We can use defmulti to define new function. We have to supply a function that is executed on the arguments in order to get the values that are used to determine the applicable method. In this case, we'll use the class function to convert the arguments into suitable values. The defmulti function can take arguments giving the equivalent of an isa? function for defining the hierarchy, and a value that is used for a default method. These default to the values isa? and :default respectively.

```
user=> (defmulti foo class)
 #'user/foo
user=> (defmethod foo clive.test.A [x] (println "called on A" x))
#<MultiFn clojure.lang.MultiFn@476128>
user=> (foo a)
called on A #<A clive.test.A@5fcf29>
nil
user=> (foo b)
called on A #<B clive.test.B@125844f>
nil
```

We didn't define a method that is applicable to either the java.lang.Integer class or the :default, so calling the multi function with 10 gives an error.

```
user=> (foo 10)
java.lang.IllegalArgumentException: No method in multimethod 'foo' for dispatch
value: class java.lang.Integer (NO_SOURCE_FILE:0)
```

If we also define a method on B, then the most applicable method will be used, as determined by the dominates method above.

```
user=> (defmethod foo clive.test.B [x] (println "called on B"))
#<MultiFn clojure.lang.MultiFn@476128>
user=> (foo a)
called on A #<A clive.test.A@5fcf29>
nil
user=> (foo b)
called on B
nil
user=> (foo 10)
java.lang.IllegalArgumentException: No method in multimethod 'foo' for dispatch
value: class java.lang.Integer (NO_SOURCE_FILE:0)
```

We can define a default method.

```
user=> (defmethod foo :default [x] (println "called on default"))
#<MultiFn clojure.lang.MultiFn@476128>
user=> (foo a)
called on A #<A clive.test.A@5fcf29>
nil
user=> (foo b)
called on B
nil
user=> (foo 10)
called on default
nil
```

We can remove methods. If we remove the method on A, then calling foo with instances of A will lead to the default method being called.

```

user=> (remove-method foo clive.test.A)
#<MultiFn clojure.lang.MultiFn@476128>
user=> (foo a)
called on default
nil
user=> (foo b)
called on B
nil
user=> (foo 10)
called on default
nil

```

We can also determine which method is going to be called. This is the return value of the get-method function and we can then apply this to some arguments. When the method is applied, no additional checking is performed, as we can see by getting a method and then applying it to a completely different set of arguments.

```

user=> (get-method foo (class b))
#<user $fn __146 user $fn __146@15a6029>
user=> (apply *1 [b])
called on B
nil
user=> (apply *2 [20])
called on B
nil

```

The isa? function also works on vectors of elements, allowing specialisation on multiple arguments.

```

foo=> (isa? [(class a)(class b)] [(class a)(class a)])
true

```

```

foo=> (defmulti foo2 (fn [x y] [(class x)(class y)]))
#'foo/foo2
foo=> (defmethod foo2 [clive.test.A clive.test.A] [x y] (println "both A"))
#<MultiFn clojure.lang.MultiFn@13f210f>
foo=> (defmethod foo2 [clive.test.B clive.test.B] [x y] (println "both B"))
#<MultiFn clojure.lang.MultiFn@13f210f>
foo=> (foo2 a a)
both A
nil
foo=> (foo2 a b)
both A
nil
foo=> (foo2 b a)
both A
nil
foo=> (foo2 b b)
both B
nil

```

We can quickly demonstrate the prefer behaviour by extending our example classes

```

public interface C { }
public class D extends A implements C { }

```

```
user=> (import '(clive.test C))
nil
user=> (import '(clive.test D))
nil
```

We can now define methods on the class A and the interface C.

```
user=> (defmulti foo3 class)
 #'user/foo3
user=> (defmethod foo3 clive.test.A [x] (println "A"))
#<MultiFn clojure.lang.MultiFn@13785d3>
user=> (defmethod foo3 clive.test.C [x] (println "C"))
#<MultiFn clojure.lang.MultiFn@13785d3>
user=> (def d (clive.test.D.))
 #'user/d
```

A call on an instance of D is ambiguous – one method is on the interface and one on the class so there is no ordering.

```
user=> (foo3 d)
java.lang.IllegalArgumentException: Multiple methods in multimethod 'foo3' match dispatch
value: class clive.test.D -> interface clive.test.C and class clive.test.A, and neither is preferred
(NO_SOURCE_FILE:0)
```

We can set one to be preferred over the other.

```
user=> (prefer-method foo3 clive.test.A clive.test.C)
#<MultiFn clojure.lang.MultiFn@13785d3>
user=> (foo3 d)
A
nil
```

And can check to see what preferences have been set so far.

```
user=> (prefers foo3)
{clive.test.A # {clive.test.C} }
```

This is a very concise way of avoiding a function with a lot of conditions for selecting a particular case, though the construct doesn't appear to be used very often in Clojure.

---

1. [http://www.amazon.co.uk/Art-Metaobject-Protocol-G-Kiczales/dp/0262610744/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1265657528&sr=8-1](http://www.amazon.co.uk/Art-Metaobject-Protocol-G-Kiczales/dp/0262610744/ref=sr_1_1?ie=UTF8&s=books&qid=1265657528&sr=8-1)

#### 4.2.5 Say hello to FeeFee (2010-02-11 22:42)

At work I get to do some really interesting things with C # and .Net. One of the things I've done lately is some work on calculating extra data alongside generated code so that it can

be stepped easily inside Visual Studio. It turns out that this all revolves around the idea of a sequence point.

Say we have the following console application, and we start it by stepping into it.

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(1);
        Console.WriteLine(2);
        Console.WriteLine(3);
    }

    // Foo bar
}
```

Inside Visual Studio, we can hit F10 which stops on the opening { of the Main method. Flipping into the disassembly view, we can see that the debugger has calculated a mapping between the source code and the x86 instructions.

```
00000024 nop
Console.WriteLine(1);
00000025 mov ecx,1
0000002a call FFAC292C
0000002f nop
Console.WriteLine(2);
00000030 mov ecx,2
00000035 call FFAC292C
0000003a nop
Console.WriteLine(3);
0000003b mov ecx,3
00000040 call FFAC292C
00000045 nop
}
00000046 nop
00000047 lea esp,[ebp-0Ch]
0000004a pop ebx
0000004b pop esi
```

How does it do this, you might ask. If we ildasm the resulting assembly, asking for line information, we get a main method that looks like this.

```
D:\temp4\SequencePointDemo\SequencePointDemo\bin\Debug>"c:\Programs\Microsoft Visual Studio 8\SDK\v2.0\Bin\ildasm.exe" /LINENUM /OUT=program.il SequencePointDemo.exe

.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    // Code size 23 (0x17)
    .maxstack 8
    .language ' {3F5162F8-07C6-11D3-9053-00C04FA302A1 }', ' {994B45C4-E6E9-11D2-903F-00C04FA302A1 }', ' {5A869D0B-6611-11D3-BD2A-0000F80849BD }'
    .line 10,10 : 9,10 'D:\temp4\SequencePointDemo\SequencePointDemo\Program.cs'
```

```

IL_0000: nop
.line 11,11 : 13,34 "
IL_0001: ldc.i4.1
IL_0002: call void [mscorlib]System.Console::WriteLine(int3 2)
IL_0007: nop
.line 12,12 : 13,34 "
IL_0008: ldc.i4.2
IL_0009: call void [mscorlib]System.Console::WriteLine(int3 2)
IL_000e: nop
.line 13,13 : 13,34 "
IL_000f: ldc.i4.3
IL_0010: call void [mscorlib]System.Console::WriteLine(int3 2)
IL_0015: nop
.line 14,14 : 9,10 "
IL_0016: ret
} // end of method Program::Main

```

The .line directives are the key. They provide a pair of start and end line numbers, followed by a pair of start and end character positions on those lines, optionally followed by a file name in which those lines reside.

We can demonstrate this by changing the first .line directive after the nop to

```

IL_0000: nop
.line 16,16 : 13,34 "
IL_0001: ldc.i4.1
IL_0002: call void [mscorlib]System.Console::WriteLine(int3 2)

```

If we then generate a new exe

```
D:\temp4\SequencePointDemo\SequencePointDemo\bin\Debug>c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\ilasm.exe /debug program.il
```

and step into that (by setting it as the start executable in Visual Studio), we now get the mapping

```

static void Main(string[] args)
{
    00000000 push ebp
    00000001 mov ebp,esp
    00000003 push eax
    00000004 mov dword ptr [ebp-4],ecx
    00000007 cmp dword ptr ds:[00952E14h],0
    0000000e je 00000015
    00000010 call 7917A651
    00000015 nop
// Foo bar
    00000016 mov ecx,1
    0000001b call 02A83E04
    00000020 nop
    Console.WriteLine(1);
    Console.WriteLine(2);
    00000021 mov ecx,2
    00000026 call 02A83E04

```

```

0000002b nop
Console.WriteLine(3);
0000002c mov ecx,3
00000031 call 02A83E04
00000036 nop
}
00000037 nop
00000038 mov esp,ebp
0000003a pop ebp
0000003b ret

```

Stepping using F10, takes us first to the line with the {, then to the // Foo bar line, then to Console.WriteLine(2), and Console.WriteLine(3).

Of course, that begs the question of how you turn off the mapping, perhaps because the same IL instruction is used by two different lines of the high level source language. And that's where 0xfffffe comes in. Using this as the line number turns off the mapping.

Changing the middle .line directive to use this special marker

```

IL_0007: nop
.line 0xfffffe,0xfffffe : 13,34 "
IL_0008: ldc.i4.2
IL_0009: call void [mscorlib]System.Console::WriteLine(int3 2)

```

We now get the following disassembly

```

static void Main(string[] args)
{
00000000 push ebp
00000001 mov ebp,esp
00000003 push eax
00000004 mov dword ptr [ebp-4],ecx
00000007 cmp dword ptr ds:[00952E14h],0
0000000e je 00000015
00000010 call 7917A651
00000015 nop

// Foo bar
00000016 mov ecx,1
0000001b call 02A83E04
00000020 nop
00000021 mov ecx,2
00000026 call 02A83E04
0000002b nop
Console.WriteLine(1);
Console.WriteLine(2);
Console.WriteLine(3);
0000002c mov ecx,3
00000031 call 02A83E04
00000036 nop
}
00000037 nop
00000038 mov esp,ebp

```

```
0000003a pop ebp  
0000003b ret
```

And the stepping happens as the line with {, then the line with // Foo Bar followed by Console.WriteLine(3).

Hence it is the sequence points that give the necessary mapping information for the source level debugging. Depending on a variety of other factors, the debugging infrastructure inside the CLR only keeps this mapping approximate when JIT optimisations are allowed, leading to stepping behaviour that doesn't always follow what the sequence points demand. Indeed some of the ngen optimisations don't track the sequence points, and the DebuggableAttribute allows an assembly author to tell the system not to track all sequence points as possible step points. However, it is amazing that the only information that the compiler needs to supply is the equivalent of the .line directives to allow this debugging to happen.

I should perhaps add that this sequence point information is encoded into the pdb file, into which you'd also add extra information to record local variable mappings. These files can be written using various COM interfaces that are present on Windows machines.

---

#### 4.2.6 That's very convoluted (2010-02-13 10:10)

[1]Dr Euler's Fabulous Formula: Cures Many Mathematical Ills by Paul J Nahin

A great book that starts with complex numbers and Euler's formula which relates some of the fundamental constants of mathematics, and then moves on to period functions and Fourier series, showing lots of practical applications along the way. This was a thoroughly interesting read. In particular I enjoyed the solutions to the runner's problem - when a runner is going around a closed circuit, is a constant wind an advantage (and the answer is no), and the dog walker's distance - if a dog stays constantly 1 metre to the right of the walker, how much extra distance does the dog walk.

---

1. [http://www.amazon.co.uk/Dr-Eulers-Fabulous-Formula-Mathematical/dp/0691118221/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1266055013&sr=1-1](http://www.amazon.co.uk/Dr-Eulers-Fabulous-Formula-Mathematical/dp/0691118221/ref=sr_1_1?ie=UTF8&s=books&qid=1266055013&sr=1-1)

#### 4.2.7 Short and Sweet (2010-02-14 17:37)

[1]Mathematics: A Very Short Introduction by Tim Gowers

A very accessible introduction to some of the concepts of university mathematics with good discussions of abstraction, axioms, infinity and limits. One of the best discussions I've read on the problems of the parallel postulate, which the book uses to introduce spherical and hyperbolic geometry. Lots of interesting content.

- 
1. [http://www.amazon.co.uk/Mathematics-Very-Short-Introduction-Introductions/dp/0192853619/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1266147551&sr=1-1](http://www.amazon.co.uk/Mathematics-Very-Short-Introduction-Introductions/dp/0192853619/ref=sr_1_1?ie=UTF8&s=books&qid=1266147551&sr=1-1)

#### **4.2.8 That's not what you said last time (2010-02-14 17:40)**

---

#### **4.2.9 It's not Pandora's (2010-02-20 22:35)**

I recently came across another set of tools for working with Clojure. [1]Clojure Box is a distribution of Clojure 1.1.0 together with emacs and the slime library. You simply install Clojure box onto your system, and when you start it you end up with a running emacs which is talking to a clojure repl which is running as the inferior Lisp. This means it's really easy to edit and execute Clojure code with the environment giving you access to things like symbol completion, macro expansions and documentation.

- 
1. <http://clojure.bighugh.com/>

#### **4.2.10 JSON told me the answer (2010-02-20 23:03)**

---

[1]Google Web Toolkit Applications by Ryan Dewsbury

I picked this book up in Cambridge library, and it was a fantastic read. I keep coming across toolkits that translate languages into javascript in order to make it easy to write applications that run in a browser on the client. Tools for F # and Clojure are two that I have recently come across. This book covers what looks like the most complete toolkit of them all, the Google Web Toolkit which uses Java as the source language.

The book's introduction makes a good argument as to why this kind of technology is really useful - Ajax applications are easy to deploy, without the need to have any kind of plug-in installed

in the client, into browsers which are now standards compliant enough to generate pretty rich user experiences. GWT allows Java skills, and the usual Java development environments such as Eclipse, to be used in the development of these kinds of applications.

The first section of the book provides a good introduction to the GWT. It covers the user interface elements and layouts, describing how they can be customised using CSS and extended by hooking into user defined javascript functions, how events are handled, how communication can happen with the server using the GWT RPC mechanisms and other techniques for fetching data. It moves on to a detailed account of how to get the toolkit running inside Eclipse and how the final application can be debugged using the toolkit's built-in browser and then deployed as a standalone application.

The second section of the book, rich web applications by example, shows the development of five applications using the GWT. These range from a simple Gadget Desktop which demonstrates the various layouts and how to implement drag and drop, to an instant messenger application. This second section shows a number of interesting techniques that I hadn't seen before. The first, was fetching data from sites that would normally be blocked by the browser's same origin check, by having the data fetched using a script tag which would fetch javascript that would execute a given callback function with the fetched data. The second was avoiding polling for events by having the client call into the server which would potentially block for up to 30 seconds, at which point it would return and the client would then reissue the call. This can be made quite scalable, as long as the server can handle the messages without using a thread per call, which is shown to be achievable in a number of web servers such as jetty.

A really good introduction to a technology that I would like to find time to explore in the future.

1. [http://www.amazon.co.uk/Google-Toolkit-Applications-Ryan-Dewsbury/dp/0321501969/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1266702736&sr=8-1-spell](http://www.amazon.co.uk/Google-Toolkit-Applications-Ryan-Dewsbury/dp/0321501969/ref=sr_1_1?ie=UTF8&s=books&qid=1266702736&sr=8-1-spell)
- 

#### **4.2.11 How did you do that (2010-02-22 18:18)**

[1]Masterminds of Programming: Conversations with Creators of Major Programming Languages by Federico Biancuzzi and Shane Warden

A reasonably interesting set of interviews with the creators of a number of programming languages such as C # and Java. Depending on the interviewee and the questions they were asked, the write-ups were a little variable in that some were very interesting and some verged on being boring. Not too much technical depth, and certainly not a book that I'll be reading again.

1. [http://www.amazon.co.uk/Masterminds-Programming-Conversations-Creators-Languages/dp/0596515170/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1266860477&sr=8-1](http://www.amazon.co.uk/Masterminds-Programming-Conversations-Creators-Languages/dp/0596515170/ref=sr_1_1?ie=UTF8&s=books&qid=1266860477&sr=8-1)

---

#### 4.2.12 I don't want another argument (2010-02-22 18:19)

Having had a very brief look at the compilation of generic classes, I thought it would be good to see how generic methods are handled by the CLR. The following example contains a `TestMethod` specialized on a generic parameter.

```
class Program
{
    class S<T>
    {
        public S(T x)
        {
            Console.WriteLine("start" + x);
        }
    }

    void TestMethod<T>(T x)
    {
        S<T> a = new S<T>(x);
        S<T> b = new S<T>(x);
    }

    static void Main()
    {
        object foo = new object();
        Program z = new Program();
        z.TestMethod(foo);
        z.TestMethod(foo);
        z.TestMethod("2");
    }
}
```

If we look at the first call to `TestMethod` in `Main` we see it has the code

```
z.TestMethod(foo);
00000069 push 979938h
0000006e mov edx,dword ptr [ebp-3Ch]
00000071 mov ecx,dword ptr [ebp-40h]
00000074 cmp dword ptr [ecx],ecx
00000076 call FD2BB058
0000007b nop
```

The surprising thing is that an extra argument, `979938h`, is being passed into the method. Why do we need the extra method parameters? Well, the body of `TestMethod` creates a new instance, and in order to do this we need a type handle for the newly created type. The purpose of this extra argument is to cache the handles for new instances.

At the point of instantiation in the `TestMethod`, we can see how the `this` table is used.

```
S<T> a = new S<T>(x);
00000043 mov eax,dword ptr [ebp+8]
00000046 mov eax,dword ptr [eax+0Ch]
```

```
00000049 add eax,4
0000004c mov dword ptr [ebp-50h],eax
0000004f mov eax,dword ptr [ebp-50h]
00000052 mov eax,dword ptr [eax]
00000054 mov dword ptr [ebp-54h],eax
00000057 cmp dword ptr [ebp-54h],0
0000005b jne 00000073
0000005d push 0
0000005f push dword ptr [ebp-50h]
00000062 push 0
00000064 mov ecx,dword ptr [ebp+8]
00000067 mov edx,1B000001h
0000006c call 7689C368
00000071 jmp 00000076
00000073 mov eax,dword ptr [ebp-54h]
00000076 mov ecx, eax
00000078 call FD2A09CC
0000007d mov dword ptr [ebp-4Ch],eax
00000080 mov edx,dword ptr [ebp-40h]
00000083 mov ecx,dword ptr [ebp-4Ch]
00000086 call FD2BAFA0
0000008b mov eax,dword ptr [ebp-4Ch]
0000008e mov dword ptr [ebp-44h],eax
```

After the load at line 43, we have eax containing

0x00979938 43000001 00050004 009798f8 00979914

and the 0xC offset points to the table. Here 7933061c is the handle for the object type.

0x00979914 7933061c 00000000 00000000 00000000

By the time we get to 76, the table has been filled with another type handle by the call at 6c.

0x00979914 7933061c 00979ad8 00000000 00000000

By the time we get to 7D, we have constructed the new object on the heap, using the handle from the table as the header

0x01394D14 00979ad8 00000000

And we are now ready to call the constructor at 86.

Next time we go through the code, we lift the type handle from the table without the need to call any code to fill it. Zero is used to represent an unfilled entry with the conditional code at 53 using any previously calculated value.

#### 4.2.13 I thought I'd tested that! (2010-02-25 20:31)

I've been spending some time looking at Clojure 1.1, in particular the changes between version 1.0 and 1.1.

The first thing I found was that there is now a test framework contained within the base image. Clojure.test contains a means of defining tests, together with some functions that run all of the tests in a given package.

We can, for example, define a simple function and then define some tests to fix its behaviour.

```
user> (use 'clojure.test)
nil
user> (defn my-function [x] (* x 2))
 #'user/my-function
user> (deftest test-my-function
(is (= (my-function 10) 20))
(is (= (my-function 1) 2)))
 #'user/test-my-function
user> (run-tests)
```

Testing user

Ran 1 tests containing 2 assertions.

0 failures, 0 errors.

```
{:type :summary, :test 1, :pass 2, :fail 0, :error 0 }
```

If the function gets redefined, it will then be picked up by the tests.

```
user> (defn my-function [x] (* x 3))
 #'user/my-function
user> (run-tests)
```

Testing user

```
FAIL in (test-my-function) (NO_SOURCE_FILE:1)
expected: (= (my-function 10) 20)
actual: (not (= 30 20))
```

```
FAIL in (test-my-function) (NO_SOURCE_FILE:1)
expected: (= (my-function 1) 2)
actual: (not (= 3 2))
```

Ran 1 tests containing 2 assertions.

2 failures, 0 errors.

```
{:type :summary, :test 1, :pass 0, :fail 2, :error 0 }
```

There is also the means to set up fixtures, which can be executed around each test or around each batch of tests. For example, we could set up a test that checks the dynamic context.

```
user> (def *a* 0
 #'user/*a*
user> (def *b* 0
 #'user/*b*
user> (deftest check-context
(is (= *a* 1)))
```

```
(is (= *b* 1))
#'user/check-context
```

This fails as we have things now.

```
user> (run-tests)
```

Testing user

```
FAIL in (check-context) (NO _SOURCE _FILE:1)
```

```
expected: (= *a* 1)
```

```
actual: (not (= 0 1))
```

```
FAIL in (check-context) (NO _SOURCE _FILE:1)
```

```
expected: (= *b* 1)
```

```
actual: (not (= 0 1))
```

Ran 1 tests containing 2 assertions.

2 failures, 0 errors.

```
{:type :summary, :test 1, :pass 0, :fail 2, :error 0 }
```

We can define fixtures to set up the context in which the test runs. These fixture functions take a function as an argument. They then establish the environment and call the passed in function, and afterwards clean up again. We set the fixtures by using the use-fixtures function.

```
user> (defn fixture1 [f]
(binding [*a* 1]
(f)))
#'user/fixture1
user> (defn fixture2 [f]
(binding [*b* 1]
(f)))
#'user/fixture2
user> (use-fixtures :each fixture1 fixture2)
{:clojure.test/each-fixtures ( #<user $fixture1 _ _3374 user $fixture1 _ _3374@199f443>
#<user $fixture2 _ _3395 user $fixture2 _ _3395@c47498>) }
user> (run-tests)
```

Testing user

Ran 1 tests containing 2 assertions.

0 failures, 0 errors.

```
{:type :summary, :test 1, :pass 2, :fail 0, :error 0 }
```

#### 4.2.14 Take your PIC (2010-02-25 20:31)

Doing interviews is great, as there's always something interesting that comes up in the conversation. In the latest one that I was part of, the issue of calling via an interface on the CLR was mentioned. I vaguely remembered a [1]blog post from a long time ago that talks about how the implementation changed in the transition from .NET 1.1 to .NET 2. In .NET 2, the system uses what I'd call a PIC, an inline cache, for handling the call efficiently.

Of course I had to try this out for myself. To enable debugging inside Visual Studio, you need to be running the example as a release build, have "just my code" turned off and have "suppress JIT optimisations on load" turned off, and also have "Enable Unmanaged Debugging" set on the Debug tab of the project.

With that we can try running this small example.

```
interface ITest
{
void CallMethod();
}

class Program : ITest
{
public void CallMethod()
{
Console.WriteLine("boo");
}

static void Main(string[] args)
{
ITest target = new Program();
for (int i =0; i < 1000; i++)
target.CallMethod();
}
}
```

Setting a breakpoint on the target.CallMethod() line, when we run it the first time, we see that the code here disassembles to:

```
for (int i =0; i < 1000; i++)
00000011 xor edi,edi
target.CallMethod();
00000013 mov ecx,esi
00000015 call dword ptr ds:[00980010h]
for (int i =0; i < 1000; i++)
0000001b inc edi
0000001c cmp edi,3E8h
```

If we step over the call instruction and then example the address 980010, we see that it contains 986012. Disassembling this location, by going to the immediate window, loading sos and using the !u instruction we find:

```
.load sos
extension C:WINDOWSMicrosoft.NETFrameworkv2.0.50727sos.dll loaded
!u 00986012
PDB symbol for mscorwks.dll not loaded
```

```
Unmanaged code  
00986012 50 push eax  
00986013 6800000300 push 30000h  
00986018 E908F25079 jmp 79E95225
```

If we now let the code run for a while, say by running until the breakpoint is hit 5 times, we see that the code has been rewritten to

```
!u 00987012  
Unmanaged code  
00987012 81394C339700 cmp dword ptr [ecx],97334Ch  
00987018 0F85F32F0000 jne 0098A011  
0098701E E985905200 jmp 00EB00A8
```

In the case when we take the jmp, we target the method itself.

```
!u 00EB00A8  
Normal JIT generated code  
ConsoleApplication43.Program.CallMethod()  
Begin 00eb00a8, size 1a  
>>> 00EB00A8 55 push ebp  
00EB00A9 8BEC mov ebp,esp  
00EB00AB E840D24378 call 792ED2F0 (System.Console.get_Out(), mdToken: 06000772)  
00EB00B0 8BC8 mov ecx, eax  
00EB00B2 8B1530203202 mov edx,dword ptr ds:[02322030h] ("boo")  
00EB00B8 8B01 mov eax,dword ptr [ecx]  
00EB00BA FF90D8000000 call dword ptr [eax+000000D8h]  
00EB00C0 5D pop ebp  
00EB00C1 C3 ret
```

So what is 97334c. This value is simply the type handle for the Program type as we can see from the instance that we construct at the start of the Main method.

```
ITest target = new Program();  
00000000 push ebp  
00000001 mov ebp,esp  
00000003 push edi  
00000004 push esi  
00000005 mov ecx,97334Ch  
0000000a call FFAB1FAC  
0000000f mov esi,eax
```

The idea here is simple. At a given call site, if we call via an interface on a particular object type, then we are likely to call again with the same object type in subsequent calls. Hence we can optimise by hardwiring code which checks for the expected type and then jumps into the target method if the type is the same. If the type is different then we can branch to a fix-up routine instead.

Of course, it's all a matter of tradeoffs. Rewriting the code means flushing instruction caches so we'd better make sure that we're calling on the same object type multiple times. That's why it takes a few calls before this optimisation happens... the runtime can monitor activity at the call site and dynamically optimise. All rather cool, I think you'll agree.

1. <http://blogs.msdn.com/vancem/archive/2006/03/13/550529.aspx>
- 

#### **4.2.15 Watch out for the apple! (2010-02-28 14:57)**

[1]Gravity's Arc: The story of gravity from Aristotle to Einstein and beyond by David Darling

This was a really interesting read, covering man's understanding of gravity from the early days of Aristotle who got it completely wrong, though the experimental results of Galileo and Kepler which were given a good theoretical underpinning by Newton, and on to the conversion of gravity in to a property of the geometry of space-time by Einstein. The book then looks forward into more recent astronomical results concerning dark-matter and dark-energy, and the attempts to unify the various forces via string theory and loop-quantum gravity.

Along the way, there are a lot of interesting personal stories. I enjoyed the stories of the mind boggling 40g accelerations that Stapp subjected his body to using rocket sleds in the 1940s in order to understand the effects on the human body; after one such ride he found out that a certain Captain Murphy has misconnected all of the sensors leading to a mass of zero readings, and the genesis of Murphy's law.

1. [http://www.amazon.co.uk/Gravity-Arc-Gravity-Aristotle-Einstein/dp/0471719897/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1267368442&sr=1-1](http://www.amazon.co.uk/Gravity-Arc-Gravity-Aristotle-Einstein/dp/0471719897/ref=sr_1_1?ie=UTF8&s=books&qid=1267368442&sr=1-1)
- 

#### **4.2.16 A little lopsided (2010-02-28 14:57)**

[1]Fearless Symmetry: Exposing the hidden patterns of numbers by Avner Ash and Robert Gross

An awesome maths book that covers the material needed for a proof of Fermat's Last Theorem. The book is written for the amateur mathematician, with lots of examples demonstrating the various definitions and omitting most of the proofs. In part one, the book starts with group theory, and then moves onto equations and varieties and on to quadratic reciprocity. Part two covers Galois theory, the field of algebraic numbers, matrices and group representations. It then moves on to cover the Galois group of a polynomial, conjugacy classes and Frobenius elements. Part three takes this material and moves on to the weak and strong reciprocity laws and sketches how the material can be put together to get a proof of Fermat's Last Theorem.

The book tries to make all of the material very accessible. Definitions and their motivation are explained very clearly with simple, well-explained examples. A very well written book.

1. [http://www.amazon.co.uk/Fearless-Symmetry-Exposing-Patterns-Numbers/dp/0691138710/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1267367782&sr=8-1](http://www.amazon.co.uk/Fearless-Symmetry-Exposing-Patterns-Numbers/dp/0691138710/ref=sr_1_1?ie=UTF8&s=books&qid=1267367782&sr=8-1)

---

#### 4.2.17 Thanks for your contribution (2010-02-28 22:18)

It's been fun looking through the clojure-contrib directories of the 1.1 distribution. There's tons of really useful source code and examples. These are some of the things that I've noticed.

There is a contribution that makes it possible to do more advanced macrology. One thing that could be quite useful is symbol-macrolet, a means of having a symbol expand into a form at macroexpansion time.

```
user> (use 'clojure.contrib.macro-utils)
nil
user> (symbol-macrolet [x (print hello)] (let [hello 27] x))
27nil
user> (macroexpand '(symbol-macrolet [x (print hello)] (let [hello 27] x)))
(do (let* [hello 27] (print hello)))
```

There are some utilities that make it easier to work at the REPL. The source macro attempts to find the definition of a given symbol and print it out at the REPL.

```
user> (use 'clojure.contrib.repl-utils)
nil
user> (source use)
(defn use
  "Like 'require, but also refers to each lib's namespace using
  clojure.core/refer. Use :use in the ns macro in preference to calling
  this directly."
```

'use accepts additional options in libspecs: :exclude, :only, :rename.

The arguments and semantics for :exclude, :only, and :rename are the same as those documented for clojure.core/refer."

```
[ & args] (apply load-libs :require :use args))
```

nil There's a library for defining and using monads. The monads defined by this library don't look nearly as neat as definitions inside Haskell, which has syntactic sugar which support their use. However, this is a library that I'll be playing with more in the future. The following example, which comes in the library examples, shows the use of a continuation monad that the library defines.

```
user> (use 'clojure.contrib.monads)
nil
user> (run-cont
(domonad cont-m
[x (m-result 1)
y (m-result 2)]
(+ x y)))
3
user> (def continuation nil)
```

```
#'user/continuation
user> (run-cont
(domonad cont-m
[x (m-result 1)
y (call-cc (fn [c] (def continuation c) (c 2)))]
(+ x y)))
3
user> (run-cont (continuation 5))
6
```

while we're talking about Clojure, the language gets a good mention on the [1]Herding Code podcast.

---

1. <http://herdingcode.com/?p=238>

## 4.3 March

### 4.3.1 Its important to have a good memory (2010-03-06 12:20)

There's always a lot of talk about profiling your application, because it's really hard to tell from the source code where the bottlenecks are going to be. There's a [1]fascinating example here that shows that there can be subtle interactions between the language and the runtime that would be hard to predict without going to the trouble of actually exercising code.

---

1. <http://www.lshift.net/blog/2010/02/28/memory-matters-even-in-erlang>

### 4.3.2 Model it and pass it on! (2010-03-08 16:03)

[1]ASP.NET MVC In Action by Jeffrey Palermo, Ben Scheirman and Jimmy Bogard

This felt like a really introduction to the ASP.NET MVC framework. It takes an application for scheduling code camps as a running example throughout the book. There are initial chapters explaining why the MVC framework is much better than the more traditional ASP.NET with these chapters giving a brief overview of the major components such as controllers, views, model binding and routing. Later in the book there are entire chapters devoted to these and other parts of the MVC framework.

There's a big emphasis on testability and simplicity, and some of the authors have started an MvcContrib project which offers extra useful functionality which is used throughout. There are chapters giving a brief comparison against MonoRail and Ruby-on-Rails and also chapters on best practices. There are examples of the use of the various extensibility points of the MVC framework. All of this was really useful for getting a feel for this web development platform.

This seemed like a book written by developers for developers and was a great read. Recommended!

1. [http://www.amazon.co.uk/ASP-NET-MVC-Action-Jeffrey-Palermo/dp/1933988622/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1268062910&sr=8-1](http://www.amazon.co.uk/ASP-NET-MVC-Action-Jeffrey-Palermo/dp/1933988622/ref=sr_1_1?ie=UTF8&s=books&qid=1268062910&sr=8-1)
- 

#### 4.3.3 Bring out your dead (2010-03-20 21:54)

Something that has been missing for the .NET framework for a while is a weak hashtable. This kind of datatype is useful as it allows you to associate data with an object in a way that allows the data to be thrown away if the object is garbage collected. The problem here is that we don't want the presence of the object as a hashtable key to keep the object alive and we don't want to add any fields to the object to hold this data.

.NET 4 has a new ConditionalWeakTable type that addresses some of these problems. It isn't a true Dictionary as it only supports reference equality, but it is useful for storing this extra information; I think that the MEF framework uses the type for this purpose.

We can quickly test this type by defining an object that has a name and prints out a message when it is finalised.

```
class Probe
{
    public string Name { get; set; }
    public Probe(string name)
    {
        Name = name;
    }
    Probe()
    {
        Console.WriteLine("Finalizer called on {0}", Name);
    }
}
```

We can build a weak table suitable for holding these probes.

```
static ConditionalWeakTable<Probe, Probe> s _TestTable = new ConditionalWeakTable<Probe, Probe>();
```

We'll use a simple function for populating the table - we don't want the lifetime of local variables affecting the garbage collection.

```

static Probe Populate()
{
    Probe key1 = new Probe("permanent");
    Probe key2 = new Probe("Temporary");
    Probe data1 = new Probe("Data1");
    Probe data2 = new Probe("Data2");
    s_TestTable.Add(key1, data1);
    s_TestTable.Add(key2, data2);
    return key1;
}

```

The GetValue method also needs a delegate for returning a default if the key is not found in the table.

```

static Probe Default()
{
    throw new Exception("Not found");
}

```

We can now run a small example, which fills the table with two keys and associated data. We keep hold of one of the keys while we do a garbage collection, and we see the other key and its associated datum are collected. When the program finishes, we see the other key and value getting collected.

```

static void Main(string[] args)
{
    Probe key1 = Populate();
    GC.Collect();
    GC.WaitForPendingFinalizers();
    Console.WriteLine("Object alive: {0}", s_TestTable.GetValue(key1, _ => Default()).Name);
}

```

Running this we get the output.

```

Finalizer called on Data2
Finalizer called on Temporary
Object alive: Data1
Finalizer called on Data1
Finalizer called on permanent

```

---

#### **4.3.4 That's an interesting development (2010-03-21 20:31)**

I was lucky in the week to get a day at [1]DevWeek. On the Wednesday I got to attend 4 sessions with a choice of 8 tracks across a very wide range of topics.

Cool graphics, hot code: ten visual effects to make you the envy of your peers by Jeff Prosise

This was really interesting stuff. I've never used WPF or Silverlight for real, but have read quite a lot about both of these technologies. The presenter showed a number of interesting visual effects and then explained how they could be implemented in Silverlight.

The first technique was to use `WriteableBitmap`. This class can be used to capture the image generated by the rendering of a number of XAML objects, and this image can then be used for implementing a number of effects, such as zooming, magnifiers and reflections. The next technique was behaviours, wrapping up pieces of code so that they can be attached to objects, notably by drag and drop inside Expression Blend. He then moved on to Pixel Shaders and how it was fairly easy to write code in the HLSL language for doing effects like blurs, compile this code and then get it installed into the rendering pipeline of Silverlight. Dynamic Deep Zoom was covered next, followed by some graphical effects for page turning.

#### Web Forms vs ASP.NET MVC by Dino Esposito

An introduction to the ASP.NET MVC framework with a discussion of the comparisons against the standard WebForms platform. The summary was basically that WebForms was really good at the time it was invented as it wrapped up a number of techniques that were being applied by hand in many ASP applications, but for new development, the MVC approach is far better.

#### Dependency Properties: the critical technology by Dave Wheeler

This talk gave a brief introduction to Dependency Properties and then covered a number of common mistakes that happen when people use them. These issues involved validation, coercion, and priority, and looked at how these things interact with layout and animation.

#### Security in .NET 4 by Dominic Baier

This was the most interesting talk of the day, though there wasn't much in terms of new APIs in this area in CLR 4.

The first half presentation covered the major change. Whereas the existing version 2 Code Access Security covers both security policy and enforcement, in CLR 4 the policy is being left up to the hosting application, rather than having it be the responsibility of the CLR. The existing policy API is hard to configure and modifying it is hard as it is tricky to predict the effects of a small policy change. Hosting applications will now be expected to use the sandboxing APIs (which were introduced in CLR 2) to configure a number of assemblies to run in full trust and another set of assemblies that will run with a given permission set active. Of course, there now needs to be a means for allowing calls between untrusted and fully trusted code - by default such calls are disallowed. This is achieved by putting code into three sets - Critical (full trust), Transparent (limited trust) and a middle layer of SafeCritical code. Critical code can only be called via this third layer which acts as the gateway into the full trust area, and it is this gateway code that needs to be audited for security. This [2]post covers more of the details.

The second part of the talk covered WIF, Windows Identity Foundation, the new claims based authorisation system. From the perspective of the CLR, this involves the introduction of two new types `IClaimsPrincipal` and `IClaimsIdentity` which have `IIdentity` and `IPrincipal` as supertypes. Microsoft have recently published a free book on claims based identity and access control [3]here.

All things considered, this was a really interesting day of talks and well worth attending.

1. <http://www.devweek.com/>

2. <http://blogs.msdn.com/shawnfa/archive/2009/11/09/transparency-as-enforcement-in-clr-v4.aspx>

#### 4.3.5 It's hard to interoperate (2010-03-21 20:32)

There's a [1]good post on the problems of meshing the deterministic finalization of the COM world with the heuristic driven garbage collection of the managed world. In particular the trick of using ReleaseComObject to deterministically clean up the COM objects has lead to problems when interfaces that were previously implemented using COM objects are replaced by managed implementations sitting behind a Com callable wrapper. The post links to some interesting posts, in particular [2]this post which talks about the creation of the runtime callable wrappers when an interface enters the managed world of the CLR.

1. <http://blogs.msdn.com/visualstudio/archive/2010/03/01/marshal-releasecomobject-considered-dangerous.aspx>
  2. <http://blogs.msdn.com/mbend/archive/2007/04/18/the-mapping-between-interface-pointers-and-runtime-callable-wrappers-rcws.aspx>
- 

#### 4.3.6 The domain of the thread (2010-03-25 20:58)

I remember that the first time I heard it, it seemed strange that the same thread could have parts of its stack running in different Appdomains. Since it's possible to unload an AppDomain, the CLR needs to have a way to abort those parts of the stack that will return to functions that are in any AppDomain that is going to be unloaded. It does this by raising ThreadAbort to give those stack frames a chance to exit, and then throws AppDomainUnload to the function that calls into the unloaded AppDomain in the first place.

The situation is easy to set up. We create a number of domains and load an object into each of them. We then call a method that transitions into the next AppDomain, and when we have transitioned into them all, we can call the AppDomain.Unload.

```
namespace ConsoleApplication1
{
    public class Client : MarshalByRefObject
    {
        public void TransitionIntoDomain(Queue<AppDomain> domains, AppDomain domainToKill)
        {
            PrintThreadDetails();
            try
            {
                if (domains.Count > 0)
```

```

{
AppDomain nextDomain = domains.Dequeue();
Client client = (Client)nextDomain.CreateInstanceAndUnwrap(Assembly.GetExecutingAssembly().FullName, "ConsoleApplication1.Client");
client.TransitionIntoDomain(domains, domainToKill);
Console.WriteLine("Call returned to domain {0}", AppDomain.CurrentDomain.FriendlyName);
}
else
{
AppDomain.Unload(domainToKill);
}
}
catch (Exception ex)
{
Console.WriteLine("Caught exception {0} in domain {1}", ex.GetType().Name, AppDomain.CurrentDomain.FriendlyName);
}
}

public void PrintThreadDetails()
{
Console.WriteLine("Running in domain {0} on thread {1}", AppDomain.CurrentDomain.FriendlyName, Thread.CurrentThread.ManagedThreadId);
}

class Program
{
static void Main()
{
Client client = new Client();
Queue<AppDomain> domains = new Queue<AppDomain>();
foreach (string name in new string[] { "domain1", "domain2", "domain3", "domain4", "domain5" })
{
domains.Enqueue(AppDomain.CreateDomain(name));
}
client.TransitionIntoDomain(domains, domains.Where(x => x.FriendlyName == "domain3").First());
}
}
}
}

```

We can see the exceptions that get thrown from the output of the program.

```

Running in domain ConsoleApplication3.exe on thread 1
Running in domain domain1 on thread 1
Running in domain domain2 on thread 1
Running in domain domain3 on thread 1
Running in domain domain4 on thread 1
Running in domain domain5 on thread 1
Caught exception ThreadAbortException in domain domain5

```

```
Caught exception ThreadAbortException in domain domain4  
Caught exception ThreadAbortException in domain domain3  
Caught exception AppDomainUnloadedException in domain domain2  
Call returned to domain domain1  
Call returned to domain ConsoleApplication3.exe
```

An interesting follow on experiment is to try to stop the ThreadAbortException by catching it and using Thread.ResetAbort(), and another experiment is to have the same domain multiple times on the queue.

In the first case we add the following code at the start of the catch clauses for the try:

```
catch (ThreadAbortException ex)  
{  
    Console.WriteLine("Abort caught");  
    Thread.ResetAbort();  
}
```

This gives

```
Running in domain ConsoleApplication3.exe on thread 1  
Running in domain domain1 on thread 1  
Running in domain domain2 on thread 1  
Running in domain domain3 on thread 1  
Running in domain domain4 on thread 1  
Running in domain domain5 on thread 1  
Abort caught  
Abort caught  
Abort caught  
Caught exception AppDomainUnloadedException in domain domain2  
Call returned to domain domain1  
Call returned to domain ConsoleApplication3.exe
```

For the case where we enter the same domain multiple times, we get output such as the following:

```
Running in domain ConsoleApplication3.exe on thread 1  
Running in domain domain1 on thread 1  
Running in domain domain2 on thread 1  
Running in domain domain3 on thread 1  
Running in domain domain4 on thread 1  
Running in domain domain5 on thread 1  
Running in domain domain1 on thread 1  
Running in domain domain2 on thread 1  
Running in domain domain3 on thread 1  
Running in domain domain4 on thread 1  
Running in domain domain5 on thread 1  
Caught exception ThreadAbortException in domain domain5  
Caught exception ThreadAbortException in domain domain4  
Caught exception ThreadAbortException in domain domain3  
Caught exception ThreadAbortException in domain domain2  
Caught exception ThreadAbortException in domain domain1  
Caught exception ThreadAbortException in domain domain5  
Caught exception ThreadAbortException in domain domain4
```

Caught exception ThreadAbortException in domain domain3  
Caught exception AppDomainUnloadedException in domain domain2  
Call returned to domain domain1  
Call returned to domain ConsoleApplication3.exe

One final thing to do is to unload the AppDomain that is running the call to Unload. I expect you can guess what happens in that case.

---

## 4.4 April

### 4.4.1 Left, Right, Left, Right... (2010-04-02 18:52)

[1]Pendulum: Leon Foucault and the Triumph of Science by Amir Aczel

This was a really entertaining read, giving lots of information about the state of French science in the 1850s. I hadn't really twigged before reading this, that Foucault's experiment was the first real proof that the Earth rotated about its axis, and hadn't realised that it was Foucault who invented the gyroscope which could be used for further proof of this rotation. Foucault did a number of other interesting experiments in his time, such as measuring the speed of light and showing that it varied depending on the medium through which it travelled. A good read.

1. [http://www.amazon.co.uk/Pendulum-Leon-Foucault-Triumph-Science/dp/0743464796/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1270217893&sr=8-1-spell](http://www.amazon.co.uk/Pendulum-Leon-Foucault-Triumph-Science/dp/0743464796/ref=sr_1_1?ie=UTF8&s=books&qid=1270217893&sr=8-1-spell)
- 

### 4.4.2 Tell me about it (2010-04-02 18:52)

The Software Engineering Radio podcast has a great [1]interview with Rich Hickey, the creator of Clojure, who explains in some detail the reasons behind some of the design of this interesting language. Well worth a listen!

1. <http://www.se-radio.net/podcast/2010-03/episode-158-rich-hickey-clojure>
-

#### 4.4.3 On reflection, it's better to call directly (2010-04-07 18:02)

Recently, some discussion at work, caused me to find [1]the following blog post by Jon Skeet which talks about getting better performance by converting a MethodInfo, obtained via reflection, into a delegate. Playing with this technique, I was surprised to find that you can enter a method with the this pointer set to null.

Take the following test method:

```
class A
{
    public int Method1(int incoming)
    {
        Console.WriteLine("This is {0}", this == null ? "null" : this.ToString());
        return incoming * 2;
    }
}
```

And call into it using the code:

```
MethodInfo method = typeof(A).GetMethod("Method1");
Func<A, int, int> methodPtr =
    (Func<A,int,int>) Delegate.CreateDelegate(typeof(Func<A, int, int>), method);
Console.WriteLine(methodPtr(new A(), 10));
Console.WriteLine(methodPtr(null, 10));
```

We get the following output:

```
This is ConsoleApplication1.A
20
This is null
20
```

Of course, the next obvious question is what happens if you call a virtual method passing in null as the target.

```
class Foo
{
    public virtual int Test()
    {
        Console.WriteLine("A");
        return 1;
    }
}

class Bar : Foo
{
    public override int Test()
    {
        Console.WriteLine("B");
        return 2;
    }
}
```

Calling using the following code:

```
MethodInfo method2 = typeof(Foo).GetMethod("Test");
Func<Foo, int> methodPtr2 =
(Func<Foo, int>)Delegate.CreateDelegate(typeof(Func<Foo, int>), method2);
methodPtr2(new Foo());
methodPtr2(new Bar());
methodPtr2(null);
```

Gives the output:

- A
- B

with a NullReferenceException on the third call.

---

1. [http://msmvps.com/blogs/jon\\_skeet/archive/2008/08/09/making-reflection-fly-and-exploring-delegates.aspx](http://msmvps.com/blogs/jon_skeet/archive/2008/08/09/making-reflection-fly-and-exploring-delegates.aspx)

#### **4.4.4 That's a letter, not a number (2010-04-11 08:20)**

[1]E, The Story of a Number by Eli Maor

This is another maths book aimed at a fairly non-technical level, but which contains a lot of interesting anecdotes and stories about the people involved. It traces the story of e, starting out with Napier's logarithms, passing through the discovery of calculus and onto the relatively recent proofs that pi and e are transcendental. Along the way, there are many interesting side stories, such as the problems of the catenary and the cycloid, the connection between e and pi, Euler's various formulae and an interesting chapter on the mappings of the complex plane. A thoroughly enjoyable read.

---

1. [http://www.amazon.co.uk/22e-22-Number-Princeton-Science-Library/dp/0691141347/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1270932197&sr=8-1](http://www.amazon.co.uk/22e-22-Number-Princeton-Science-Library/dp/0691141347/ref=sr_1_1?ie=UTF8&s=books&qid=1270932197&sr=8-1)

#### **4.4.5 Pack your bags (2010-04-11 08:21)**

It's all very well having your Clojure application compiled into a series of class files, but sometimes you just want something that's easy to deploy. Ideally a single jar file with everything ready to go.

There's a tool called Leiningen which does just this. The tool is good, but I had a couple of problems getting it to run under windows, so I thought I'd document here how I got it working for a simple hello world example.

Leiningen can self-install on other platforms apart from windows. In the Windows case, you need the `lein.bat` script from the Installation section on the [1]github page. You'll need to the [2]1.1.0 version of Leiningen – the script comments leads you to the 1.0.1 version of Leiningen, but this didn't work for me. You'll also need a version of Clojure – I used 1.1.0 which I already had on my machine. You need to point some environment variables at these jar files.

```
set LEIN_JAR=C:/Users/Clive/Desktop/lein/lein-1.1.0-standalone.jar  
set CLOJURE_JAR=C:/Users/Clive/Desktop/lein/clojure.jar
```

I set up a directory structure - I made a `helloworld` directory, and into `src/test/hello.clj` I put the code

```
(ns test.hello  
(:gen-class))  
  
(defn -main [& args]  
(println "Hello world!"))
```

We can now define a project file that specifies our entry point, and the jars that we depend upon.

```
(defproject test.hello "1.0"  
:dependencies [[org.clojure/clojure "1.1.0-master-SNAPSHOT"]]  
:main test.hello)
```

Leiningen offers a number of command line arguments. The first important one is “compile” to build the project.

```
C:/Users/Clive/Desktop/lein/helloworld>..lein.bat compile  
[copy] Copying 1 file to C:/Users/Clive/Desktop/lein/helloworld/lib  
Compiling test.hello
```

We can then run a repl with the compiled code on the class path.

```
C:/Users/Clive/Desktop/lein/helloworld>..lein.bat repl  
Clojure 1.1.0  
user=> (use 'test.hello)  
nil  
user=> (-main)  
Hello world!  
nil
```

But better still, we can build a jar of the project, and we can also build an uberjar that includes the project code and the code for Clojure i.e an easily deployed solution.

```
C:/Users/Clive/Desktop/lein/helloworld>..lein.bat uberjar  
All :namespaces already compiled.  
Created C:/Users/Clive/Desktop/lein/helloworld/test.hello.jar  
Including test.hello.jar  
Including clojure-1.1.0-master-20091231.150150-10.jar
```

```
C:/Users/Clive/Desktop/lein/helloworld>dir  
Directory of C:/Users/Clive/Desktop/lein/helloworld
```

```
11/04/2010 09:09 <DIR> .  
11/04/2010 09:09 <DIR> ..  
11/04/2010 09:07 <DIR> classes
```

```
11/04/2010 09:07 <DIR> lib  
10/04/2010 22:03 143 project.clj  
10/04/2010 21:58 <DIR> src  
11/04/2010 09:09 1,876,304 test.hello-standalone.jar  
11/04/2010 09:09 5,493 test.hello.jar
```

We can easily run this using the command line.

```
C:UsersCliveDesktopleinhelloworld>java -cp test.hello-standalone.jar test.hello  
Hello world!
```

- 
1. <http://github.com/technomancy/lein>
  2. <http://github.com/downloads/technomancy/lein/lein-1.1.0-standalone.jar>

#### **4.4.6 Watch these (2010-04-11 21:23)**

I've recently been watching some online Clojure videos.

Craig Andera has a great sequence of videos on concurrency support in Clojure [1]here. There's a great talk introducing monads [2]here and a great sequence of topic based videos [3]here.

- 
1. <http://link.pluralsight.com/clojure>
  2. <http://www.youtube.com/user/LinkedInTechTalks?feature=mhw5#p/u/0/0bR3qi4Guys>
  3. <http://vimeo.com/channels/fulldisclosure>

---

#### **4.4.7 Pack your bags 2 (2010-04-11 21:24)**

Before posting about Leiningen I ought to have mentioned two other useful things that it does.

First, it supports a new command which sets up a project structure.

```
C:UsersCliveDesktoplein>lein.bat new test2  
Created new project in: test2
```

```
C:UsersCliveDesktoplein>dir  
Volume in drive C is ACER  
Volume Serial Number is 2AA8-1BF3
```

```
Directory of C:UsersCliveDesktoplein>
```

```
11/04/2010 22:10 24 .gitignore  
11/04/2010 22:10 168 project.clj  
11/04/2010 22:10 115 README  
11/04/2010 22:10 <DIR> src  
11/04/2010 22:10 <DIR> test
```

The test directory is set up to contain a sequence of unit tests.

```
C:UsersCliveDesktop\test2>type core _test.clj  
(ns test2.core-test  
(:use [test2.core] :reload-all)  
(:use [clojure.test]))  
(deftest replace-me ;; FIXME: write  
(is false))
```

And the project file is set up to include the contrib project that contains the test framework.

```
C:UsersCliveDesktop\test2>type project.clj  
(defproject test2 "1.0.0-SNAPSHOT"  
:description "FIXME: write"  
:dependencies [[org.clojure/clojure "1.1.0"]  
[org.clojure/clojure-contrib "1.1.0"]])
```

Leiningen will then run the unit tests using the test command.

```
C:UsersCliveDesktop\test2>..lein.bat compile  
Downloading: org/clojure/clojure/1.1.0/clojure-1.1.0.pom from central  
Downloading: org/clojure/clojure-contrib/1.1.0/clojure-contrib-1.1.0.pom from central  
Downloading: org/clojure/clojure-contrib/1.1.0/clojure-contrib-1.1.0.pom from clojure  
Transferring 1K from clojure  
Downloading: org/clojure/clojure/1.1.0/clojure-1.1.0.jar from central  
Downloading: org/clojure/clojure-contrib/1.1.0/clojure-contrib-1.1.0.jar from central  
Downloading: org/clojure/clojure-contrib/1.1.0/clojure-contrib-1.1.0.jar from clojure  
Transferring 3215K from clojure  
[copy] Copying 2 files to C:UsersCliveDesktop\test2\lib  
All :namespaces already compiled.
```

```
C:UsersCliveDesktop\test2>..lein.bat test
```

Testing test2.core-test

```
FAIL in (replace-me) (core _test.clj:6)  
expected: false  
actual: false
```

Ran 1 tests containing 1 assertions.

1 failures, 0 errors.

---

Total:

```
Ran 1 tests containing 1 assertions.  
1 failures, 0 errors.
```

These functions look really useful.

---

#### **4.4.8 Don't get into a fight with him (2010-04-25 14:23)**

[1]Duelling Idiots and Other Probability Puzzlers by Paul J. Nahin

An interesting collection of 21 probability questions with solutions and computer simulation code written for MathLab. An enjoyable read with the author emphasising that not all problems have a nice closed form solution and sometimes a computer simulation is the only way to get a result.

1. [http://www.amazon.co.uk/Duelling-Idiots-Other-Probability-Puzzlers/dp/0691102864/ref=sr\\_1\\_fkmr0\\_1?ie=UTF8&qid=1272190032&sr=1-1-fkmr0](http://www.amazon.co.uk/Duelling-Idiots-Other-Probability-Puzzlers/dp/0691102864/ref=sr_1_fkmr0_1?ie=UTF8&qid=1272190032&sr=1-1-fkmr0)
- 

#### **4.4.9 Down, down, deeper and down. (2010-04-25 14:23)**

There are some interesting blog posts on the [1]blog of Cliff Click of Azul Systems. This company is producing a hardware assisted version of the JVM There is a good summary of their recent work [2]here, including some more information on their use of inline [3]call site optimisation, [4]calling conventions of the JIT and the [5]internals of the locking mechanism.

1. <http://www.azulsystems.com/blogs>
  2. <http://www.azulsystems.com/blog/cliff-click/2009-12-22-touching-base>
  3. <http://www.azulsystems.com/blog/cliff-click/2010-04-08-inline-caches-and-call-site-optimization>
  4. <http://www.azulsystems.com/blog/cliff-click/2010-04-08-inline-caches-and-call-site-optimization>
  5. <http://www.azulsystems.com/blog/cliff-click/2010-01-09-biased-locking>
- 

#### **4.4.10 Procrastination is (in fact) the thief of space (2010-04-25 14:23)**

In conversation the other day someone mentioned that in Haskell, a pure functional language, it is possible to write a doubly linked list. I remember finding it hard to believe the first time I heard this, so I guess the best way to prove that it is possible is to write the code.

```
data DoubleLinkedList a = Node a (DoubleLinkedList a) (DoubleLinkedList a) | Nil
```

```

mkList' :: [a] -> DoubleLinkedList a -> DoubleLinkedList a
mkList' [] prev = Nil

mkList' (x:xs) prev =
let cur = Node x prev (mkList' xs cur)
in cur

mkList :: [a] -> DoubleLinkedList a
mkList xs = mkList' xs Nil

```

To do some quick testing we'll also need some accessors.

```

getNext (Node current prev next) = next
getPrevious (Node current prev next) = prev
getCurrent (Node current prev next) = current

```

If we load the above code into GHC, we get the following results:

```

*Main> getCurrent (mkList [1,2,3])
1
Main> getCurrent (getNext (mkList [1,2,3]))
2
Main> getCurrent (getPrevious (getNext (mkList [1,2,3])))
1
Main> getCurrent (getPrevious (getPrevious (getNext (getNext (mkList [1,2,3])))))
1

```

The trick is of course that Haskell supports recursive bindings in let expressions, allowing us to tie the knot. In mkList', the cur binding extends into the value that it is bound to, and, of course, this works because in the implementation of Haskell we effectively have delay cells wrapped around the thunks.

Eager languages make this self referencing binding harder to express, though F# offers a limited form of recursive binding:

```

> let rec x = seq { yield 1; yield! x };;
let rec x = seq { yield 1; yield! x };;
----- ^ ^

```

stdin(13,35): warning FS0040: This and other recursive references to the object(s) being defined will be checked for initialization-soundness at runtime through the use of a delayed reference. This is because you are defining one or more recursive objects, rather than recursive functions. This warning may be suppressed by using #nowarn "40" or -nowarn 40

```

val x : seq<int>
> Seq.take 4 x;;
val it : seq<int> = seq [1; 1; 1; 1]

```

Other eager languages such as Clojure don't allow us to express this "knot tying" so easily. We can write fib using lazy sequences, and using an atom to tie the knot, using code like the following:

```

(def fib
(letfn [(stepfn [items]
(lazy-seq
(cons (+ (first items) (second items))
(stepfn (rest items)))))])

```

```
(let [holder (atom nil)
this
(lazy-seq
(cons 1
(cons 1
(stepfn (deref holder))))]
(swap! holder (fn [old] this))
this))

user> (take 10 fib)
(1 1 2 3 5 8 13 21 34 55)
```

However, we have lost the concise expression. However, using macros we can gain a little of this back. We'll need the symbol-macrolet macro which is available in the macro-utils contributed code.

```
user> (use 'clojure.contrib.macro-utils)

(defmacro with-knotted-sequence [binding & body]
(let [[var value] binding]
'(let [holder # (atom nil)]
(let [ var
(lazy-seq
(symbol-macrolet [ var (deref holder #)] value))]
(swap! holder # (fn [ignore #] var))
(do [1] @body)))))

(def fib2
(letfn [(stepfn [items]
(lazy-seq
(cons (+ (first items) (second items))
(stepfn (rest items))))]
(with-knotted-sequence
[this (cons 1 (cons 1 (stepfn this)))]
this)))

user> (take 10 fib2)
(1 1 2 3 5 8 13 21 34 55)
```

We can write the double linked list example in Clojure, but we need to use a delay node to allow us to tie the knot. In the following code, the next item will be a delay that needs to be forced to get the node value.

```
(defn double-link
([x] (double-link x nil))
([x prev]
(when x
(let [holder (atom nil)
following
(delay
(let [this (deref holder)]
(double-link (rest x) this))]
this {:current (first x) :previous prev :next following }])
(swap! holder (fn [ _] this))
this))))
```

Again, we could write some macros to tidy this all up. There is one problem with this implementation.

```
user> (:current a)
1
user> (:current (:next a))
nil
```

We need to use force in this case to get the value from the delay. Using sequence accessors in fib, the laziness can be hidden, with the lazy sequence being forced when required. In this double linked list we need to do the forcing, though we'd hide it inside an accessor function in a real implementation.

```
user> (:current (force (:next a)))
2
user> (:current (:previous (force (:next a))))
1
user> (:current (:previous (:previous (force (:next (force (:next a)))))))
1
```

The Clojure implementation makes it clearer where the suspensions are present, but this makes the code harder to read than the Haskell. However, it does make it easier to reason about the amount of space that is going to be taken up.

1. mailto:~@body

---

#### 4.4.11 What material is that made from? (2010-04-28 08:46)

[1]Mark and I have been preparing some material on Clojure, ready for a presentation in May. It's in a very early state, but I've got together the material I prepared for our first run through which happened last night and have put it on line [2]here. The various clj files contain a mix of text and Clojure which I evaluated inline using the emacs bindings set up by [3]Clojure Box.

1. <http://blog.software-acumen.com/>
  2. <http://cid-3f21df299c355e7f.skydrive.live.com/self.aspx/Public/closure-materials1.zip>
  3. <http://clojure.bighugh.com/>
-

## 4.5 May

### 4.5.1 Is that structurally sound? (2010-05-20 18:52)

I don't know. You spent a lot of time mastering the oddities of the struct types in the C # world, carefully preparing yourself for the subtle copying that happens when you call through an interface, and then a blog post turns up that shows that the behaviour when you call a method on a struct depends on whether it is held in a readonly field or not.

```
interface IInc
{
    void Increment();
}

struct TestStruct : IInc
{
    public int count;
    public void Increment()
    {
        count++;
    }
}
```

Defining a field of the appropriate type, we see 0,1,2 is printed by the following

```
TestStruct struct1;

((IInc)struct1).Increment();
Console.WriteLine(struct1.count);
struct1.Increment();
Console.WriteLine(struct1.count);
struct1.Increment();
Console.WriteLine(struct1.count);
```

The first value printed is zero because the call via the cast boxes the structure and hence copies it before the call mutates it.

The shocking behaviour is what happens if you change the field declaration to  
readonly TestStruct struct1;

when the output changes to 0,0,0.

Simon's [1]blog post explains it and also links to [2]Eric Lippert's explanation.

---

1. <http://www.simple-talk.com/community/blogs/simonc/archive/2010/05/19/91401.aspx>

2. <http://blogs.msdn.com/ericlippert/archive/2008/05/14/mutating-readonly structs.aspx>

## 4.5.2 Blimey (2010-05-25 17:47)

[1]The Penguin Dictionary of Curious and Interesting Numbers by David Wells

Funnily enough this was a really interesting read, despite the fact that it is a little more than a list of numbers together with a small amount of text describing the interesting properties of the number. Lots of interesting titbits, ranging from the Taxi Cab number 1729 to Graham's number which is the largest number listed in the Guinness Book of Records.

There were a number of entries describing divisibility tricks for working out if a number is divisible by a prime without having to do the long division. It took me a while to spot the unifying algorithm for some of these.

Say we want to test divisibility by 17. We know that  $3 * 17 = 51$  which is close to  $5 * 10 = 50$ . If we want to determine if the number  $10a+b$  is divisible by 17, we can look reduce this to looking at the divisibility of  $a-5b$  because we know that  $10a+b$  being divisible by 17, means that  $5(10a+b)$  is divisible by 17, and this latter form reduces to  $-a+5b$ . We can look at  $a-5b$  instead to keep things positive. Using this trick we can reduce large numbers fairly quickly to smaller numbers and then repeat this trick until the number is small enough to check mentally.

It is easy to write some Clojure code to check this:

```
(defn find-multiplier [modulus]
  (loop [multiplier 1]
    (let [multiplier-modulo (mod (* multiplier 10) modulus)]
      (if (= multiplier-modulo 1)
          multiplier
          (if (= multiplier-modulo (- modulus 1))
              (- multiplier)
              (recur (inc multiplier)))))))

(defn test [number by]
  (let [multiplier (find-multiplier by)]
    (loop [current number]
      (if (<= current by)
          (or (= current 0) (= current by))
          (let [last-digit (mod current 10)
                other-digits (quot current 10)
                next (+ other-digits (* multiplier last-digit))]
            next-loop-value (if (>= next 0) next (- next)))
          (if (>= next-loop-value current)
              (if (< current 100)
                  (= (mod current by) 0)
                  (throw (Exception. "failure")))
              (recur next-loop-value))))))

(defn check-algorithm [by]
  (dotimes [i 1000000]
    (when (not (= (= (mod i by) 0) (test i by)))
      (println "Failure at " i))))
```

1. [http://www.amazon.co.uk/Penguin-Dictionary-Curious-Interesting-Numbers/dp/0140261494/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1274774718&sr=8-1](http://www.amazon.co.uk/Penguin-Dictionary-Curious-Interesting-Numbers/dp/0140261494/ref=sr_1_1?ie=UTF8&s=books&qid=1274774718&sr=8-1)
- 

#### 4.5.3 One thing at a time please! (2010-05-25 17:48)

[1]The Art of Concurrency - A Thread Monkey's Guide To Writing Parallel Applications by Clay Breshears

This was an fairly basic introduction to the theory of writing parallel programs. It started with chapters covering the strategies for writing parallel programs, and how to check the correctness and measure the performance of a parallel solution. Breshears gives eight simple rules for designing parallel applications and later gives four criteria for evaluating the design of a parallel algorithm – efficiency, simplicity, portability, scalability. The book then moves on to looking at parallel implementations of a number of algorithms – parallel sum and prefix scan, map reduce, sorting, searching and graph algorithms. Generally the author starts with a sequential algorithm, modifies it a little and then shows how a number of standard techniques can be used for covering it into a parallel algorithm.

These algorithms are implemented on top of a number of different libraries – Open MP, TBB, windows threads and pthreads. None of these libraries were covered in a great deal of detail, but the examples gave a nice flavour of what the libraries offered.

Looking at the various correctness criteria for the algorithms, got me thinking again about the .NET memory model and the needs for various types of memory barriers. While surfing the internet, I came across the [2]following paper which is the best I have read on the subject of the interaction between the software and hardware, and why memory barriers are needed. The author has a great example where a thread moves between CPUs, demonstrating the need for the operating system to put in memory barriers during context switches. The .NET 2 memory model only guarantees that stores will not be reordered with regard to other stores, which corresponds to a guarantee the x86 architecture makes. However, barriers also effect the compiler optimisations that may be applied, preventing the compiler, for example, moving loads and stores across barriers.

This book recommends some other more advanced books that cover the analysis and design of a number of other algorithms. I will be following up some of these recommendations in the near future,

1. [http://www.amazon.co.uk/Art-Concurrency-Monkeys-Parallel-Applications/dp/0596521537/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1274775902&sr=1-1](http://www.amazon.co.uk/Art-Concurrency-Monkeys-Parallel-Applications/dp/0596521537/ref=sr_1_1?ie=UTF8&s=books&qid=1274775902&sr=1-1)
  2. <http://www.rdrop.com/users/paulmck/scalability/paper/whymb.2009.04.05a.pdf>
-

#### 4.5.4 That's a bit complex (2010-05-25 17:48)

I've been doing some interviewing at work, and during these sessions, the notion of the complexity of an algorithm always comes up. It's always been a bit of an annoyance to me, that when people state things about complexity, it isn't always clear what that complexity is measuring. For example, sorting is  $O(n \log n)$  in the number of comparisons, though for some kinds of data it is possible to sort more efficiently by not actually using comparisons at all.

The [1]Using Your Head site had a good series of puzzles based on this theme, with a number of really clever algorithms for doing a sort of  $n$  integers in  $O(n)$ , in a complexity model where the integer values are unbounded and all of the operations such as adding and multiplying are assumed to be  $O(1)$  in cost. Here we'll re-implement one of the given solutions in Clojure.

We'll assume all of the integers are positive – if they aren't then we can adjust the array by mapping the smallest value to 0 by adding a constant to all elements. We'll then represent the array using blocks of  $N$  bits for an appropriate  $N$ . To allow for overflow, we'll have a bit between the elements. This will act as a sign bit when we do a subtraction in the `ge` function which simultaneously compares two arrays element by element by doing a subtraction.

Given an input sequence, we'll encode it in the representation described above. We'll return a map containing the encoding together with a lot of additional information we'll need from time to time.

```
(defn generate-vectors[input]
  (let [size (count input)
        width (+ (max size (reduce max 0 input)) 1)]
    (let [flagmask (bit-shift-left 1 (- width 1))
          datamask (- flagmask 1)
          items-and-indices (zipmap (iterate inc 0) input)]
      {
        :width width
        :size size
        :flagmask flagmask
        :datamask (- flagmask 1)
        :encoded-input
        (reduce (fn [sofar [index item]]
                  (+ sofar (bit-shift-left item (* index width))))
               0
               items-and-indices)
        :ones-vector
        (reduce (fn [sofar [index item]]
                  (+ sofar (bit-shift-left 1 (* index width))))
               0
               items-and-indices)
        :flagmask-vector
        (reduce (fn [sofar [index item]]
                  (+ sofar (bit-shift-left flagmask (* index width))))
               0
               items-and-indices)
        :datamask-vector
        (reduce (fn [sofar [index item]]
                  (+ sofar (bit-shift-left datamask (* index width))))
               0
               items-and-indices)}))
```

```
0
items-and-indices)
})))
```

For debugging purposes, we'll define a function that prints the contents of the encoded vector.

```
(defn extract-index [vector position setup-data]
(bit-and (bit-shift-right vector (* position (:width setup-data))) (:datamask setup-data)))

(defn print-vector [vector setup-data]
(dotimes [ i (:size setup-data)]
(print (extract-index vector i setup-data) " "))
(println))
```

We'll quickly test the code on a small example:

```
user> (def vector1-setup (generate-vectors [1 3 2]))
#'user/vector1-setup
user> (print-vector (:encoded-input vector1-setup) vector1-setup)
1 3 2
nil
```

We can now define the function which compares two vectors. We subtract the two integer representations, and then use the sign bits to see which subtractions gave a negative result.

```
(defn ge [a b setup-data]
(bit-shift-right
(bit-and (+ a (bit-and (bit-not b) (:datamask-vector setup-data)) (:ones-vector setup-data))
(:flagmask-vector setup-data))
(- (:width setup-data) 1)))

user> (print-vector (:encoded-input vector2-setup) vector2-setup)
2 2 2
nil
user> (print-vector (:encoded-input vector3-setup) vector3-setup)
2 1 3
nil
user> (print-vector (ge (:encoded-input vector2-setup) (:encoded-input vector3-setup)
vector2-setup)
vector2-setup)
1 1 0
nil
```

We can now put this into a function which compares a start vector will all cyclic shifts of the same vector, adding the result vectors as we go along.

```
(defn compare-counts [vector-plus-data]
(let [width (:width vector-plus-data)
size (:size vector-plus-data)
datamask (:datamask vector-plus-data)
vector (:encoded-input vector-plus-data)]
(loop [data vector less-than-counts 0 shifts size]
(if (= shifts 0)
less-than-counts
(let [shifted-data (+ (bit-shift-right data width)
(bit-shift-left (bit-and data datamask) (* width (- size 1))))]
(inc less-than-counts))))
```

```
(recur shifted-data
(+ less-than-counts (ge vector shifted-data vector-plus-data))
(dec shifts)))))
```

We can test this:

```
user> (def vector4-setup (generate-vectors [17 18 12 11 13 15 16]))
#'user/vector4-setup
user> (print-vector (compare-counts vector4-setup) vector4-setup)
6 7 2 1 3 4 5
nil
```

We now have the indices. We then simply need to unpack the array, taking care to handle the case where there are duplicate values.

There is a follow-up puzzle on the mentioned site which shows how various other functions can be implemented efficiently in this complexity model.

---

1. <http://brand.site.co.il/riddles/200812q.html>

#### 4.5.5 You spin me around and around (2010-05-25 17:53)

F# supports the idea of a recursive value binding, allowing one to define various recursive data structures.

```
> type Foo = { next : Foo };;
> let rec x = { next = x };;
val x : Foo
```

Of course, we then have problems using the cyclic structure that we've just constructed. For example, the following expression loops:

```
> x = x.next;;
```

There's a [1]good paper explaining why this facility is useful.

I came across this while looking at an [2]interesting paper that discusses the duality between the notion of garbage for events and for standard objects in a managed runtime. The paper uses this to design a modified garbage collection algorithm, which is then used to drive the design of a set of combinators and primitives for a reactive library which doesn't leak garbage.

---

1. <http://research.microsoft.com/pubs/79951/valrec-final-ml-workshop.pdf>
2. <http://tomasp.net/academic/event-chains/event-chains.pdf>

#### **4.5.6 Put Something In The Way (2010-05-28 19:41)**

There's a great article on memory barriers [1]here. The article is aimed at the JVM and discusses various instruction sequences that HotSpot generates to enforce the memory barriers in the program.

1. [http://www.infoq.com/articles/memory\\_barriers\\_jvm\\_concurrency](http://www.infoq.com/articles/memory_barriers_jvm_concurrency)
- 

#### **4.5.7 That's some beautiful architecture (2010-05-29 08:58)**

[1]Pro Javascript Design Patterns by Ross Harmes and Dustin Diaz

It's been a while since I've done much Javascript development, and it was great to get back to some Javascript coding. This book was a joy to read. The large collection of fairly extensive and realistic examples that it contains to illustrate the various design patterns, served to reinforce the idea that it is possible to write large applications in this interesting language. The book makes it clear that Javascript is such a flexible language, with many possible ways of doing things, that programmers need discipline to keep the resulting applications maintainable and extensible.

For me, it was also interesting to get back to using [2]FireBug which makes developing Javascript a dream. The interactive console makes it easy to try things out, and the great tools for looking at the html and the css makes it really easy to get a page looking good.

Of course, I fell into the usual Javascript traps.

Forgetting to use new with a function that is intended as a constructor, causing the global environment to get the properties that were intended for the object

```
>>> function Foo() { this.foo = 20; }
>>> var a = Foo()
>>> a
>>> window.foo
20
>>> var a = new Foo()
>>> a
Object foo=20
```

Forgetting to bind the current this to another variable (that) for use inside a closure.

```
function Button(text)
{
  var button = document.createElement("button");
  window.document.body.appendChild(button);
  button.title = text;
  this.pressCount = 0;
  var that = this;
```

```

button.addEventListener(
  "click",
  function() { that.pressCount++; },
  false);
}

var button1 = new Button("Press this");

>>> button1
Object pressCount=4

```

The book itself is very good. It starts off with a discussion of the patterns of implementing inheritance and interfaces in Javascript. The inheritance discussion covers the prototypical inheritance of Javascript and then discusses ways to make this emulate the more classic style of inheritance found in other languages. The authors suggest having functions for defining an interface (as a group of properties that must exist on an object), and other functions to check that these properties exist. The text also covers encapsulation – the means for doing this in Javascript is the closure which can be used to encapsulate data inside objects, though unfortunately, using a closure to do this means that there is no way to simulate the protected access of classical inheritance.

The first pattern to be covered is the classic singleton. This is used in Javascript to emulate namespaces, and as a means of doing branching – sniffing the browser and offering functions customised for the current browser's abilities. The example they use to illustrate this, is the different types of methods for doing an XMLHttpRequest which depend on the browser type. This is followed by a chapter on chaining. In this technique methods generally return the object to which they were applied, making it really easy to call multiple methods on the same object using the object.FirstOperation().SecondOperation() syntax.

The book goes on to cover Factory, Bridge, Composite, Facade, Adapter, Decorator, Flyweight, Proxy, Observer, Command and Chain of Responsibility. In each case, the pattern is discussed and then an example or two illustrate the pattern. Typically, this example may consist of several pages of code, implementing some useful functionality.

One thing I noticed. Several of the examples made an XMLHttpRequest, got the result text which was in the form of java object notation, and then used

```
var obj = eval("(" + responseText + ")")
```

to get the result converted into a Javascript object. Using eval feels really wrong to me – I assume people typically use a real JSON parser to do this conversion to prevent a response containing malicious text doing all sorts of interesting function calls.

All things considered, a really good read.

1. [http://www.amazon.co.uk/JavaScript-Design-Patterns-Recipes-Problem-Solution/dp/159059908X/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1274988701&sr=8-1](http://www.amazon.co.uk/JavaScript-Design-Patterns-Recipes-Problem-Solution/dp/159059908X/ref=sr_1_1?ie=UTF8&s=books&qid=1274988701&sr=8-1)
2. <http://getfirebug.com/>

## 4.5.8 Knock me down and I'll keep coming back (2010-05-29 21:09)

I was going to sit down this weekend and try to write Pong using the [1]Reactive Extensions for Javascript. Pong seems to me to be the Reactive version of hello world, and is a good way to get into using some of the combinators. Unfortunately, I found that the Matt Podwysocki's excellent blog already contains his modifications of a Pong game to use the Reactive Extensions. The excellent post can be found [2]here, together with a number of related posts.

1. <http://msdn.microsoft.com/en-us/devlabs/ee794896.aspx>

2. <http://codebetter.com/blogs/matthew.podwysocki/archive/2010/05/04/introduction-to-the-reactive-extensions-for-javascript-refactoring-a-game.aspx>

---

## 4.6 June

### 4.6.1 I'm not bothered! (2010-06-11 19:25)

[1]Nonplussed! Mathematical Proof of Implausible Ideas by Julian Havil

This book covers a number of interesting mathematical puzzles, giving comprehensive solutions and great discussions. The puzzles include Conway's Chequerboard Army, the UpHill Roller and several pursuit puzzles. A very interesting read with great explanations.

My favourite though had to be the coverage of Conway's Fractran. Fractran is a programming language that runs on a very simple virtual machine. This machine takes an input number and a finite sequence of fractions. The machine takes the first number from the sequence and multiplies it against the input number. If the value isn't an integer then the next fraction is taken from the sequence and is multiplied against the input number, and this continues until the result is an integer. If no integer result can be found, the machine terminates with the final result being the input number. If an integer result is found, then the process is repeated, using the result as the input and restarting at the first element of the integer sequence.

The chapter starts out by giving a sequence of integers and then describes how this implements an algorithm to generate the nth prime number.

To see how this works, let's take a small register machine for adding two numbers and show how to translate it into the FRACTRAN language.

Let's take the initial inputs in the registers r2 and r3. Using the states s5,s7,s11,... we can encode a machine which produces the sum of the numbers in r2 by starting in state s5 and doing.

s5: if r3 > 0 then r3 = r3 - 1; goto s7

s7: r2 = r2 + 1; goto s5

The trick is to label the registers by using prime numbers and record the state of the machine as the product of state \* 2 ^ r2 \* 3 ^ r3.

The integer representing the state 5 is  $7 / (5 * 3)$  – we divide by 5 to check we are in state 5, and multiply by 7 to transition to state 7. We divide by 3 to subtract one from the r3 register.

The integer representing the state 7 is  $5 * 2 / 7$  – we divide by 7 to check we are in state 7, multiply by 5 to move to state 5, and multiply by 2 to add to r2.

If neither of these rules matches, then we must have finished in state 5. We therefore need the final rule 1/5 to get rid of the state marker leaving just the register contents.

Hence our program is { 7/15, 10/7, 1/5 }

Let's run this on initial state r2=3 r3=4, which gives a start value of  $5 * 2 ^ 3 * 3 ^ 4 = 3240$ . We multiply 3240 by the various integers giving (1512 32400/7 648) and take the first one which is integer, 1512.

We get the following transitions: 2160, 1008, 1440, 672, 960, 448, 640, 128.

The final result is  $128 = 2 ^ 7$ , so the machine has added 3 and 4 to get the result 7.

---

1. [http://www.amazon.co.uk/Nonplussed-Mathematical-Proof-Implausible-Ideas/dp/0691120560/ref=sr\\_1\\_1?ie=UTF8&qid=1276275534&sr=1-1-spell](http://www.amazon.co.uk/Nonplussed-Mathematical-Proof-Implausible-Ideas/dp/0691120560/ref=sr_1_1?ie=UTF8&qid=1276275534&sr=1-1-spell)

#### 4.6.2 That's very general (2010-06-18 18:26)

[1]Java Generics and Collections by M Naftalin and P Wadler

This is very much a book of two halves.

The first part of the book is a tutorial on Java Generics, and includes coverage of wild cards, subtyping, bounds, reification and reflection. Coming from the CLR world, the implementation of Java Generics by using erasure sounds very strange, and the book does a good job of explaining the benefits of this implementation technique. It discusses how erasure allows generic and non-generic code and libraries to interact, with the system inserting casts at the appropriate points with the knowledge that they cannot fail at runtime. Unfortunately there are edge cases, where the casts can fail and the book goes into some discussion about why this is. In these cases the compiler will emit a warning, though this is at compile time, so it isn't always useful if you are using a library with such problems, as often you are provided with the compiled bytecode and never see these warnings, but the inserted casts can fail in interesting ways at runtime making it hard to deduce the actual root cause.

There's loads of good material and comments in the first half, not all of it related to generics. For example, the book points out that implementing the compare method on integer values as

```
int CompareTo(int x, int y) { return x - y; }
```

is wrong, because in unchecked arithmetic negating int.MINVALUE equals int.MAXVALUE.

The second part of the book covers the Java Collections. Generics have made it easy to define a rich set of collection types and the book recommends using them instead of the more traditional array types. There's a good discussion of whether arrays should be considered a deprecated

type because of the way that they interact with generics and in particular because they should only be used for reifiable types.

The Java Collection library is quite impressive, containing a large number of datatypes, with some being concurrently accessible. The types include HashSets and LinkedHashSets, Sorted Sets, NavigableSets, TreeSets, Queues including PriorityQueues and ConcurrentLinkedQueues, Lists, Maps including HashMaps, and WeakHashMaps, SortedMaps and NavigableMaps.

1. [http://www.amazon.co.uk/Java-Generics-Collections-M-Naftalin/dp/0596527756/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1276277621&sr=1-1](http://www.amazon.co.uk/Java-Generics-Collections-M-Naftalin/dp/0596527756/ref=sr_1_1?ie=UTF8&s=books&qid=1276277621&sr=1-1)
- 

#### **4.6.3 You're talking about my generation (2010-06-18 18:47)**

I got to spent today at [1]Code Generation 2010 where I attended a number of really interesting talks.

The [2]first talk covered the [3]Spoofax language workbench. The speaker covered how languages are defined using a set of context free rules together with some disallow rules. The system could provide multiple parses of ambiguous grammars. Most facilities like syntax highlighting and type checking were implemented using term rewriting, with the ability to scope the available rewrite rules and define new ones at various points to implement type checking and error marking.

The [4]second talk covered the author's IDE for defining and applying code transformations.

The best talk, by far, was the [5]third talk which demonstrated how the [6]JetBrains MPS could be used for generating a C program which controlled a [7]Lego Mindstorms robot. This program controlled the robot as it followed a line on the ground. The speakers then added a sonar device to the robot, and showed how easy it was to change the features that conditionally added functionality to the model – the code could then be regenerated and downloaded onto the robot, which caused it to emergency stop if the sonar detected something solid in its path. The MPS was initially configured for a language syntax that looked a lot like normal C code, with a fairly straightforward generator. They then demonstrated how easy it was to mix a DSL into the model by taking a debugging assert statement and adding into the language and the generator. They emphasised that the MPS holds the tree representation of the data, and you are really editing a projection of this. In particular, they showed an embedded DSL for representing a state machine when it was rendered as text and then as a table, and it could be edited when displayed in both forms. This was a very practical and engaging demo of the use of modelling and code generation.

The [8]last talk was a panel discussion on Model-Driven Software Development Myths. This was an interesting discussion that raised some good points.

A very interesting conference – thanks to [9]Mark for organising it for the fourth consecutive year.

1. <http://www.codegeneration.net/cg2010/programme.php#day3>
  2. <http://www.codegeneration.net/cg2010/keynotes.php#visser>
  3. <http://strategoxt.org/Spoofax>
  4. <http://www.codegeneration.net/cg2010/sessioninfo.php?session=15>
  5. <http://www.codegeneration.net/cg2010/sessioninfo.php?session=9>
  6. <http://www.jetbrains.com/mps/index.html>
  7. <http://mindstorms.lego.com/en-us/default.aspx>
  8. <http://www.codegeneration.net/cg2010/sessioninfo.php?session=106>
  9. <http://blog.software-acumen.com/>
- 

#### 4.6.4 One step at a time (2010-06-23 20:51)

There's a very interesting timeline of monad tutorials [1]here. Of all the tutorials, I thought the [2]original monads tutorial by Phil Wadler was great, particularly when read together with Noel Winstanley's "[3]What the hell are Monads?" and there's also a good introduction to monad transformers in [4]Monad transformers step by step by Martin Grabmuller.

1. [http://www.haskell.org/haskellwiki/Monad\\_tutorials\\_timeline](http://www.haskell.org/haskellwiki/Monad_tutorials_timeline)
  2. <http://homepages.inf.ed.ac.uk/wadler/papers/marktoberdorf/baastad.pdf>
  3. <http://www-users.mat.uni.torun.pl/~fly/materialy/fp/haskell-doc/Monads.html>
  4. <http://www.grabmueller.de/martin/www/pub/Transformers.pdf>
- 

### 4.7 July

#### 4.7.1 Er, mind your language (2010-07-03 07:47)

Erlang Programming by Francesco Cesarini and Simon Thompson

Erlang has always sounded like an interesting language to learn. A functional language with an emphasis on concurrent processes and fault tolerance, which has been used in various bits of telecoms software for many years. This book proved to be a really good introduction. It covers both the language and the associated runtime system, showing the implementation of a number of reasonably sized examples.

I thought it would be good to write up a quick summary of Erlang. We'll show what happens when we type things at the read-eval-print-loop (REPL). The most basic data types are atoms (items starting with a non-capital letter), strings, integers and floating point values. Tuples are represented by enclosing comma separated items in { }.

```
15> {hello, "hello", 1, 2.3 }.  
{hello,"hello",1,2.3 }
```

Items starting with a capital letter are variables. These are assigned using pattern matching.

```
22> {Var1, Var2, 1, 2.3 } = {hello, "hello", 1, 2.3 }.
```

```
{hello,"hello",1,2.3 }
```

```
23> Var1.
```

```
hello
```

```
24> Var2.
```

```
"hello"
```

Once a variable is bound, its value is used in the pattern matching.

```
25> Var1 = "bong".
```

```
* exception error: no match of right hand side value "bong"
```

```
28> {Var1, Var2 } = {"a", "b" }.
```

```
* exception error: no match of right hand side value {"a","b" }
```

Code is packaged as a set of match expressions and results. We could write a source file, fact.erl, containing the following module definition.

```
-module(fact).  
-export([fact/1]).  
fact(0) -> 1;  
fact(N) -> N * fact(N-1).
```

We can then go to the relevant directory and compile and load the module into the running Erlang session.

```
29> cd("c:/users/clive/desktop/erlang").
```

```
c:/users/clive/desktop/erlang
```

```
ok
```

```
30> c("fact").
```

```
{ok,fact }
```

```
31> fact:fact(10).
```

```
3628800
```

```
32> fact:fact(20).
```

```
2432902008176640000
```

There's also a list datatype, which can be pattern matched using a Prolog type syntax.

```
34> [P|Q] = [1,2,3].
```

```
[1,2,3]
```

```
36> P.
```

```
1
```

```
37> Q.
```

```
[2,3]
```

Of course, the really impressive thing about Erlang is that the process is very much a first class concept.

```
-module(fact).  
-export([fact/1, factorialServer/0]).  
  
fact(0) -> 1;  
fact(N) -> N * fact(N-1).
```

```

factorialServer() ->
receive
{X, Pid} -> Pid ! {ok, fact(X)}, factorialServer();
stop -> { }
end.

```

The key idea is that every Erlang process has a mailbox into which messages are delivered. Messages are sent using the ! method which takes a process id and sends it the message which is often a tuple. This tuple will typically contain the pid of the caller, allowing a response to be delivered. The server is typically implemented as a self tail-recursive function which uses receive with a set of match clauses. Receive causes the system to scan the mailbox messages to find the first that matches, at which point the message is removed from the mailbox and the corresponding result expression is evaluated. This method of processing means that the messages are processed one at a time, and hence there is no user level locking to guard against concurrent execution.

We can compile this in the shell, and then spawn a new process that runs the factorialServer function. flash() is a function we can call in the shell to print out all of the messages in the shell process mailbox.

```

45> c("fact").
{ok,fact}
46> Pid = spawn(fact, factorialServer,[]).
<0.135.0>
47> Pid.
<0.135.0>
55> Pid ! {20, self()} .
{20,<0.149.0>}
56> flush().
Shell got {ok,2432902008176640000 }
ok

```

The language contains lots of support for functional programming, including anonymous functions.

```

4> (fun (2) -> 1; (Bar) -> 2 end)(2).
1
5> (fun (2) -> 1; (Bar) -> 2 end)(3).
2

```

There's tons more to the language, but I'll recommend you go and read the book. There are various standard libraries for collections and list manipulation. There are ways for processes to monitor other processes, so they can deal with failure. There are ways to dynamically reload code modules. There are ways to pass binary data around, and easy ways to get processes communicating using sockets. There are various debugging tools and libraries for producing GUIs.

Erlang seems to me to be an undiscovered gem, and this book covered tons of very interesting material. A great read.

#### 4.7.2 Calm - don't react! (2010-07-12 22:05)

I've found the idea of lazy event streams fascinating ever since I came across the idea in the mid-80s when I started playing in [1]Lazy Lispkit Lisp. There was some work at the Programming Research Group in Oxford to implement an operating system using a functional language, with incoming events represented as lazy lists (sequences). The [2]Reactive Extensions, which Microsoft has been releasing as a DevLabs project, share a lot of the ideas behind this concept.

This whole technology is based around the interfaces `IObservable<T>` and `IObserver<T>`, the duals to the `IEnumerable<T>` and `IEnumerator<T>` interfaces. Observers subscribe to observables which push messages to them using the `OnNext` method, and inform them of completion using `OnComplete` and of errors using `OnError`. The power comes from the pre-supplied set of observables, the utility methods for converting things such as WinForms events into Observables, and the rich set of combinators for combining the Observables. The latter are designed to be usable easy via LINQ.

We could for example write some code that prints times in tenths of a second.

```
var ticktock = Observable.Interval(TimeSpan.FromSeconds(1));  
  
var clock =  
from tick in ticktock  
from tenths in Observable.Interval(TimeSpan.FromSeconds(0.1)).TakeUntil(ticktock)  
select tick + "." + tenths;  
  
clock.Subscribe(x => Console.WriteLine(x));
```

Notice the way that we subscribe to the same observable twice in order to show the use of the `TakeUntil` method. This method acts a bit like a switch allowing us to consume the output from the Observable which is ticking every tenth of a second until the Observable which fires every second fires again. That piece of LINQ syntax translates into something of the form

```
var clock =  
ticktock.SelectMany(  
    tick => Observable.Interval(TimeSpan.FromSeconds(0.1)).TakeUntil(ticktock) ,  
    (tick, tenths) => tick + "." + tenths);
```

When running under .NET 3.5 on my laptop, this gives the following output, which might not be quite the set of results you expect.

0.0  
0.1  
0.2  
0.3  
0.4  
0.5  
0.6  
0.7  
0.8  
1.0  
1.1  
1.2  
1.3  
1.4  
1.5  
480

1.6  
1.7  
1.8  
1.9  
2.0  
2.1  
2.2  
2.3

Using Observables is a little different from the lazy functional programming way of using streams of events. First, in most of the functional models, everything is single-threaded. In Rx, the user can take some control of the threading model that is used for responding to the events (using ObserveOn). In the above example, whereas in the functional model there would be a single list of events which the two references to ticktock share, in the Rx model the timer is a cold observable with the semantics that the two subscriptions are to two different timers. These two timers are independent and may not be synchronised – this is what leads to the “strange” output above.

We can study what is going on a little more, by defining our own Observable and using that instead of the timer.

```
class MyObservable : IObservable<long>
{
    List<IObserver<long>> m_Targets = new List<IObserver<long>>();
    object m_Lock = new object();

    public IDisposable Subscribe(IObserver<long> observer)
    {
        lock (m_Lock)
        {
            m_Targets.Add(observer);
            return new UnSubscribe(this, observer);
        }
    }

    class UnSubscribe : IDisposable
    {
        readonly MyObservable m_Target;
        readonly IObserver<long> m_Observer;
        public UnSubscribe(MyObservable target, IObserver<long> observer)
        {
            m_Target = target;
            m_Observer = observer;
        }

        public void Dispose()
        {
            lock (m_Target.m_Lock)
            {
                m_Target.m_Targets.Remove(m_Observer);
            }
        }
    }
}
```

```

public void Fire(long value)
{
List<IObserver<long>> targets = new List<IObserver<long>>();
lock(m_Lock)
{
targets.AddRange(m_Targets);
}
foreach (var target in targets)
{
target.OnNext(value);
}
}
}

```

We can now change the example to the following code.

```

var ticktock = new MyObservable();
var clock =
from tick in ticktock
from tenths in Observable.Interval(TimeSpan.FromSeconds(0.1)).TakeUntil(ticktock)
select tick + "." + tenths;
clock.Subscribe(x => Console.WriteLine(x));

```

If we run this code, we find that at the end of it, there is a single instance of System.Collections.Generic.AnonymousObserver<long> subscribed to our ticktock observable. If we follow the OnNext method calls, we eventually (after several steps) arrive at the code representing the “x => Console.WriteLine(x)” lambda expression.

We might wonder where the methods are that handle the “from tenths ...” code of the LINQ expression. The methods for dealing with this are setup on demand, so nothing will happen until the ticktock fires for the first time.

If we then execute

```
ticktock.Fire(1);
```

we see that there is now a second subscription to the ticktock observer. If we follow the chain of OnNext calls we find an Observable that takes input from two sources, the Interval Observable that generates every tenth of a second and the ticktock. This observable is rigged to stop the data flow once the ticktock fires again.

In memory, if we look at only the OnNext methods, we have a dataflow network that is dynamically changed depending on the firing of various events. It is this dynamic change of the network that requires knowledge of the threading model that is happening under the covers. This is particularly important in the above example where the two events from the timers will race, with one timer driving the dataflow network to lose the connection to the existing tenths timer and set up a connection to a new one, while at the same time the tenths timer is trying to push a value through to the select. The data race isn’t too hard to see in this example, but with a large network it might be tricky to determine if there are multiple subscriptions to a source, and the ordering of the push events from the multiply connected source might not be well defined.

In a recent down tools week at [3]Red Gate, a week in which the developers get to work on a project of their choosing, I spent a little time working with the memory profiler technology we

have to see if it could be used to analyse the state of the dataflow network associated with an Rx expression. We could imagine using this technology to view a dataflow network and perhaps answer questions about multiple subscriptions to the same source. Unfortunately, I didn't have enough time to get particularly far with the prototype, but I see some kind of visualization support as a vital debugging tool if this declarative style for specifying concurrency becomes prevalent.

1. <ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Geraint.Jones/PRG-TM-32.pdf>
2. <http://msdn.microsoft.com/en-us/devlabs/ee794896.aspx>
3. <http://www.red-gate.com/>

---

Reactive extensions in action | Thirty years tapping a keyboard (2018-04-18 05:46:46)  
[...] reactive extensions have been around for a long time. I remember coming across them in C # something like a decade ago, but I don't think I've seen a book or [...]

#### **4.7.3 I said stay calm (2010-07-13 20:34)**

I should have given an alternative solution in the last blog post. This uses Switch so that the timer is only subscribed to once. Switch is rather neat int hat it takes a sequence of observables and propagates the value of the latest element of this sequence.

```
var ticktock = Observable.Interval(TimeSpan.FromSeconds(1));  
var clock =  
Observable.Switch(  
ticktock.Select(  
tick => Observable.Interval(TimeSpan.FromSeconds(0.1))  
.Select(tenths => new { Tick = tick, Tenths = tenths }))  
.Select(anon => anon.Tick + "." + anon.Tenths);  
clock.Subscribe(x => Console.WriteLine(x));
```

---

#### **4.7.4 Don't believe everything you read (2010-07-18 20:10)**

[1]The Tiger That Isn't: Seeing Through a World of Numbers by Michael Blastland and Andrew Dilnot

A gentle but interesting read about interpreting statistics and metrics like those that are published in newspapers. This book uses lots of real-life examples to demonstrate how misleading

some of these numbers can be. It covers a lot of recent newspaper headlines including the many changes to the values behind the school league tables.

1. [http://www.amazon.co.uk/Tiger-That-Isnt-Through-Numbers/dp/1861978391/ref=sr\\_1\\_2?s=books&ie=UTF8&qid=1279449595&sr=1-2](http://www.amazon.co.uk/Tiger-That-Isnt-Through-Numbers/dp/1861978391/ref=sr_1_2?s=books&ie=UTF8&qid=1279449595&sr=1-2)
- 

#### **4.7.5 Back to basics (2010-07-18 20:11)**

There's a good interview with Ralph Johnson and Joe Armstrong [1]here on InfoQ where they discuss the state of OOP and how it has moved away from the initial emphasis on objects and messaging towards objects and classes. It raises some interesting points.

1. <http://www.infoq.com/interviews/johnson-armstrong-oop>
- 

### **4.8 August**

#### **4.8.1 Chip away at the problem (2010-08-01 15:24)**

[1]Microprocessor Architecture: From Simple Pipelines To Chip Multiprocessors by Jean-Loup Baer

Two things have got me interested in understanding more about computer architecture. First, at work, I'm hoping to soon be working on the performance profiler. Second, I have been interested for a long time in understanding the memory models of various programming languages, to fully understand the circumstances when locking and synchronisation is required when multiple threads communicate using shared memory.

This is a really good book on contemporary computer architecture. The first chapter gives an overview on the ways that the performance of an architecture can be measured. The second chapter then gives the basic model for a modern pipelined processor, covering an initial five stage pipeline and showing where forwarding is required and where hazards (structural, data and control) can affect the performance of the system. Caching (L1 and L2) is also covered, followed by a quick introduction to virtual memory and the TLB. Chapter 3 moves onto superscalar processors (focusing on the Alpha chip), discussing where instruction level parallelism can be uncovered. This moves on to cover out-of-order processors, including the use of register renaming via Tomasulo's algorithm – the focus of this is the Pentium P6 architecture. There

is a brief discussion of compiler determined parallelism like that in the VLIW architectures in contrast to the hardware discovery of potential parallelism.

The next few chapters walk over this material in further depth. Chapter 4 covers branch prediction in a lot of detail – mis-predicted branches may lead to a need to flush the pipeline, losing potential execution time until the pipeline refills (though this can be mitigated using speculative execution which is covered later in the book). It also covers instruction fetching, and the trace cache of the Pentium which attempts to merge basic blocks into a cache entry, and then covers register renaming in even more detail.

Chapter 5 covers instruction scheduling and memory access, covering things like store buffers which allow the processor to proceed without waiting for data to be written to main memory. Load instructions and load speculation are also covered.

Chapter 6 goes into detail about the cache hierarchy, and explains how memory latency can be hidden from the processor by using techniques such as predictive prefetching.

Chapter 7 covers multiprocessors, in particular cache coherence protocols and relaxed memory models. This is followed by chapter 8 on chip multithreading (hyperthreading) and chip multiprocessors.

The final chapter, chapter 9, covers current and future challenges including the need to make chips consume less power.

This is a really good book, which goes into a lot of detail about issues affecting the current generation of microprocessors. The author successfully communicates the trade-offs in microprocessor design and gives a great overview of the current state of the art.

1. [http://www.amazon.co.uk/Microprocessor-Architecture-Simple-Pipelines-Multiprocessors/dp/0521769922/ref=sr\\_1\\_1?ie=UTF8&qid=1280668318&sr=1-1](http://www.amazon.co.uk/Microprocessor-Architecture-Simple-Pipelines-Multiprocessors/dp/0521769922/ref=sr_1_1?ie=UTF8&qid=1280668318&sr=1-1)
- 

#### **4.8.2 Keep your ancestors hidden (2010-08-08 05:59)**

There were some slides going around at work from a recent talk from Jon Skeet on surprising aspects of C#. I'd seen a lot of them before, but the one that really surprised me was the following, where the question is which method Foo is invoked in Main.

```
class Base
{
    public virtual void Foo(int x) { }
}

class Dervived : Base
{
    public override void Foo(int x) { }

    public void Foo(double x) { }
}
```

```
static void Main(string[] args)
{
    new Derived().Foo(10);
}
```

The answer lies in the [1]member lookup mechanism.

- 
1. [http://msdn.microsoft.com/en-us/library/aa691331\(v=VS.71\).aspx](http://msdn.microsoft.com/en-us/library/aa691331(v=VS.71).aspx)

#### **4.8.3 They've got a load of class (2010-08-08 05:59)**

There is an interesting series of [1]blog posts and a video about the subject of Java classloaders on the ZeroTurnaround blog site. These articles also link to the [2]following onJava article. Interesting stuff and very different from the .NET world which I'm more used to.

- 
1. <http://www.zeroturnaround.com/blog/reloading-objects-classes-classloaders/>
  2. <http://onjava.com/pub/a/onjava/2005/01/26/classloading.html>

---

#### **4.8.4 Marvin, marvellous (2010-08-18 21:06)**

I realise I'm behind the times, but I've only just discovered the [1]DownloadHelper addin for Firefox, which allows you to download videos and convert them into a format suitable for viewing on an iPhone. This has allowed me to view a number of Google IO talks from 2008 and 2010, with the aim of understanding a little more about [2]Android. I thought the following talks were particularly good and have added some brief notes about each of them.

##### [3]A Beginner's Guide To Android

A great introduction to the Android platform with a high level overview of the parts of the system. Intents are the mechanism that allow one application to launch another to carry out some desire.

##### [4]Dalvik Virtual Machine Internals

The Dalvik VM is used to run java applications on the Android platform. The talk covered the reasons for needing a new more concise bytecode which is typically generated by converting java class files. The architecture of the VM was covered with a good explanation of how the interpreter is optimised by inlining the dispatch routine and aligning the individual bytecode handlers on instruction cache boundaries.

On Android, a modified Linux kernel is used which uses copy-on-write as a means for sharing pages between OS processes. A zygote process is created on system startup. This is a VM instance which has some system classes and libraries preloaded. When a new VM instance is required, this original VM instance is forked. Of course, this means that we want to avoid writing to pages as this will cause them to be no longer shared with the zygote. This led to the design decision of keeping the mark bits (that the garbage collector uses) separate from the objects themselves.

#### [5]A JIT Compiler For Android's Dalvik VM

A trace JIT has recently been added to the Dalvik VM. The talk covered why a trace jit is better suited to a mobile platform when compared against a method level jit. In particular, the interpreter is used for large chunks of code that are not deemed by the system to be hot. This keeps things nice and compact, with the code expansion that happens when jitting limited to the hot paths through the code. The system will freely pitch code when the generated code gets to a certain size, restarting the generation process.

#### [6]Inside The Android Application Framework

A good introduction to APK (application packages), tasks and processes. This talk covered the lifecycle events to which an application binds, allowing the application to load and persist its data state and (separately) its UI state when focus moves between applications and when applications are shut down.

#### [7]V8: Building a High Performance Javascript Engine

A great introduction to the technique that the V8 engine uses to get good performance for Javascript. In brief, they impose structure on the Javascript object instances which are essentially dictionaries, grouping together objects via a class which identifies a set of objects with the same properties (as long as they were added to the dictionary in the same order). This extra structure allows native code to be generated which can use inline caches, using the class object to guard the inlined action, to get good performance. The talk also covers the garbage collector – this has two generations with the first generation collected via copying, and the second generation using mark-sweep and mark-compact depending on heuristics.

The standard Javascript objects are also written in Javascript. This requires a bootstrap phase, where a new collection of system objects are compiled into a part of the heap which can then be dumped and reloaded into a new VM. In the old days, when I worked on Lisp systems, we had the same mechanism. Virtually all of the system was written in Lisp, and it was bootstrapped by taking an existing Lisp image, loading the new code into this image taking note of the heap segment into which it was allocated, and using this segment as the initial heap for the new system which could then read in other parts of the system. Writing the system in itself is great because performance improvements to the compiler and runtime affect the performance of the system itself.

#### [8]Optimizing Apps with the GWT Compiler

There seem to be loads of systems around for compiling a high level language down into Javascript so that it can execute in a browser. GWT is one of the more mature systems and this talk gave a good overview of some of the improvements to the GWT system.

1. <http://www.downloadhelper.net/>
  2. <http://www.android.com/>
  3. <http://www.youtube.com/watch?v=yqCj83leYRE>
  4. <http://www.youtube.com/watch?v=ptjedOZEXPM>
  5. <http://www.youtube.com/watch?v=Ls0tM-c4Vfo>
  6. <http://www.youtube.com/watch?v=TkPiXRNee7A>
  7. <http://www.youtube.com/watch?v=Ls0tM-c4Vfo>
  8. <http://www.youtube.com/watch?v=qT6ZsQBM7kY>
- 

#### **4.8.5 I really did read it! (2010-08-22 18:18)**

[1]An Imaginary Tale: The Story of "i" by Paul J Nahin

This was a very interesting read on the history and application of "i", the square root of -1.

It starts with the history of the solution of the polynomial equations and explains the confusion that the mathematicians of the time had when they found they could manipulate  $\text{SQRT}(-1)$  to get valid solutions of equations with real roots, despite them not believing that such a value could exist. The book covers the geometrical justifications of allowing such a root, and then moves on to Argand diagrams. There are then several chapters giving various uses of the complex numbers, showing their use in deriving various power series expansions, Kepler's laws and some properties of various circuits from electronics. The final chapter covers complex function theory and Cauchy's Integral theorems which can be used to calculate various complicated integrals. Some of these results are almost magical in the simplicity of their derivation.

The book takes some work in places, with sometimes a page or two of algebra to derive a result. I think this is one of the appeals of the book, in that the author tries to make things fairly rigorous, at the expense of giving the reader some work to do.

1. [http://www.amazon.co.uk/Imaginary-Tale-Story-square-minus/dp/0691127980/ref=sr\\_1\\_2?ie=UTF8&s=books&qid=1282499997&sr=8-2](http://www.amazon.co.uk/Imaginary-Tale-Story-square-minus/dp/0691127980/ref=sr_1_2?ie=UTF8&s=books&qid=1282499997&sr=8-2)
- 

#### **4.8.6 I know I'm ponderously slow (2010-08-22 18:18)**

Sometimes the [1]IBM Ponder has a solution that is really beautiful. I think the recent [2]July challenge has such a solution which is really clever and, to me, unexpected.

1. [http://domino.research.ibm.com/Comm/wwwr\\_ponder.nsf/pages/index.html](http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/pages/index.html)
2. [http://domino.research.ibm.com/Comm/wwwr\\_ponder.nsf/Challenges/July2010.html](http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/Challenges/July2010.html)

---

#### 4.8.7 Time to be a little more dynamic (2010-08-23 16:57)

C # 4 added support for the new dynamic type, which uses a version of the DLR to dispatch calls at runtime, allowing objects to change their behaviour quite dramatically.

At the C # level, it is relatively easy to implement a dynamic object by inheriting from the `DynamicObject` class. This convenience class makes it really easy to implement dynamic objects, though to really see what's going on one needs to look at the `IDynamicMetaObjectProvider` interface.

To do a little experimentation, I defined the following class:

```
class MyBaseObject : DynamicObject
{
    public override bool TryGetMember(GetMemberBinder binder, out object result)
    {
        result = "Object1";
        return true;
    }
}
```

This dynamic object is going to return the string "Object1" from any member call that is made on it.

In order to study the call site caching that happens, I also defined the following classes.

```
class MyObject1 : MyBaseObject { }
class MyObject2 : MyBaseObject { }
class MyObject3 : MyBaseObject { }

...
class MyObject11 : MyBaseObject { }
```

We'll use the following test program to drive the creation of the dynamic objects.

```
object[] targets = { new MyObject1(), new MyObject1(), new MyObject1(),
    new MyObject2(), new MyObject1(), new MyObject2(),
    new MyObject3(), new MyObject4(), new MyObject5(),
    new MyObject6(), new MyObject7(), new MyObject8(),
    new MyObject9(), new MyObject10(), new MyObject1()
};

foreach (dynamic target in targets)
{
    dynamic result = target.Foo;
}
```

Using Reflector, with the optimization set at 2.0, we see how the dynamic call is implemented. The foreach loop in the above translates to the following code.

```
foreach (object obj2 in objArray)
{
```

```

if (<Main>o __SiteContainer0.<>p __Site1 == null)
{
<Main>o __SiteContainer0.<>p __Site1 =
CallSite<Func<CallSite, object, object>>.Create(
    Binder.GetMember(CSharpBinderFlags.None, "Foo", typeof(Program),
new CSharpArgumentInfo[] {
    CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null)
    }));
}
object obj3 = <Main>o __SiteContainer0.<>p __Site1.Target(<Main>o __SiteContainer0.<>p __Site1, obj2);
}

```

I wanted to look at the dynamic behaviour, so the next thing to do was to ensure that the assembly containing the interesting code is unoptimised so that I could step through it using Reflector Pro. We can do this by uninstalling the default optimised ngen version.

c:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe uninstall System.Core.dll

Create an ini file to turn off optimization

```

>type System.Core.ini
[.NET Framework Debugging Control]
GenerateTrackingInfo=1
AllowOptimize=0

```

And then ngen the assembly again. This time the ini file will ensure that there is no optimisation.

c:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe install System.Core.dll

If you use Reflector to follow the code from CallSite<.>.Create, you see that the Target property holds a delegate that calls UpdateAndExecute1. I therefore decompiled System.Core using Reflector Pro and navigated to the UpdateDelegates class where I put a breakpoint on UpdateAndExecute1. On the first call to target.Foo, with target bound to MyObject1, we enter this method.

For a given call site, the system maintains some caches that hold dispatches that we've made in the past at this call site. On the first call, we drop through to

func = site2.Target = site2.Binder.BindCore<Func<CallSite, T0, TRet>>(site2, args);

which is code that is going to compile a test and fast dispatch to a given set of typed arguments.

The code passes through a set of methods until it eventually reaches the Compile method in System.Linq.Expression.Expression<T>. Grabbing the expression tree, by going to the locals window and using "Make ObjectID", we can later look at the code to determine that it is compiling the following delegate. The debugger has a view for exposing Expression Trees in the textual format seen below.

```

.Lambda      CallSite.Target<System.Func`3[System.Runtime.CompilerServices.CallSite,System.Object,System.Object]>(
System.Runtime.CompilerServices.CallSite $ $site,
System.Object $ $arg0) {
    .Block() {
        .If (
            $ $arg0 .TypeEqual ConsoleApplication14.MyObject1
        ) {

```

```

.Return #Label1 { .Block(System.Object $var1) {
.If (
.Call ((System.Dynamic.DynamicObject) $ $arg0).TryGetMember(
.Constant<System.Dynamic.GetMemberBinder>(Microsoft.CSharp.RuntimeBinder.CSharpRuntimeBinder),
$var1)
) {
$var1
} .Else {
.Throw .New Microsoft.CSharp.RuntimeBinder.RuntimeBinderException("ConsoleApplication14.MyObject1' does not contain a definition for 'Foo'")
}
}
} .Else {
.Default(System.Void)
};
.Label
.LabelTarget CallSiteBinder.UpdateLabel:;
.Label
.If (
.Call System.Runtime.CompilerServices.CallSiteOps.SetNotMatched( $ $site)
) {
.Default(System.Object)
} .Else {
.Invoke (((System.Runtime.CompilerServices.CallSite`1[System.Func`3[System.Runtime.CompilerServices.CallSite,System.Object,System.Object]])) $ $site).Update)(
$ $site,
$ $arg0)
}
.LabelTarget #Label1:
}
}

```

The cool thing is that the Target delegate of the callsite is set to point to the compiled version of this expression tree. Hence the next time we call target.Foo on an instance of MyObject1, we'll dispatch really quickly into the TryGetMember code, and will continue to do so until we try to dispatch on something that isn't of type MyObject1. On the way out, we add the compiled expression tree to the Rules associated with the CallSite. We can store at most 10 such rules, and we'll see where they come in to play in a little while.

When we make a call on MyObject2, the code above misses the fast path and we go back into UpdateAndExecute1. The code in this method looks at the rules that we have previously compiled and tries to find something that works. On this pass it doesn't find anything, so we go back into the compile code and compile the following ExpressionTree.

```

.Lambda      CallSite.Target<System.Func`3[System.Runtime.CompilerServices.CallSite,System.Object,System.Object]>(
System.Runtime.CompilerServices.CallSite $ $site,
System.Object $ $arg0) {
.Block() {
.If (
$ $arg0 .TypeEqual ConsoleApplication14.MyObject2

```

```

) {
.Return #Label1 { .Block(System.Object $var1) {
.If (
.Call ((System.Dynamic.DynamicObject) $ $arg0).TryGetMember(
.Constant<System.Dynamic.GetMemberBinder>(Microsoft.CSharp.RuntimeBinder.CS harpGe-
tMemberBinder),
$var1)
) {
$var1
} .Else {
.Throw .New Microsoft.CSharp.RuntimeBinder.RuntimeBi nderException("ConsoleApplication14.MyOb-
ject2' does not contain a definition for 'Foo'")
}
} }
} .Else {
.Default(System.Void)
};
.Label
.LabelTarget CallSiteBinder.UpdateLabel:;
.Label
.If (
.Call System.Runtime.CompilerServices.CallSiteOps.SetNotMatched( $ $site)
) {
.Default(System.Object)
} .Else {
.Invoke (((System.Runtime.CompilerServices.CallSite`1[System.Func`3[System.Runtime.Compil-
erServices.CallSite,System.Object,System.Object]])) $ $site).Update)(
$ $site,
$ $arg0)
}
.LabelTarget #Label1:
}
}

```

This is the same as the first expression tree, apart from the change to the guard type. Again, this compiled delegate is set into the Target property of the CallSite so we'll quickly dispatch to the right code for future calls to objects of type MyObject2 at this call site, and it is added to the rules for the callsite.

In our test program, we now dispatch to an instance of MyObject1. This causes the fast path to miss, and we get back into the UpdateAndexecute1 method.

The system gets the Rules for the CallSite, and in this case there are two that are applicable. One matches the MyObject1 case and the other matches the MyObject2 case. We pull the MyObject1 dispatcher from the rules cache, and set it into the Target property. We dispatch the call, notice it has been successful, and before we return the result, we call UpdateRules to move the rule that was successful earlier in the Rules array. This means that when new rules come into effect, it will take longer for it to be flushed out of the 10 item cache. We have also set the Target property to the matching rule, so we'll dispatch to it quickly if we make another call on the same type.

The test program then goes on to make lots of calls on different types of target.

The Rules cache for the CallSite holds at most 10 items - roughly, the ten most recently used types at that call site. The BindCore method also maintains a cache at the level of the Binder for the call site, so that all call sites sharing the same binder can share the compiled results.

This is the cache that is used by the code that does:

```
RuleCache<Func<CallSite, T0, TRet>> ruleCache = CallSiteOps.GetRuleCache<Func<CallSite, T0, TRet>>(site2);
rules = ruleCache.GetRules();
```

This cache is also bounded, but can hold a lot more items which are more globally applicable. This cache is searched only after the site specific cache has been fully tried. This is a clever way to exploit the observation that for many programs, we tend to dispatch for large amounts of time on the same type at a given call site.

An easy way to see the optimised calling is to amend the code to:

```
class MyBaseObject : DynamicObject
{
    public override bool TryGetMember(GetMemberBinder binder, out object result)
    {
        System.Diagnostics.Debugger.Break();
        result = "Object1";
        return true;
    }
}
```

and then run under windbg with the sos extension loaded. The first three calls are to a target of type MyBaseObject... during the first call the cache is set up, but the subsequent two calls don't go through the setup code. The fourth call targets a different object type and therefore goes back through the cache setup code.

```
0:000> !ClrStack
OS Thread Id: 0x1b2c (0)
Child SP IP Call Site
0042eecc 76ed22a1 [HelperMethodFrame: 0042eecc] System.Diagnostics.Debugger.BreakInternal()
0042ef24 62ee00e6 System.Diagnostics.Debugger.Break()
0042ef50 001f0777 ConsoleApplication14.MyBaseObject.TryGetMember(System.Dynamic.GetMemberer, System.Object ByRef)
0042ef68 0021017b DynamicClass.CallSite.Target(System.Runtime.CompilerServices.Closure, System.Runtime.CompilerServices.CallSite, System.Object)
0042ef88 60ac5ed3 System.Dynamic.UpdateDelegates.UpdateAndExecute1[[System.__Canon, mscorelib],[System.__Canon, mscorelib]](System.Runtime.CompilerServices.CallSite, System.__Canon)
0042efe0 001f0429 ConsoleApplication14.Program.Main(System.String[])
0042f2ac 634521db [GCFrame: 0042f2ac]

0:000> !ClrStack
OS Thread Id: 0x1b2c (0)
Child SP IP Call Site
0042ef24 76ed22a1 [HelperMethodFrame: 0042ef24] System.Diagnostics.Debugger.BreakInternal()
0042ef7c 62ee00e6 System.Diagnostics.Debugger.Break()
```

```

0042efa8 001f0777 ConsoleApplication14.MyBaseObject.TryGetMember(System.Dynamic.GetMemberBind-
er, System.Object ByRef)
0042efc0 0021017b DynamicClass.CallSite.Target(System.Runtime.CompilerServices.Closure,
System.Runtime.CompilerServices.CallSite, System.Object)
0042efe0 001f0429 ConsoleApplication14.Program.Main(System.String[])
0042f2ac 634521db [GCFrame: 0042f2ac]

0:000> !ClrStack
OS Thread Id: 0x1b2c (0)
Child SP IP Call Site
0042ef24 76ed22a1 [HelperMethodFrame: 0042ef24] System.Diagnostics.Debugger.BreakInternal-
I()
0042ef7c 62ee00e6 System.Diagnostics.Debugger.Break()
0042efa8 001f0777 ConsoleApplication14.MyBaseObject.TryGetMember(System.Dynamic.GetMemberBind-
er, System.Object ByRef)
0042efc0 0021017b DynamicClass.CallSite.Target(System.Runtime.CompilerServices.Closure,
System.Runtime.CompilerServices.CallSite, System.Object)
0042efe0 001f0429 ConsoleApplication14.Program.Main(System.String[])
0042f2ac 634521db [GCFrame: 0042f2ac]

0:000> !ClrStack
OS Thread Id: 0x1b2c (0)
Child SP IP Call Site
0042eeac 76ed22a1 [HelperMethodFrame: 0042eeac] System.Diagnostics.Debugger.BreakInternal-
I()
0042ef04 62ee00e6 System.Diagnostics.Debugger.Break()
0042ef30 001f0777 ConsoleApplication14.MyBaseObject.TryGetMember(System.Dynamic.GetMemberBind-
er, System.Object ByRef)
0042ef48 0021030b DynamicClass.CallSite.Target(System.Runtime.CompilerServices.Closure,
System.Runtime.CompilerServices.CallSite, System.Object)
0042ef68 60ac5ed3 System.Dynamic.UpdateDelegates.UpdateAndExecute1[[System. _ 
_Canon, mscorlib],[System. _ _Canon, mscorlib]](System.Runtime.CompilerServices.CallSite,
System. _ _Canon)
0042efc0 002101db DynamicClass.CallSite.Target(System.Runtime.CompilerServices.Closure,
System.Runtime.CompilerServices.CallSite, System.Object)
0042efe0 001f0429 ConsoleApplication14.Program.Main(System.String[])
0042f2ac 634521db [GCFrame: 0042f2ac]

```

The extended Expression trees allow us to express a lot of constructs from many source languages. One quick example would be a simple lambda expression which prints a message and then returns a value.

```

BlockExpression res =
Expression.Block(

```

```

Expression.Call(typeof(Console).GetMethod("WriteLine", new Type[] { typeof(string) }),
Expression.Constant("Hello"),
Expression.Constant(2));
var delegateItem = Expression.Lambda(res).Compile();
Console.WriteLine(delegateItem.DynamicInvoke());

```

The above code prints Hello and 2. The runtime contains a compiler for converting these expression trees into delegates making this an easy way to implement high level languages.

What I really started out wanting to investigate was the `IDynamicMetaObjectProvider` interface, but that will have to wait until next time.

---

#### 4.8.8 Get rid of it quickly (2010-08-30 17:18)

You sometimes see pieces of code which set local variables to null with the aim of releasing instances early. In debug builds, local variable lifetimes are extended to the end of the method, but for release builds I think you'd expect the jit compiler to calculate the liveness information (in most situations) to allow resources to be freed early.

Given a class `Test1`, I want to have a quick look at the liveness information that is calculated for the method:

```
void DoTest()
{
    Type1 target = new Type1();
    Type1 target2 = new Type1();
    Type1 target3 = new Type1();
    Type1 target4 = new Type1();
    Type1 target5 = new Type1();
    target.Hello();
    target2.Hello();
    target3.Hello();
    target4.Hello();
    target5.Hello();
    Console.WriteLine("Finished");
}
```

Using the `sos gcinfo` command, we get the following output.

```
0:000> !gcinfo 003C00A0
entry point 003c00a0
Normal JIT generated code
GC info 000e196c
Method info block:
method size = 00B6
prolog size = 9
epilog size = 8
epilog count = 1
epilog end = yes
callee-saved regs = EDI ESI EBX EBP
ebp frame = yes
fully interruptible= no
double align = no
arguments size = 0 DWORDs
```

stack frame size = 2 DWORDs

untracked count = 0

var ptr tab count = 2

epilog at 00AE

argTabOffset = 6

81 36 E5 C6 2A |

02 06 |

Pointer table:

10 16 5C | 0016..0072 [EBP-10H] a pointer  
14 51 2C | 0067..0093 [EBP-14H] a pointer  
B2 40 | 0032 call [ ESI ] argMask=00  
2A | 003C call [ ESI ] argMask=00  
3A | 0046 call [ EDI ESI ] argMask=00  
3A | 0050 call [ EDI ESI ] argMask=00  
7A | 005A call [ EDI ESI EBX ] argMask=00  
7A | 0064 call [ EDI ESI EBX ] argMask=00  
7B | 006F call [ EDI ESI EBX ] argMask=00  
79 | 0078 call [ EDI ESI EBX ] argMask=00  
58 | 0080 call [ EDI EBX ] argMask=00  
48 | 0088 call [ EBX ] argMask=00  
FF |

Note that in the above C # code, I needed to keep going until five local variables to make sure that there is a need to spill some locals on to the stack. This is the disassembled code for the method which we can cross-reference with the pointer tracking information from gcinfo.

```
0:000> u 003C00A0
003c00a0 55 push ebp
003c00a1 8bec mov ebp,esp
003c00a3 57 push edi
003c00a4 56 push esi
003c00a5 53 push ebx
003c00a6 83ec08 sub esp,8
003c00a9 b9b0380e00 mov ecx,0E38B0h
003c00ae e86d1fd1ff call 000d2020
```

We have now allocated the object in the local variable target. This is returned in eax, which we place into the [ebp-10h] spill. The pointer tracking, above, notes that the spill contains a tracked value after this instruction.

```
003c00b3 8945f0 mov dword ptr [ebp-10h],eax
003c00b6 8bc8 mov ecx,eax
003c00b8 ff15f0380e00 call dword ptr ds:[0E38F0h]
003c00be b9b0380e00 mov ecx,0E38B0h
003c00c3 e8581fd1ff call 000d2020
003c00c8 8bf0 mov esi,eax
```

The local variable target2 is held in esi. This is live for the next 8 call instructions according to the pointer map from the gcinfo. Information such as

call [ EDI ESI ] argMask=00

is showing that EDI and ESI contain live pointers at the point of the calls.

```
003c00ca 8bce mov ecx,esi
003c00cc ff15f0380e00 call dword ptr ds:[0E38F0h]
003c00d2 b9b0380e00 mov ecx,0E38B0h
003c00d7 e8441fd1ff call 000d2020
003c00dc 8bf8 mov edi,eax
003c00de 8bcf mov ecx,edi
003c00e0 ff15f0380e00 call dword ptr ds:[0E38F0h]
003c00e6 b9b0380e00 mov ecx,0E38B0h
003c00eb e8301fd1ff call 000d2020
003c00f0 8bd8 mov ebx,eax
003c00f2 8bcb mov ecx,ebx
003c00f4 ff15f0380e00 call dword ptr ds:[0E38F0h]
003c00fa b9b0380e00 mov ecx,0E38B0h
003c00ff e81c1fd1ff call 000d2020
```

On the next instruction we spill the local target5. This will remain a valid pointer source until offset 0093.

```
003c0104 8945ec mov dword ptr [ebp-14h],eax
003c0107 8bc8 mov ecx,eax
003c0109 ff15f0380e00 call dword ptr ds:[0E38F0h]
003c010f 8b4df0 mov ecx,dword ptr [ebp-10h]
003c0112 ff15a4380e00 call dword ptr ds:[0E38A4h]
```

The pointer tracking for [ebp-10h], the spill holding target, is no longer live according to the pointer tracking information. [ebp-10h] was live from 0016..0072. The local variable target is now dead.

```
003c0118 8bce mov ecx,esi
003c011a ff15a4380e00 call dword ptr ds:[0E38A4h]
```

Esi is no longer a valid pointer and hence is not included in the live register information for subsequent calls.

```
003c0120 8bcf mov ecx,edi
003c0122 ff15a4380e00 call dword ptr ds:[0E38A4h]
003c0128 8bcb mov ecx,ebx
003c012a ff15a4380e00 call dword ptr ds:[0E38A4h]
003c0130 8b4dec mov ecx,dword ptr [ebp-14h]
003c0133 ff15a4380e00 call dword ptr ds:[0E38A4h]
```

[ebp-14h], the spill for target5, is now not longer a source of pointers. The target5 local is now dead.

```
003c0139 e8926f525b call mscorlib _ni+0x2570d0 (5b8e70d0)
003c013e 8bc8 mov ecx,eax
003c0140 8b1530200303 mov edx,dword ptr ds:[3032030h]
003c0146 8b01 mov eax,dword ptr [ecx]
003c0148 8b403c mov eax,dword ptr [eax+3Ch]
003c014b ff5010 call dword ptr [eax+10h]
```

```
003c014e 8d65f4 lea esp,[ebp-0Ch]
003c0151 5b pop ebx
003c0152 5e pop esi
003c0153 5f pop edi
003c0154 5d pop ebp
003c0155 c3 ret
```

From the liveness information, we can see that locations holding the various locals are only tracked for some of the method scope. Hence there is no need to set such locals to null as the liveness information means that the values they hold are not going to be traced by the garbage collector anyway.

---

#### 4.8.9 That's one aspect of the problem (2010-08-30 17:22)

[1]AspectJ In Action: Enterprise AOP with Spring Applications by Ramnivas Laddad

I came across aspect oriented programming around 2002 when the development manager at the company where I was working asked me to do a talk on the subject. At the time, the main exponent of this technology was the AspectJ compiler that had been developed by Gregor Kiczales, who we respected as one of the main designers of the object system that was included in the Common Lisp specification.

This book shows just how far this technology has come over the years and how well it is supported by Spring, one of the large Java application frameworks.

Aspects are a means of dealing with many of the cross-cutting concerns that can be found in object oriented applications. A program in an object oriented language chooses one of the many possible decompositions of a problem into groups of inheritance related classes... this always leaves related functionality (think tracing or security) that is scattered across the classes with no easy way of modularising this functionality. Interfaces give a way of relating non-inheritance related parts of the main inheritance tree, but the single inheritance found in Java and C # doesn't allow these interfaces to take state into the related items. Aspect oriented programming allows mixins which step over this problem while enabling so much more.

The book introduces AOP (aspect oriented programming) via AspectJ and its syntax extensions to standard Java. From the dynamic point of view, pointcuts identify collections of join points which can have advice added to them in the form of code which is executed before, after (either always or on value return or if the method exits via an exception) or around the existing code. In the latter case, the user code uses the new keyword proceed() to cause execution of the original code, or alternatively can call proceed(...) modifying the arguments that the original code runs on (though with the limitation that the matching method cannot be changed).

For example, we could modify the program to print a message before any of the deliver methods in the MessageCommunicator class are executed, by using the following:

```
pointcut secureAccess() : execution(* MessageCommunicator.deliver(..));
```

```
before() : secureAccess()  
System.out.println("Checking access...")
```

The AspectJ syntax allows a lot of flexibility in the selection of the join points. We can, for example, select by method name, arguments, return type, annotation and which type the method is owned by. Moreover, one can test for method invocations happening inside other join points and even capture the arguments so that they can be used inside the added advice. There are also facilities for reflection over the join point inside the advice. As well as allowing dynamic modification, the system also allows the static structure of the program to be changed (at compile time). Interfaces and annotations can be added to existing classes, and instances of standard classes and aspects can be associated in several ways – aspects can be associated per type, per object or per control flow, and default to having a singleton instance of the aspect in the system. Moreover, there is a means to raise errors and warning if particular point cuts are empty or non-empty.

In order to make adoption easier, AspectJ has a second version of its syntax which uses Java Annotations instead of syntax extensions. This means that programs can be compiled using a standard java compiler, though with the downside of throwing away some of the potential error checking.

The book goes into some detail about how aspects are woven into the existing code to produce the final program. This weaving can happen at compile time, as a post compile action or as a load time step, or even, in the case of Spring, by using proxy objects instead of the real objects and adding the advice as decorators called by the proxy. In this last case, no special compilation is needed, but the proxy implementation cannot add advice to calls via the this pointer which will always end up unproxied. The book goes into details about the advantages of these various implementation strategies, and points to a couple of papers including [2]this one.

There are some really good examples in the book of the use of aspects. Adding change notification functions to properties is one of the classic examples where a lot of boiler plate code typically has to be written which has a standard form which could be made into an aspect. The book shows how easy this pattern is to implement inside an aspect, leaving the original code much easier to read, though you'd really need tool support to understand how the aspects mix into your existing java code. The book emphasises the fact that the aspect can be abstract, allowing parts such as the point cut to be defined in another place closer to where the aspect is used. Aspects can inherit from abstract aspects but not from concrete aspects (though this limitation might go in the future).

There are loads of other good examples in the book, all with great explanations. An aspect that enforces restrictions on the return value such that collection returning methods must return an empty collection instead of null, and an aspect that ensures that code that modifies the GUI runs on the gui thread, together with a related aspect that adds locking to a reader-writer pattern.

The book also introduces a number of new patterns.

The wormhole: The cflow join point is used to transfer values from an invocation higher up the stack to the current advice without needing to pass the data as a parameter or put it into thread local storage.

The worker: The around advice uses a closure to move the execution of the original code on to another thread.

The Participant pattern: which uses abstract pointcuts whose meaning can be defined nearer to where the aspects are used.

The book finishes by covering transactions and security, which are the standard example of aspects.

The explanations in the book are fantastic and the practical advice is worth every penny.

1. [http://www.amazon.co.uk/AspectJ-Action-Enterprise-Spring-Applications/dp/1933988053/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1283171969&sr=8-1](http://www.amazon.co.uk/AspectJ-Action-Enterprise-Spring-Applications/dp/1933988053/ref=sr_1_1?ie=UTF8&s=books&qid=1283171969&sr=8-1)
  2. <http://hugunin.net/papers/aosd-2004-cameraReady.pdf>
- 

## 4.9 September

### 4.9.1 Get down! (2010-09-02 19:29)

We saw before how inheriting from `DynamicObject` allows us to quickly write an object which gets a chance to handle operations on the object, like method invocations and property access. We didn't need to do anything with expression trees using this approach, but in order to get more of an idea of what is happening, in this post we'll delve into the world of `IDynamicMetaObjectProvider` and get to the point where we can start playing with expression trees.

We're going to define a dynamic object of type `MyDynamicObject` which we'll exercise using the following harness.

```
static void Main(string[] args)
{
    foreach (dynamic target in new dynamic[] { new MyDynamicObject(), new MyDynamicObject() })
    {
        foreach (object targetArgument in new object[] {
            1, 2, 3, "string1", 4, 5, "string2", 1.2, "string3", "string4", target, target })
        {
            target.Dolt(targetArgument);
        }
    }
}
```

The skeleton of our implementation is going to be

```
class MyDynamicObject : IDynamicMetaObjectProvider
```

```
{
```

```
    public object OnInt(int argument)
```

```
    {
        Console.WriteLine("On int {0}", argument);
        return argument;
    }
}
```

```

public object OnSomethingElse(object argument)
{
    Console.WriteLine("On something else {0}", argument);
    return argument;
}

```

We'll implement an object that redirects any method invocation so that the method `OnInt` is called if the method's first argument is an int, and `onSomethingElse` if the first argument is of another type.

We implemented `IDynamicMetaObjectProvider` which requires us to implement the method `GetMetaObject`. We'll define this as follows.

```

public DynamicMetaObject GetMetaObject(Expression expression)
{
    Console.WriteLine("Provide meta object");
    return new MyDynamicMetaObject(expression, this);
}

```

The incoming expression is something that will evaluate to the target object. We pass this though to the nested dynamic metaobject subclass which we define as follows.

```

class MyDynamicMetaObject : DynamicMetaObject
{
    public MyDynamicMetaObject(Expression parameter, MyDynamicObject self)
        : base(parameter, BindingRestrictions.Empty, self)
    {
    }
}

```

The next method is the one that handles the code generation. We check the type of the first of the incoming arguments. If it is int then we are going to generate a call which uses the `OnInt` method, otherwise we'll direct to the `OnSomethingElse` method. In the `DynamicMetaObject` that we return, we provide a restriction which demands that the argument is of the relevant type. The DLR will use this restriction to determine if we get a hit in the inline cache for the next call through the call site.

```

public override DynamicMetaObject BindInvokeMember(InvokeMemberBinder binder, DynamicMetaObject[] args)
{
    Console.WriteLine("BindInvokeMember on {0}", binder.Name);

    if (args[0].LimitType == typeof(int))
    {
        return new DynamicMetaObject(
            Expression.Call(
                Expression.Convert(Expression, typeof(MyDynamicObject)),
                typeof(MyDynamicObject).GetMethod("OnInt"),
                Expression.Convert(args[0].Expression, typeof(int))),

            BindingRestrictions.GetTypeRestriction(args[0].Expression, typeof(int)))
    };

    Type targetType = args[0].LimitType;

```

```

DynamicMetaObject result =
new DynamicMetaObject(
Expression.Call(
Expression.Convert(Expression, typeof(MyDynamicObject)),
typeof(MyDynamicObject).GetMethod("OnSomethingElse"),
Expression.Convert(args[0].Expression, typeof(object))),

BindingRestrictions.GetTypeRestriction(args[0].Expression, targetType)
);

return result;
}
}

```

Running this gives the following output. From the output we can see the inline caching that the DLR is doing to avoid needing to call to our metaobject if the restrictions we provide when we return an expression tree are valid. So, for example, the DLR calls our metaobject the first time the incoming first argument is an integer, and then doesn't need to call again into the metaobject until the argument type changes, on the call with "string1". When we call again on an int, the metaobject isn't needed as the cache already has the code to execute for the int case.

```

Provide meta object
BindInvokeMember on Dolt
On int 1
On int 2
On int 3
Provide meta object
BindInvokeMember on Dolt
On something else string1
On int 4
On int 5
On something else string2
Provide meta object
BindInvokeMember on Dolt
On something else 1.2
On something else string3
On something else string4
Provide meta object
Provide meta object
BindInvokeMember on Dolt
On something else ConsoleApplication1.MyDynamicObject
On something else ConsoleApplication1.MyDynamicObject
On int 1
On int 2
On int 3
On something else string1
On int 4
On int 5
On something else string2
On something else 1.2

```

```
On something else string3
On something else string4
On something else ConsoleApplication1.MyDynamicObject
On something else ConsoleApplication1.MyDynamicObject
```

---

#### **4.9.2 Steady! (2010-09-05 13:36)**

[1]The Drunkard's Walk: How Randomness Rules Our Lives by Leonard Mlodinow

An ace book on probability and statistics, relating the theory to the effects we see in our normal lives.

Just how confident can we be that a fund manager who has achieved above average results for a couple of years didn't just get these results due to the normal run of random events? If an HIV test comes back positive, what does that really tell us about the chance of us dying in the next ten years?

The book is really well written, drawing the reader in with historical anecdotes about the discovery of various parts of probability theory which gives a great setting for the explanations. There are loads of interesting questions. For example, in the Bayes' theorem section, the author presents the difference in probability between having two girls in a family in the situations where you know the family has two children and one of them is a girl ( $1/3$ ) and where you know the family has two children and one of them is a girl named Florida ( $1/2$ ). Simple puzzles like this keep the reader interested, and the many interesting observations on sampling and the human inclination to find patterns where there are none, make this a really good book.

---

1. [http://www.amazon.co.uk/Drunkards-Walk-Randomness-Rules-Lives/dp/0141026472/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1283669915&sr=8-1](http://www.amazon.co.uk/Drunkards-Walk-Randomness-Rules-Lives/dp/0141026472/ref=sr_1_1?ie=UTF8&s=books&qid=1283669915&sr=8-1)

---

#### **4.9.3 Break it down (2010-09-05 13:37)**

At the weekend, my son showed me a piece of encrypted text in one of the books he was reading. It was obviously there to be broken, but it was really good fun doing frequency analysis on the letters to check that it was a [1]substitution cipher, and then using guesswork to break the message down. In this case, the big clue was the presence of two words TCO and TCOJ. The exercise made it clear just how easy this kind of scheme is to break, so never use it to protect anything of value!

1. [http://en.wikipedia.org/wiki/Substitution\\_cipher](http://en.wikipedia.org/wiki/Substitution_cipher)
- 

#### **4.9.4 Visibility is bad (2010-09-18 07:26)**

[1]Cloud Application Architectures: Building Applications and Infrastructure in the cloud by George Reese

A book aimed at systems administrators outlining the cost benefits of moving to a cloud architecture, and covering issues such as firewalls, back ups and disaster recovery as well as strategies for scaling. It provides a brief introduction to EC2 and S3, with an appendix documenting the various command line utilities for dealing with the EC2 system. A good read for getting a general feel for this new way to deploy your applications, but without a lot of technical depth.

1. [http://www.amazon.co.uk/Cloud-Application-Architectures-Infrastructure-Transactional/dp/0596156367/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1284793421&sr=8-1](http://www.amazon.co.uk/Cloud-Application-Architectures-Infrastructure-Transactional/dp/0596156367/ref=sr_1_1?ie=UTF8&s=books&qid=1284793421&sr=8-1)
- 

#### **4.9.5 2000 years and you haven't reached five? (2010-09-18 07:27)**

[1]The Fifth Postulate: How unraveling the two-thousand-year-old mystery unraveled the universe by Jason Socrates Bardi

I remember the first time I learned of non-Euclidean geometries, and how shocked I was that you could model the normal axioms using a sphere, identifying "lines" with great circles and "points" with pairs of opposite points in order to produce a model where the parallel postulate didn't apply.

This book covers the story of the discovery of non-Euclidean geometry, starting with the Greeks and the book of Euclid, covering some of the numerous attempts to prove the fifth postulate from the other axioms. The story then focuses on Gauss who was connected to the two people, Lobachevsky and Bolyai, who eventually published their (at the time) controversial ideas of hyperbolic geometry. Gauss never published on the subject, but may well have had such ideas himself.

The book is full of interesting anecdotes and biographical detail about the central characters, though there is a little too much repetition. I'm not sure it really explains how non-Euclidean geometry unravelled the universe, but it was an interesting read nevertheless.

1. [http://www.amazon.co.uk/Fifth-Postulate-Unraveling-Thousand-Unraveled/dp/0470149094/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1284794079&sr=1-1](http://www.amazon.co.uk/Fifth-Postulate-Unraveling-Thousand-Unraveled/dp/0470149094/ref=sr_1_1?s=books&ie=UTF8&qid=1284794079&sr=1-1)
- 

#### 4.9.6 Ouch that's sharp (2010-09-23 13:01)

[1]Community for F # had a good Livemeeting the other day, where Adam Granicz gave a presentation on [2]WebSharper, a framework of F # libraries that makes it possible to write WebApps in F #. The system translates the client side pieces from F # into javascript, and allows easy communication between the two parts of the system. He gave a demo of a shopping cart type application which was all written in F # and which did most of the processing on the client. He emphasised how easy it was to interoperate between the F # code and other existing javascript libraries. Additionally, the F # contains an embedded DSL for writing the HTML of controls and pages. The only thing he didn't mention was how one debugged the generated javascript - I imagine you'd need to have a good knowledge of the translation to map back from the javascript in a debugger like FireBug to the F # code that you need to fix. This is certainly an interesting approach for writing web applications.

There's some documentation on the translation [3]here and [4]here.

1. <http://www.communityforfsharp.net/>
  2. <http://www.websharper.com/>
  3. <http://www.intellifactory.com/docs/websharper/ar01s02.html>
  4. <http://www.intellifactory.com/docs/websharper/ar01s03.html>
- 

#### 4.9.7 Get that through the eye of a needle (2010-09-23 13:13)

[1]Threaded Interpretive Languages by R.G.Loeliger

I found this book in Oxfam at the weekend and was immediately hooked. Written in 1981, it documents the implementation of FORTH on a Z80 processor. It contains the code for the full implementation in a combination of Z80 assembler and FORTH in FORTH. The book is really interesting, and answers a number of questions I've had about the implementation since the early-80s when I had FORTH running on my BBC Micro.

FORTH, in case you haven't seen it before, is a language that was designed to be easily portable and was targeted at embedded languages. It is a stack based language, with operations acting on the stack. To do a calculation such as  $(2+3)*5$ , you could enter the following expression at the FORTH system, noting that . pops the top value off the stack and prints it to the console.

5 2 3 + \* .

which would respond

25 OK

There are many operations that you can apply to the stack. DUP duplicates the top item on the stack, so we could define an operation to cube the top value on the stack by defining

: TRIPLE DUP DUP \* \* ;

and then execute this using

5 TRIPLE

125 OK

The FORTH system is written using a small amount of assembler, with most of the operations being written in FORTH. The system from the book uses what is known as indirect threaded interpretation - the code is more compact than true compilation, but runs faster than byte code. To get a feel for how this works, we can look at how the definition of the TRIPLE function above is compiled. Definitions are stored in a block of memory, the dictionary, which is available to the running FORTH code. A definition takes the form of 16 bit words:

- (1) the name - on the old 16 bit system this was encoded as a length byte together with the first three characters
- (2) link to the previous dictionary entry
- (3) pointer to the handler for the body
- (4) data corresponding the function definition

In the case of TRIPLE we'd have

- (1) the bytes 6TRI
- (2) link to previous entry
- (3) address of in-built assembler stub named COLON
- (4) pointers to the (3) fields of the dictionary entries for DUP, DUP, \*, \* followed by the address of the built-in routine SEMI.

The in-built COLON routine is responsible for working through the items in the (4) block and calling them one-by-one. SEMI is responsible for doing a return back to the caller.

Some base set of primitives in the system is implemented in assembler. These take the same format, but the (3) block points to the start of the (4) block, and this block can contain the actual assembler code for the primitive. This keeps the calling sequence the same. This mechanism around the COLON routine is called the inner interpreter.

There is an outer interpreter that takes care of reading a line from the console and converting it into code to be executed. This outer interpreter simply takes the next chunk of characters and looks them up in the dictionary. Then, depending on the mode, the address is either written to the next free word in the dictionary or the primitive is executed. This allows the : , defining keyword, to be implemented in FORTH itself. : is an immediate keyword - when the outer loop sees it, it executes it immediately, transitioning the system into compile mode after the : handler has read the name of the function from the command line and written the encoded name together with the (2) and (3) fields to the end of the dictionary. Subsequent tokens will be looked up and their addresses will be added to the dictionary. When the end of definition ";" keyword is reached, this is again an immediate keyword which will be executed and which will reset the mode back to the normal execute mode after adding a pointer to the internal SEMI routine to the words for the current definition. Numbers are also handled specially by this outer interpreter - if a lookup fails in the dictionary, the system tries to parse the characters

as a number, in which case a pointer to the built-in constant handler and then the value of the constant are pushed on to the end of the dictionary.

All of the code for manipulating the dictionary can be written in FORTH itself. Primitives (that are in the dictionary and have assembler associated with their execution fields) can expose all of the various internal state to the system itself, and FORTH code can write and read from memory, so the system is very flexible.

The book is a joy to read. The author covers all of the material connected with implementing the system, starting with a high level view and moving down to a complete Z80 assembler implementation. Having the assembler answers any lingering questions that the author hasn't covered in his very good explanations, and the author's enthusiasm is infectious. I may well have to write a simple FORTH system in the near future!

- 
1. [http://www.amazon.co.uk/Threaded-Interpretive-Languages-R-G-Loeliger/dp/007038360X/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1285244881&sr=8-1](http://www.amazon.co.uk/Threaded-Interpretive-Languages-R-G-Loeliger/dp/007038360X/ref=sr_1_1?ie=UTF8&s=books&qid=1285244881&sr=8-1)

#### **4.9.8 It's a Joy (2010-09-30 20:08)**

I've just finished reading [1]The Joy of Clojure which [2]I thought was a really good read.

- 
1. <http://www.manning.com/fogus/>
  2. <http://www.simple-talk.com/community/blogs/clivet/archive/2010/09/30/94827.aspx>

---

## **4.10 October**

### **4.10.1 Another time, another place (2010-10-16 02:41)**

[1]The Time Traveller by Ronald Mallet

The story of an American Professor of Theoretical Physics, who started life in the Bronx and by hard work went on to become a tenured professor. As a result of his father's death when he was ten, Ron Mallet started a life long quest to achieve time travel. In order to do this he became an expert on Einstein's Theory of General Relativity, and may well have discovered how to use lasers to achieve localised frame dragging and hence localised manipulation of time.

Quite an uplifting story of success against adversity, but with no real theoretical content.

1. [http://www.amazon.co.uk/Time-Traveller-Mission-Travel-Reality/dp/0552155756/ref=sr\\_1\\_fkmr0\\_1?ie=UTF8&qid=1287196822&sr=8-1-fkmr0](http://www.amazon.co.uk/Time-Traveller-Mission-Travel-Reality/dp/0552155756/ref=sr_1_fkmr0_1?ie=UTF8&qid=1287196822&sr=8-1-fkmr0)
- 

#### 4.10.2 I wish I knew that (2010-10-16 02:43)

[1]97 Things Every Programmer Should Know by Kevlin Henney

A set of 97 two page essays by a variety of authors, each making some point connected to the development of software. They ranged from ideas such as using a source control system to developing using pair programming. To me, the various articles were quite variable. Some were really interesting, though many repeated the same point, and the shortness of the various articles made some of them not detailed enough to be really engaging.

1. [http://www.amazon.co.uk/Things-Every-Programmer-Should-Know/dp/0596809484/ref=sr\\_1\\_1?ie=UTF8&qid=1287196970&sr=1-1-spell](http://www.amazon.co.uk/Things-Every-Programmer-Should-Know/dp/0596809484/ref=sr_1_1?ie=UTF8&qid=1287196970&sr=1-1-spell)
- 

#### 4.10.3 Hoisted by his own bootstraps (2010-10-16 02:46)

I've been spending some time looking through the sources of the F # compiler. There are various posts around that tell you how to build the compiler from the sources. This requires some effort, but it is really interesting to step through the sources to see what is going on. My particular interest is in how the compiler uses constraint solving to calculate the types of the various functions that the program defines - more on that in another post.

One interesting find was that in the prim-types.fs file, the system uses a construct ( # .. #) which can be used to directly generate IL instructions into the code.

```
> let foo3 (x : int) = ( # "add" x x : int32 #);;
( # "add" x x : int32 #);;
```

```
—^^^^tasciicircum^^ircum^^^^ ^^^^^ciicircum^^^
stdin(2,4): warning FS0042: This construct is deprecated: it is only for use in the F # library
val foo3 : int -> int32
> foo3 20;;
val it : int32 = 40
```

You can also use sequences of IL instructions. For example,

```
let foo33 (x : int) =
( # "ldarg.0 ldarg.0 add" : int #)
```

which Reflector shows compiling to the following IL

```
.method public static [1]int32 [2]foo33([3]int32 x) cil managed
{
    .maxstack 3
    L_0000: ldarg.0
    L_0001: ldarg.0
    L_0002: add
    L_0003: ret
}
```

You can also mix the IL generation into other F # code

```
let foo (x : int) =
if ( # "ceq" x 0 : bool #)
then 0
else ( # "add" x x : int32 #)
```

which Reflector shows as

```
.method public static [4]int32 [5]foo([6]int32 x) cil managed
{
    .maxstack 4
    L_0000: ldarg.0
    L_0001: brtrue.s L_0005
    L_0003: ldc.i4.0
    L_0004: ret
    L_0005: ldarg.0
    L_0006: ldarg.0
    L_0007: add
    L_0008: ret
}
```

1. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Int32>
2. [http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://main/Main/foo33\(Int32\):Int32](http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://main/Main/foo33(Int32):Int32)
3. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Int32>
4. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Int32>
5. [http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://main/Main/foo\(Int32\):Int32](http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://main/Main/foo(Int32):Int32)
6. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:2.0.0.0:b77a5c561934e089/System.Int32>

#### 4.10.4 Taking self-reference to the extreme (2010-10-22 20:24)

[1]Godel's Proof by Nagel and Newman

This is a great introduction to the famous 1931 proof of Godel. It starts off with a good discussion of what it means for a proof system to be consistent, and discusses why mathematicians in the early part of the twentieth century were interested in proving arithmetic (and other systems) consistent. The book then sketches out the ideas behind Godel's proof of the inconsistency of arithmetic. This uses self-reference, or reflection as we would call it in modern programming languages, encoding proofs about arithmetic using numbers and the various methods on proofs using functions that can be defined inside arithmetic.

This book is a great introduction to the proof, covering all of the material in an easy going manner.

1. [http://www.amazon.co.uk/Godels-Proof-Routledge-Classics-Ernest/dp/0415355281/ref=sr\\_1\\_1?ie=UTF8&qid=1287776913&sr=8-1-catcorr](http://www.amazon.co.uk/Godels-Proof-Routledge-Classics-Ernest/dp/0415355281/ref=sr_1_1?ie=UTF8&qid=1287776913&sr=8-1-catcorr)
- 

#### 4.10.5 All together now! (2010-10-31 13:01)

I have just finished reading two academic papers connected to F #. The [1]first is an academic paper on the async tasking model which has been in F # for several years. The async model makes it easy to use lightweight tasks via the computational workflows that F # has built into the language. These make it easy to express algorithms that require a set of steps that need to wait for some external stimulus between the steps. For the rest of this post, we'll use the example:

```
> let task =
- async {
- do printfn "part 1"
- do! Async.Sleep 1000
- do printfn "part 2"
- do! Async.Sleep 1000
- do printfn "part 3"
- };;
val task : Async<unit>
```

This task prints a message, and then sleeps for second using a function that the Async library makes available. This function finishes the current action, potentially allowing the current thread to be released, with a new thread later being allocated to carry out the next step when the wait has finished. We are using the Sleep here to simulate something like an I/O action which might take a long time and where we don't want a thread to be allocated while this wait is happening (for scalability reasons).

The paper discusses some interesting aspects of the design. For example, why Async is implemented as a task factory and not as a constructor for a hot or cold task, and also how cancellation is handled by the model – essentially tasks pass around cancellation tokens, which are polled at regular intervals to see if the current action should be aborted.

Another interesting aspect, and one that is often missed out, is that SynchronizationContexts are captured at the appropriate points and are used to ensure that the actions are carried out on the right thread. It is quite instructive to see this in action.

First, we'll define our own SynchronizationContext. This will only support the Post method which is used to move a callback into the correct context for its execution. Our SynchronizationContext will actually do its work on the threadpool, and will need to set the context of the threadpool thread which is allocated to it to ensure that when the next step executes the context is passed on correctly.

```
> open System.Threading;;
> type MyContext() =
- inherit SynchronizationContext()
- override this.Post (callback, state) =
- printfn "Posted!"
- let _ =
- ThreadPool.QueueUserWorkItem (
- WaitCallback (
- fun _ ->
- let _ = SynchronizationContext.SetSynchronizationContext this
- callback.Invoke(state)))
- ();;
```

We can build an instance of this type

```
> let myContext = MyContext();;
```

And then we can start the Async running inside this context. The "Posted!" output strings show the work items coming back into our context when they are available for running.

```
> let _ = SynchronizationContext.SetSynchronizationContext myContext
- Async.StartImmediate task;;
part 1
> Posted!
part 2
Posted!
part 3
```

You may of course worry that setting the context on the Threadpool thread has had a lasting effect, but it's easy to check by running some code which uses Threadpool threads, that this one of the thread properties that gets reset before the next work item is taken off the ThreadPool work queue.

Computational workflows are a really nice feature of F#. The language contains a small amount of syntax which is de-sugared by the compiler which uses methods of a builder object to produce the final result. For example, the workflow above

```
async { .... }
```

uses the methods on the pre-built instance of AsyncBuilder.

```
> async;;
val it : AsyncBuilder = Microsoft.FSharp.Control.FSharpAsyncBuilder
```

The [2]second paper takes this idea of syntactic language extensions and applies them to pattern matching. These joinads can be used to pattern match over libraries like the joins library, allowing the matching to be expressed concisely within the context of a workflow using the match! operation. These new patterns get translated into code using another set

of methods of a builder object, making the whole mechanism nice and extensible. It will be interesting to see when this new functionality makes its way into the F # language.

1. <http://blogs.msdn.com/b/dsyme/archive/2010/10/21/the-f-asynchronous-programming-model-padl-2010-pre-publication-draft.aspx>
  2. <http://tomasp.net/blog/match-bang-paper.aspx>
- 

## 4.11 November

### 4.11.1 What language do ghosts use? (2010-11-06 10:03)

[1]DSLs in Boo: Domain-specific Languages in .NET by Ayende Rahien

This is a really well written book containing lots of useful information. The author has clearly used DSLs to great effect in various consulting assignments, and has some very strong opinions on how they should be used. In particular, he advocates the use of small DSLs targeted at a specific part of the application, and uses other DSLs to test the original DSL.

Boo is a very good language to use for defining DSLs. It has a very natural syntax with very little ceremony, and the compiler has a very open architecture. The whole compiler works using a pipeline into which the user can easily add new phases. Moreover, there is inbuilt support for macros – methods that are passed an AST and have a chance to modify it before returning it to the compiler for it to use in future processing. Building on top of an existing language means that the DSL implementer does not need to write a parser, and can use existing IDE tools for the language to add IDE support for the DSL.

The book gives good advice on writing DSLs. The implementation needs to consist of an object model with an engine that processes this model, together with an API for configuring the model and triggering the engine. The DSL sits on top of this and is translated into calls into this API. Keeping this separation makes it easy to test and support revision of the DSL in the future.

The author provides a number of tools for making it easy to work with DSLs in Boo. In particular he offers modules that can be inserted into the compiler pipeline to make the language more user friendly. The Boo DSL is often interpreted as a class definition, and one of these additional modules automatically changes the base class of the defined class so it sits on top of the implementation code. This makes it trivial to call into the API layer, and therefore makes it easy to write a DSL.

The book is full of useful hints and tips, both on the scope of the DSL and how to keep things maintainable over multiple versions. A really good read!

1. [http://www.amazon.co.uk/DSLs-Boo-Domain-Specific-Languages/dp/1933988606/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1289031492&sr=8-1](http://www.amazon.co.uk/DSLs-Boo-Domain-Specific-Languages/dp/1933988606/ref=sr_1_1?ie=UTF8&s=books&qid=1289031492&sr=8-1)
-

#### 4.11.2 That's a library Hercules would have been proud of (2010-11-14 15:34)

I've just finished reading an early draft of [1]Patterns of Parallel Programming (in .net) which was a really good introduction to the TPL, and in particular the Task type. Now, after the recent PDC, it is going to turn out that the Task class has become a vital part of the C # language – but more on that in a subsequent post.

A Task is a really neat abstraction for a block of work that will execute at some point and provide a value in the future. One can think of a Task as a future, as in the following example, where the task is effectively a placeholder for a potential integer value at some later stage.

```
Console.WriteLine("Running on thread {0 }", Thread.CurrentThread.ManagedThreadId);
Task<int> task1 = Task.Factory.StartNew<int>(() =>
{
    Thread.Sleep(3000);
    Console.WriteLine("Hello from {0 }", Thread.CurrentThread.ManagedThreadId);
    return 20;
});
Console.WriteLine("Final value is {0 }", task1.Result);
```

The use of Result causes the current thread to wait for the task to complete, at which point the Result property is available for fetching the computed result.

The Task traps any exception that was raised, making it available via properties on the Task, and calls to Result will also rethrow the exception.

```
Console.WriteLine("Running on thread {0 }", Thread.CurrentThread.ManagedThreadId);
Task<int> task1 = Task.Factory.StartNew<int>(() =>
{
    Thread.Sleep(3000);
    throw new Exception("Bang");
});
try
{
    Console.WriteLine("Final value is {0 }", task1.Result);
}
catch (Exception ex)
{
    Console.WriteLine("Exception caught");
}
```

Tasks have all sorts of interesting properties which allow an outside party to see the status of the task – whether it has started running, and if the contained computation has finished executing or has raised an exception. It is also possible to cancel tasks. This uses a polling model and uses a CancellationToken(Source) type which is passed into the Task when it is created. At certain points, the library itself will poll the token for cancellation, and user code inside the Task is expected to do this too at regular intervals (by making a library API call). When cancellation is desired, the computation can throw an OperationCanceledException which will be caught and will set the Task status appropriately.

Most impressively, there is a set of combinator for combining tasks. In particular the ContinueWith combinator allows Tasks to depend on the results of other tasks in very complicated data flow networks.

```

Task<int> task1 = new Task<int>(() =>
{
    Thread.Sleep(3000);
    Console.WriteLine("part 1");
    return 20;
});
Task<int> task2 = task1.ContinueWith<int>(t1 =>
{
    var result1 = t1.Result;
    return result1 * 10;
});
task1.Start();
Console.WriteLine(task2.Result);

```

Since the language now supports closures, the `ContinueWith` delegate can be used to implement a continuation of the current computation, and we'll see later how this is used by the new C # `async` construct.

The paper looks through the various patterns that the TPL supports. It covers the Parallel For and Foreach loops, showing their PLINQ equivalents, and then goes on to discuss the various patterns of parallel tasks, from striping across the data to pipelining and scatter-gather.

It was a good introduction to the .NET 4 support for parallel programming and is well worth a read.

1. <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=86B3D32B-AD26-4BB8-A3AE-C1637026C3EE>

---

#### **4.11.3 Give me some time (2010-11-14 15:40)**

There's a been a lot of buzz around about the new `async` construct for C # 5. This is explained quite well in the [1]Channel9 interview with Mads Torgersen with two other really good presentations on the background material from [2]Stephen Toub and [3]Lucian Wischik.

To me the key idea is that this is all about allowing a thread to be shared while the initial method is waiting for something else to finish. This contrasts with other asynchrony mechanisms where the emphasis is to get a threadpool serving lots of requests efficiently. With the `async` construct there is an well known use-case of allowing the GUI thread to be used for what used to relegated to background processing. The old way of using a `BackgroundWorker` introduced difficulties with synchronising the background and the foreground processing.

Moreover, the `async` modifier allows the compiler to take on the work of generating continuations. Instead of requiring this to be expressed at the user program level, making the code hard to read, we get code that is far clearer. We've seen this kind of idea before with the introduction of iterators into C # 2. The co-routine, required of an iterator implementation, can be automatically generated by the compiler, instead of requiring the user to implement the state machine.

As in the iterator case, there is a pattern behind the implementation of a user object that can be await'ed. Adding the `async` modifier to a method, which must then return a `Task` or `void`,

allows that method to use the await keyword. This await is implemented along the lines of yield return in C# (though it can be used in expressions too). The compiler generates code which records the state point at which the await occurs, and the compiler subscribes a delegate which will be notified when the next state transition needs to be made. This latter stage is handled by the await'ed object having to support the GetWaiter/BEGINWait/ENDWait pattern.

In the following example, where we'd use a top level call of the form

```
Task<int> worker = DoWork();
int result = worker.Result;
```

the DoWork method will process each word using a worker object and it will wait for the result using await. The use of await will allow the processor to exit from the DoWork method, and the generated code will have registered a callback using the BEGINWait. This callback will transition the state machine implemented inside the DoWork method allowing the next call to be made. The code inside the DoWork method looks fairly natural without the need for the user to explicitly write the continuations.

```
static async Task<int> DoWork()
{
    int currentLength = 0;
    foreach(string word in new [] { "It", "was", "the", "best", "of", "times" })
    {
        Console.WriteLine("Processing '{0}' on thread {1}",
            word, Thread.CurrentThread.ManagedThreadId);
        currentLength += await new GetLength(word);
        if (currentLength > 10)
            return currentLength;
    }
    return -1;
}
```

We write the GetLength code to implement the GetWaiter pattern, and pretend that the actual computation takes some time to execute by using a Thread.Sleep.

```
class GetLength
{
    readonly string m_Input;
    public GetLength(string input)
    {
        m_Input = input;
    }
    public MyAwaiter GetAwaiter()
    {
        return new MyAwaiter(m_Input);
    }
    internal class MyAwaiter
    {
        readonly int m_Result;
        public MyAwaiter(string input)
        {
```

```

m _Result = input.Length;
}

public bool BeginAwait(Action resumption)
{
    ThreadPool.QueueUserWorkItem( _ =>
{
    Thread.Sleep(2000);
    resumption();
});
return true;
}

public int EndAwait()
{
    return m _Result;
}
}
}
}

```

There are several things to mention. The BeginWait can return false to say that the task has already finished, allowing a potential efficiency gain. Also, the above code makes no use of SynchronizationContexts which are what should be used to guarantee the execution context of subsequent code. In this example, the first part of the code will happen on the main thread, and the rest will happen on various thread pool threads. In a real implementation we would also have to worry about catching exceptions and re-raising them in the EndAwait.

Generally the code that is await'ed is going to have generated a Task instead of a user object. This class has appropriate extensions method defined on it in the CTP, which support the pattern. We can run the following code to demonstrate this.

```

Task task = new Task(() => Console.WriteLine("Bang"));
var awainerObject = task.GetAwaiter();
awainerObject.BeginAwait(() => Console.WriteLine("Resume"));
task.Start();
task.Wait();

```

In the past, people such Jeff Richter and the CCR team have used the iterator pattern to try to get natural looking code which can give up the thread when it is waiting. It's nice to see that with a simple language extension we can get the benefits in the language itself, just by adding one modifier and a new keyword, and in a way that allows user classes to extend the facility.

1. <http://channel9.msdn.com/Shows/Going+Deep/Mads-Torgersen-Inside-C-Async>
2. <http://channel9.msdn.com/Shows/Going+Deep/Stephen-Toub-Task-Based-Asynchrony-with-Async>
3. <http://channel9.msdn.com/Shows/Going+Deep/Lucian-Wischik-Inside-VBNET-Async>

#### **4.11.4 That's the way you do it (2010-11-23 18:58)**

I've recently given two talks at work. [1]The first was on the new C # async modifier and the await keyword. [2]The second was on using the Red Gate memory profiler to diagnose memory management issues.

1. <http://cid-3f21df299c355e7f.office.live.com/view.aspx/Public/The%20next%20C.pptx>
  2. <http://cid-3f21df299c355e7f.office.live.com/view.aspx/Public/VisualisingMemoryProblems.pptx>
- 

#### **4.11.5 It's all connected you know (2010-11-23 19:03)**

[1]Introduction To Graph Theory by Robin J. Wilson

I've recently had the opportunity to do some work on the [2]Ants Memory profiler code base. This has involved looking into some of the algorithms that the product uses for calculating the strongly connected components and various shortest distance algorithms.

Despite really enjoying discrete mathematics and combinatorics, I didn't do the Graph Theory course when I was at university. This book is a really interesting introduction to basic graph theory. It covers all of the basic theorems, with plenty of examples and exercises for getting the reader familiar with the material, and was well worth a read.

1. [http://www.amazon.co.uk/Introduction-Graph-Theory-Robin-Wilson/dp/027372889X/ref=sr\\_1\\_1?ie=UTF8&qid=1290361155&sr=8-1](http://www.amazon.co.uk/Introduction-Graph-Theory-Robin-Wilson/dp/027372889X/ref=sr_1_1?ie=UTF8&qid=1290361155&sr=8-1)
  2. [http://www.red-gate.com/products/ants\\_memory\\_profiler/index.htm](http://www.red-gate.com/products/ants_memory_profiler/index.htm)
- 

#### **4.11.6 My evaluation is good (2010-11-27 21:28)**

This week one of my work colleagues asked if I had noticed the resurgence of interest in supercompilation. I'd come across partial evaluation in the past... the idea that you can evaluate large chunks of some programs at runtime, basically by using some kind of evaluator that works on the chunks of program that depend on fixed data. There were many applications of this technique to Scheme in the past. I'm also subscribed to a number of Haskell mailing lists, where I've seen the work of Mitchell mentioned repeatedly.

At the weekend, I therefore downloaded the paper [1]Supercompilation by evaluation from Simon Peyton Jones' recent papers collection. This paper is a really interesting read. The two authors have developed a very effective supercompiler. Most interesting to me was that they have derived it from an initial operational semantics for a language, rather than just taking a load of ad hoc techniques and putting them together. They use memoization to prevent the

evaluator doing lots of repeated work, and go into some detail of the well order that they use to ensure that the supercompiler terminates.

Their supercompiler also deals with let bound recursive functions, something that previous supercompilers have found difficult. The key here is that when the split phase is carried out to focus the compiler's attention on a sub-term, certain state needs to be passed down the recursive branch. You must be careful to avoid passing too little data into the recursive call, or potential optimisation is lost, and too much information can lead to invalid optimisations. The paper does a good job of explaining the difficulties.

The authors postulate that their techniques could be applied to call-by-value languages and it will be really interesting to see if this claim is true. In a language like C #, side effects happen more frequently and I wonder if this would remove some of the potential gains of partial evaluation. I'm also interested in a comparison with modern JIT architectures, where the system can find the hot paths and spend effort optimising those paths, in contrast to whole program optimisation techniques.

I also downloaded the paper [2]Seq no more which discusses some improvements to the GHC implementation which supports the par and pseq functions that have been added to allow parallelism.

par x y suggests to the compiler that a spark (a GHC runtime task) should be launched to start the evaluation of x while the computation continues with the evaluation of y which is the result of the par function.

pseq ensures that x is evaluated before the system continues to evaluate y, and adds evaluation order into a pure language.

The paper discusses some of the implications of the spark pool, in particular whether the elements in the pool should be taken as strong or weak roots by the runtime system, and how the system should deal with fizzled sparks – sparks representing computations that have already been evaluated by someone else. In a call-by-need language these fizzled sparks are an administrative overhead. By doing some reworking, the resulting system can treat sparks as weak roots, which helps to prevent the processors becoming overwhelmed by tasks.

The main part of the paper is concerned with an improved definition of strategies. A strategy is a specification of how something should be evaluated with respect to the use of additional sparks, and the paper ends up with something that looks like an embedded DSL for specifying these strategies. A system library can provide a number specifications that work with many built-in datatypes. For example, a strategy for evaluating lists that will evaluate the first n elements in parallel while keeping the evaluation of the rest of the elements lazy.

Both papers are full of clever ideas and are well worth a read. Like all Peyton Jones papers there are good practical results all based on solid theory.

1. <http://research.microsoft.com/en-us/um/people/simonpj/papers/supercompilation/>

2. <http://www.haskell.org/~simonmar/papers/strategies.pdf>

---

Amin (2010-11-29 06:29:30)

Pretty interesting write about the super-compilation paper. I have been working on Partial evaluation of Java programs for a year now. Maybe we can discuss it if you are interested.

clivetong (2010-11-30 08:38:42)

That sounds really interesting. I'll send you some email.

#### 4.11.7 Check to see if that's my type of thing (2010-11-27 21:28)

I got asked at work how to get type inference in the following situation. Here we'd like to write `Foo.Print(x);` but end up having to write `Foo<int>.Print(x);` when calling the static method of the class below.

```
class Foo<T>
{
    public static void Print(T x)
    {
        Console.WriteLine(x.ToString());
    }
}
```

My answer was to use method level generic parameters, so the class gets rewritten to

```
static class Foo // static optional here
{
    public static void Print<T>(T x)
    {
        Console.WriteLine(x.ToString());
    }
}
```

but I have to confess that I'd never really noticed before that there is no type inference in the first situation. The whole section in the C# specification on type inference (7.4.2) talks only of the generic parameters at the method level and the two phases (with phase two repeated) that lead to the inference of the types in order to determine the applicable methods.

Using method level generic parameters is a good technique to use to avoid having to decorate the static method with types and is applicable in many situations.

---

### 4.12 December

#### 4.12.1 That's a lot more than 6 items (2010-12-04 14:08)

There are some good recorded sessions by Mark Russinovich which were presented at the [1]recent PDC concerning the mysteries of Windows Memory Management. The first session

concerns virtual memory and the second tackles physical memory and a process's working set. There is a strong practical element, with the presenter using several of the sysinternals tools to demonstrate the concepts. I have, of course, read [2]Windows Internals, but there's nothing like a demonstration to bring some concepts into focus.

Other recent interesting videos include the recent [3]Silverlight Firestarter Keynote and a talk by [4]Jon Skeet on the mistakes in C#. This latter talk has an interesting section on static interfaces and some controversial ideas on sealing classes.

1. <http://player.microsoftpdc.com/schedule/sessions>
  2. [http://www.amazon.co.uk/Windows-Internals-PRO-Developer-Mark-Russinovich/dp/0735625301/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1291401092&sr=1-1](http://www.amazon.co.uk/Windows-Internals-PRO-Developer-Mark-Russinovich/dp/0735625301/ref=sr_1_1?s=books&ie=UTF8&qid=1291401092&sr=1-1)
  3. <http://channel9.msdn.com/Series/Silverlight-Firestarter/Silverlight-Firestarter-2010-Keynote-with-Scott-Guthrie>
  4. <http://oredev.org/2010/sessions/c-s-greatest-mistakes>
- 

#### 4.12.2 I can live within those constraints (2010-12-04 14:41)

I've just spent a little time reading the [1]F# specification.

I must confess that I hadn't come across the upcast and downcast functions before, which can be used to change the type associated with a value.

```
> let x : obj = upcast "hello";;
val x : obj = "hello"
> let y : string = downcast x;;
val y : string = "hello"
```

My main reason for reading the specification was to get a better mental model of the type system that the language uses. In the past I was part of a team that implemented an ML compiler. This used a variant of the Hindley-Milner type checking algorithm. In contrast, the F# system type checks types using constraint solving, and there are a lot more types of constraint that need to be checked.

```
> type First() = class end;;
> type Second() = class inherit First() end;;
> let f x : 'T when 'T :> First = x;;
val f : 'T -> 'T when 'T :> First
```

The :> constraint demands a subtype relationship between the argument and the target type.

```
> f(First());;
val it : First = FSI _0005+First
> f(Second());;
val it : Second = FSI _0006+Second
```

and has a short form using the hash symbol.

```
> let f x : #First = x;;
val f : 'a -> 'a when 'a :> First
```

The member constraint can be used to ensure that a type supports a certain operation. It is used in conjunction with type variables that cannot be generalised and which are written in the form  $\wedge T$ .

```
> let f x :  $\wedge a = x;;$ 
let f x :  $\wedge a = x;;$ 
_____^
```

```
stdin(14,12): warning FS0064: This construct causes code to be less generic than
indicated by the type annotations. The type variable 'a' has been constrained to
be type 'obj'.
val f : obj -> obj
```

Because of this lack of generalisation, the  $\wedge$  types are most often used in the context of inline functions.

```
> let inline doit (x:  $\wedge a) = (^a : (static member speak: int -> string) (20));;$ 
val inline doit :
 $\wedge a -> string$  when  $\wedge a : (static member speak : int -> string)$ 
```

The previous function ensures that the argument is of a type that has a static member called speak.

```
> type Boo() = static member speak x = "Called on " + x.ToString();;
> doit(Boo());;
val it : string = "Called on 20"
```

It took me quite a while to figure out how to call a non-static member, mainly because I failed to notice that a tuple was required to apply the member.

```
> let inline doit (x:  $\wedge a) = (^a : (member speak: int -> string) (x,20));;$ 
val inline doit :
 $\wedge a -> string$  when  $\wedge a : (member speak :  $\wedge a * int -> string)$$ 
```

```
> type Test3() = class member x.speak(y) = "Hello" end;;
> doit(Test3());;
val it : string = "Hello"
```

F # is great in that it discourages the use of null as a member of lots of datatypes. We can constrain a function to take an argument that is of a type that has null as a member:

```
> let f x : 'a when 'a : null = x;;
```

This function works on standard .NET classes like string, which have null as a member.

```
> f "hello";;
val it : string = "hello"
```

But not on the typical F # datatypes which do not have a null member.

```
> type Foo = A | B;;
> f A;;
f A;;
_ ^
```

```
stdin(20,3): error FS0001: The type 'Foo' does not have 'null' as a proper value
```

We can constrain on types that have a particular constructor:

```
> let f x : 'a when 'a : (new : unit -> 'a) = x;;
```

And much like C # generics we can demand a valuetype

```
> let f x : 'a when 'a : struct = x;;
> f "hello";;
f "hello";;
```

-^ ^ ^ ^ asciicircum ^ ^

stdin(28,3): error FS0001: A generic construct requires that the type 'string' is a CLI or F # struct type

```
> f System.DateTime.Now;;
val it : System.DateTime = 03/12/2010 18:55:33 {Date = 03/12/2010 00:00:00;
```

or a reference type.

```
> let f x : 'a when 'a : not struct = x;;
> f "hello";;
val it : string = "hello"
> f System.DateTime.Now;;
f System.DateTime.Now;;
```

-^ ^ ^ ^ asciicircum ^ ^ ^ ^ rcum ^ ^ ^ ^ textasciicircum ^ ^ ^ ^ iicircum ^

stdin(33,3): error FS0001: A generic construct requires that the type 'System.DateTime' have reference semantics, but it does not, i.e. it is a struct

One can demand types that are unmanaged or delegate types.

```
> let f x : 'a when 'a : unmanaged = x;;
> f "hello";;
f "hello";;
```

-^ ^ ^ ^ asciicircum ^ ^

stdin(39,3): error FS0001: A generic construct requires that the type 'string' is an unmanaged type

```
> let handler = Microsoft.Win32.TimerElapsedEventHandler(fun o ev -> ());
val handler : Microsoft.Win32.TimerElapsedEventHandler
> let f x : 'a when 'a : delegate<Microsoft.Win32.TimerElapsedEventArgs,unit> = x;;
val f : 'a -> 'a when 'a : delegate<Microsoft.Win32.TimerElapsedEventArgs,unit>
> f handler;;
val it : Microsoft.Win32.TimerElapsedEventHandler = ...
```

For interoperability there's also a way to specify the underlying type for an enumeration.

```
> let f x : 'T when 'T : enum<int> = x;;
val f : 'T -> 'T when 'T : enum<int>
> f(System.Diagnostics.TraceOptions.DateTime);;
val it : System.Diagnostics.TraceOptions = DateTime
```

Most importantly, you can demand types that support equality or comparison.

```
> [<NoEquality;NoComparison>]
- type Foo() =
- class end;;
```

```
> let f x : 'a when 'a : equality = x;;
val f : 'a -> 'a when 'a : equality

> f 2;;
val it : int = 2
> f (Foo());;
f (Foo());;
```

—^<sup>^</sup><sup>^</sup><sup>^</sup>tasciicircum

stdin(91,4): error FS0001: The type 'Foo' does not support the 'equality' constraint because it has the 'NoEquality' attribute

```
> let f x : 'a when 'a : comparison = x;;
val f : 'a -> 'a when 'a : comparison
> let t = new System.GenericUriParser(System.GenericUriParserOptions.NoQuery);;
> f t;;
f t;;
_^
```

stdin(69,3): error FS0001: The type 'System.GenericUriParser' does not support the 'comparison' constraint. For example, it does not support the 'System.IComparable' interface

It is also worth pointing out that you can have multiple constraints.

```
> let f x : 'a when 'a : not struct and 'a : comparison = x;;
val f : 'a -> 'a when 'a : not struct and 'a : comparison
```

The system takes the various constraints on the type variable, and the constraints that are generated from the structure of the program and attempts to find a solution to them.

For example, suppose that we know  $f : \text{string} * \text{int} \rightarrow \text{int}$  and we need to check the following function definition.

```
let g x y = f(x, f(x, 10))
```

We'd generate new type variables ty1, ty2 and ty3 and the constraints

$g : ty1 \rightarrow ty2 \rightarrow ty3$

The inner application would involve a new variable ty4 and the constraints

$ty4 : \text{int}, ty1 : \text{string}$

And the outer application would give

$ty1 : \text{string}, ty4 : \text{int}, ty3 : \text{int}$

Solving the constraints would give the solution with  $ty1 = \text{string}$ ,  $ty3 = \text{int}$  and  $ty2$  as anything.

We could therefore infer the type of  $g$  as

```
val g : string -> 'a -> int
```

1. <http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/spec.pdf>

---

#### 4.12.3 Bless you! (2010-12-19 18:06)

[1]The Soul Of a New Machine by Tracy Kidder

I remember reading this book a long time ago when I was young. It's the story of the development of the Eagle, a Skunkworks project at Data General which finished in 1980. It chronicles how the team were successful in producing a new 32 bit mini-computer as a competitor to the DEC VAX.

I thought it was a fairly dry, but interesting read. [2]Wikipedia has a good review of it.

1. [http://www.amazon.co.uk/Soul-New-Machine-Tracy-Kidder/dp/0316491977/ref=sr\\_1\\_2?s=books&ie=UTF8&qid=1292779693&sr=1-2](http://www.amazon.co.uk/Soul-New-Machine-Tracy-Kidder/dp/0316491977/ref=sr_1_2?s=books&ie=UTF8&qid=1292779693&sr=1-2)
  2. [http://en.wikipedia.org/wiki/The\\_Soul\\_of\\_a\\_New\\_Machine](http://en.wikipedia.org/wiki/The_Soul_of_a_New_Machine)
- 

#### **4.12.4 Thanks for the purity (2010-12-19 18:08)**

I'm not fully sure what it is about the Haskell language that I find fascinating. One thing is certainly the purity. The call-by-need evaluation means that one can take very mathematical looking definitions and they will correctly evaluate even when applied to unbounded data structures. Additionally, the lack of side-effects make it really easy to prove properties about the programs when they are executed. Another aspect of the language is its simplicity, though admittedly this is tainted in places by the support for eagerness in areas like the pattern matching.

There are a couple of really interesting short talks connected with work on getting GHC working well on multi-core computers - [1]Scheduling lazy evaluation on multi-core and [2]Runtime support for multi-core Haskell.

Haskell is also a ready arena for the current spate of work on super-compilation. There's a an interesting talk on this area: [3]Re-thinking supercompilation

1. <http://vimeo.com/15573590>
  2. <http://vimeo.com/6654930>
  3. <http://vimeo.com/15833948>
- 

#### **4.12.5 Very enthusiastic (2010-12-19 18:10)**

I'm be up front about it... I love the idea of formally verifying the semantics of a program.

When I was a little younger, I was really fascinated by the idea of using standard mathematical techniques to prove that a program did what it claimed to do. Moreover, the idea of using constructive mathematical logics to extract computational meaning from proofs, seemed almost magical to me. This latter area is summarised nicely in a recent paper by [1]Robert Constable. I started working on a Phd in this area and spent some time working with Martin-Lof type theory

inside [2]Isabelle. At one point, I was offered the chance to prove the correctness of a washing machine controller as a thesis topic.

It was therefore interesting to see the recent channel9 interview with Chris Hawblitzel on [3]Verve, a verified kernel for a single processor operating system. This is really interesting work which is covered in the [4]paper. The idea is that a small amount of x86 machine code (the nucleus) is verified using Hoare logic. This is then used to execute C # programs which have been compiled down to typed assembled language. Typed assembly language is covered in [5]this paper, and the author goes into a lot of detail about verifying two garbage collection algorithms in [6]this paper. The last paper is really interesting in that the authors didn't use an interactive theorem prover, but used a system that derived verification conditions from a high level language (BoogiePL) representation of x86 instructions. They needed to provide hints to the theorem prover about which definitions with universal quantifiers needed expanding (the T function of the paper), and used regions to convert global theorems into local theorems that the prover could solve. After the correctness proof, the x86 code could be automatically generated.

The system doesn't yet support more interesting constructs such as exceptions and multiple processors, but this is great foundation work that will surely be improved in the future.

1. <http://www.srccf.ucam.org/principia/files/rc.pdf>
  2. <http://www.cl.cam.ac.uk/research/hvg/Isabelle/>
  3. <http://channel9.msdn.com/Shows/Going+Deep/Verve-A-Type-Safe-Operating-System>
  4. <http://research.microsoft.com/apps/pubs/?id=122884>
  5. <http://research.microsoft.com/apps/pubs/default.aspx?id=55983>
  6. <http://research.microsoft.com/apps/pubs/default.aspx?id=74285>
- 

#### **4.12.6 Pause a second would you? (2010-12-27 20:21)**

There's a great interview with Gil Tene covering the work Azul have done on a [1]pauseless garbage collector. Incremental collectors have been around for a long time, but the problem has always been the cost of implementing a read barrier to prevent the mutator from seeing objects that have been moved. A read barrier is used to trap access to these moved objects, at which point the system can change the reference to point to the new place to where the object was relocated.

In past implementations, read barriers have either been implemented in software which has been prohibitively expensive by requiring either an extra indirection during object access or a test during read operations, or by using virtual memory support which proved to be too expensive due to the overhead required by the operating system to handle a trap and branch into a user handler and by virtual memory designs that did not expect page protection to be modified at a high rate (to add the barrier to a page and then remove the barrier after the page is fixed up).

There is some good discussion of the interview [2]here which also contains a pointer to a paper covering the [3]work.

1. [http://www.artima.com/lejava/articles/azul\\_pauseless\\_gc.html](http://www.artima.com/lejava/articles/azul_pauseless_gc.html)
  2. <http://news.ycombinator.com/item?id=2022723>
  3. [https://www.usenix.org/events/vee05/full\\_papers/p46-click.pdf](https://www.usenix.org/events/vee05/full_papers/p46-click.pdf)
- 

#### 4.12.7 This is the first iteration (2010-12-29 20:53)

I recently got to [1]blog about work that I've been doing as part of my day job - adding some functionality to Reflector.

1. <http://www.simple-talk.com/community/blogs/clivet/archive/2010/12/16/96199.aspx>
- 

#### 4.12.8 On your Next action, please follow these guidelines (2010-12-29 20:55)

I recently got around to reading the [1]Rx Design Guidelines. This is a fairly small document that outlines advice for using the Rx library.

The early documents on Rx didn't seem to offer much advice on the scheduling of the actions that are to be carried out when the methods of IObservable are called. When subscribing to a stream of events, the context is often important if you want to interact with thread bound items such as Windows controls.

This document and the later releases of Rx cover the concurrency aspect in much more detail. There is now an IScheduler interface that allows the user to control when and in which context the actions are executed (in addition to the ObserveOn and Synchronize methods), and the user has a fair amount of control over the scheduling using one of many pre-defined Schedulers. There is also a Scheduler that uses a concept of virtual time, which makes it easy to write unit tests where events happen at particular instants.

I was also interested to see that a form of the Join calculus is supported by the Rx library. We can use Subject objects to represent the channels of the Join calculus and can then define patterns that look very much the usual Counter example.

```
var counter = new Subject<int>();
var inc = new Subject<int>();
var get = new Subject<IObserver<int>>();

var channelObserver =
Observable.Join(
counter.And(inc).Then((current, result) =>
{
    counter.OnNext(current+result);
    return new Unit();
})),
```

```

counter.And(get).Then((current, channel) =>
{
    channel.OnNext(current);
    counter.OnNext(current);
    return new Unit();
}));
```

This observer needs to have a subscription to make it active, and then we can send it the various messages as in the standard example. If we were doing this for real, we would, of course, need to dispose the observers at the relevant points.

```

channelObserver.Subscribe();
counter.OnNext(5);
inc.OnNext(3);
inc.OnNext(5);

var counterValue = new Subject<int>();
counterValue.Subscribe(x => Console.WriteLine(x));
get.OnNext(counterValue);

inc.OnNext(3);
inc.OnNext(5);
get.OnNext(counterValue);
```

It's quite exciting to see how rapidly this library is developing. The concurrency side of things is really different from the FRP (Functional Reactive Programming) examples of old, where everything was single threaded and hence context safe, but the concurrency side of things is being worked into the Rx library. This library is certainly worth watching as it develops in the future.

1. <http://go.microsoft.com/fwlink/?LinkID=205219>

---

#### **4.12.9 That's not the value you showed me before (2010-12-29 20:57)**

I've just been reading a very interesting MSR paper on [1]Verifying Compiler Transforms For Concurrent Programs. The various memory models at both the hardware level on x86, and at the compiler level for C # seem very underspecified. This paper documents work on formalising the memory models, and using the formalisms to check that programs show the same effects after compiler transforms have been applied.

The work is great in two ways. First they have found bugs in the .NET x86 JIT which is obviously practically useful. The work also gives a great formal definition of the effects of the memory model in terms of how it modifies the traces of concurrent threads when they are merged to produce an effective trace. This makes it easier to understand the effects than the various wordy descriptions one finds in various books and blog articles. Of course, the real problem is that hardware level memory model is not formally specified by Intel - hence the authors have had to infer the formal model and then check that it exhibits various effects of the memory model that are specified in Intel design documents.

Of course, most C # programmers don't care too much about the memory model. They will be writing code that is generally single threaded in its access of data (using lock statements). [2]Joe Duffy's blog has a great write-up of how he thinks things are going in the future. He points out that weaker memory models than the x86, such as those for the ARM chip, are going to become even more important in the future, and the average programmer is going to use other techniques to get thread safety in their code.

Nevertheless, I would like to understand memory models better, and this paper is a great help in that process.

1. <http://research.microsoft.com/pubs/76524/tr-2008-171-latest-03-11-09.pdf>
  2. <http://www.bluebytesoftware.com/blog/2010/12/04/SayonaraVolatile.aspx>
-

# 2011

## 5.1 January

### 5.1.1 Not just William (2011-01-03 12:46)

Over the holiday I thought I ought to do some maintenance on our family laptop. The disc has two partitions, and the C: drive is almost full. It was easy to move my c:\users to the D: drive and then put a link in place to keep old links working (using MKLINK /D). Everything seemed fine, except that in IE (and not Firefox), Flash areas were blank instead of containing the background picture and the play button, and using “Save as” from the context menu kept failing. I tried uninstalling Flash and then reinstalling without any improvement.

What did get things working again? Uninstalling the Google Toolbar!

---

### 5.1.2 Phone me! (2011-01-03 12:48)

[1]Programming Windows Phone 7 by Charles Petzold

Over the holiday period I decided that it was time to get to grips with Windows Phone 7 programming using Silverlight, and this free 1000 page book turned out to be just the thing to read. The book is long, and contains three sections – one for Silverlight, one for XNA and an initial section that covers the more general idea behind the phone. Hence, if you are just interested in Silverlight then you need only read roughly 700 pages.

The book is really good. I’ve skipped through other books on WPF, but have never found them as easy to read as this book. The chapters introduce the relevant concepts, mainly by way of worked examples that start simple and build up. Like the Petzold Windows Forms book, there is a great emphasis on writing the code (and XAML) without using any design tools in order to really understand what is going on. The coverage of dependency objects and layout was the best that I’ve read... indeed, several times I had questions that the author then answered within a couple of pages. Better still, the author often asks questions at the start of a chapter to get you thinking, and then answers them before the chapter finishes.

Petzold has made a zip file of the examples available, and playing with them is a great way to get a better understanding of the material, and I can’t stress enough what a great set of

worked examples the author has assembled. The only unanswered questions I was left with were how the branch of Silverlight 3 that the phone uses differs from the desktop Silverlight 3 and how this differs from the current Silverlight 4, and a better comparison between the VisualStateManager that Silverlight uses and the triggers that are offered by WPF.

A really good book and free too.

1. <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=bb8f5eb6-8214-4387-bf02-f78f314a74eb>
- 

### 5.1.3 One at a time please! (2011-01-10 22:49)

I spent some more time at the weekend reading up about some of the multi-core related work that Microsoft are doing.

First I watched a [1]video about the Orleans research project and read the [2]associated paper. Orleans is basically what, in the old days, would have been called an application server along the lines of COM+/EJB/CORBA, but with some novel ideas. What we used to call components are now known as grains. Grains are single threaded in their execution – they take part in one orchestrated activity at a time, and can only be reused once this has finished. This is one interesting angle in the recent search for scalability... programming can get easier with threading happening at the application server level rather than at the level of user objects. Grains can be persisted, and they can call other grains via object references that they are passed. Multiple instances of a particular grain can be created by the infrastructure, but only a single instance can take part in a given activity – this may occasionally mean that transactions need to be rolled back if two instances of a given grain are inadvertently pulled into the same activity.

Transactions can be aborted and restarted, and the infrastructure is responsible for controlling the amount of concurrency in order to meet service level requirements.

This is all implemented in managed code, and the use of this actor model really simplifies the programming. At the user code level, the code sees the result of calls to a grain as a future. These futures are built on top of the TPL, and they can be linked together like Tasks in the TPL – using ContinueWith and various Wait operations to link them together.

The Task Parallel Library is a great .NET library for handling concurrent tasks, and there is now a dataflow library that is written on top of it. There's a great overview document on [3]TPL dataflow. This library can be used for applications which can be structured as a dataflow pipeline. You can take pipelines which take values and process them before pushing them out to other consumers. The consumers can take multiple input values from different inputs, and there is a two-phase commit protocol for allowing this to happen. There are two .NET interfaces, one for sources and one for targets, which the library user can implement, and there are a number of pre-supplied components which look really useful. There are some similarities between this and the Rx library, though the focus of the two libraries is a little different, the latter being aimed at enabling LINQ manipulation of event streams rather than pipelined data processing.

The first release of TPL Dataflow was part of the recent Async CTP. There are some great [4]power point slides on the design of async functionality which I hadn't noticed before.

Talking of scalability, there's also [5]an interview with Gil Tene concerning Azul's highly scalable Java technology, including their pauseless garbage collector.

1. <http://channel9.msdn.com/Shows/Going+Deep/Project-Orleans-A-Cloud-Computing-Framework>
  2. <http://research.microsoft.com/pubs/141999/pldi%2011%20submission%20public.pdf>
  3. <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=d5b3e1f8-c672-48e8-baf8-94f05b431f5c&displaylang=en>
  4. <http://blogs.msdn.com/b/lucian/archive/2010/10/29/a-technical-walk-through-all-of-the-async-ctp.aspx>
  5. <http://www.infoq.com/interviews/gil-tene-azul-zing>
- 

#### **5.1.4 I did know that (2011-01-25 21:33)**

[1]100 essential things you didn't know you didn't know by John D. Barrow

As is usual with books by John Barrow, this was an excellent read. It contains 100 two to four page descriptions of various interesting facts from physics and mathematics, ranging from the reasons that flash fires happen in custard factories to the mathematics behind high jumping to the maximum efficiency of a wind turbine (59 % Betz's law). There's also an entertaining story of a football match when one team ended up defending their opponent's goal.

One story I particularly enjoyed was an example of when omniscience is a disadvantage – you'll have to read the book to find out, though it revolves around a game of chicken. A really good read.

1. [http://www.amazon.co.uk/100-Essential-Things-Didnt-Know/dp/1847920039/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1295802128&sr=8-1](http://www.amazon.co.uk/100-Essential-Things-Didnt-Know/dp/1847920039/ref=sr_1_1?ie=UTF8&s=books&qid=1295802128&sr=8-1)
- 

#### **5.1.5 Share! (2011-01-25 22:20)**

[1]Customizing the Microsoft .Net Framework Common Language Runtime by Steven Pratschner

I've been wanting to read this book for some time, and finally bought it for six pounds over Christmas. The CLR is designed to be used as a library, sharing the address space with other code (the host). This isn't particularly novel – the Lisp system I worked on in the 1990s could be saved as a dll which could be loaded into another process. What is novel about the CLR is that it offers a set of interfaces that allow the host process to control its actions. Everything from thread scheduling (and hence access to synchronisation primitives), threadpool implementation and memory allocation can be controlled by the host by implementing a number of interfaces that the CLR calls out to when it needs to do certain actions. The host can interact with the CLR, asking it to garbage collect and warning it of low memory conditions. Moreover

the host can customise the assembly loading process and the security sandbox in which these assemblies run, and also use HostProtection, surfaced into the libraries via the HostProtection-Attribute, to constrain what the code running in the CLR is allowed to do.

The book covers add-ins really well, and as part of this has a couple of really good chapters on Application Domains. Why they exist, how they are configured, loaded and unloaded and how they can be used effectively are all covered in good detail. The book looks into how the host can protect itself against badly behaved add-ins, preventing resource leakage using critical regions and critical finalisers. These are not really covered very well in other books I have read.

The book is a little dry, but does contain some interesting examples, and gives a good glimpse into what the CLR is doing behind the scenes.

1. [http://www.amazon.co.uk/Customizing-Microsoft-NET-Framework-Language-Runtime/dp/0735619883/ref=sr\\_1\\_1?ie=UTF8&qid=1295991351&sr=1-1](http://www.amazon.co.uk/Customizing-Microsoft-NET-Framework-Language-Runtime/dp/0735619883/ref=sr_1_1?ie=UTF8&qid=1295991351&sr=1-1)
- 

## 5.2 February

### 5.2.1 How do you build a football? (2011-02-18 20:15)

[1]Euler's Gem: The Polyhedron Formula and the Birth of Topology by David S. Richeson

I've always been fascinated by topology. The idea that certain simple properties constrain the shapes that can be constructed in the real world seems amazing to me. Writing a book around Euler's polyhedron formula ( $F+V=E+2$ ) and using it as a means of introducing lots of concepts of topology was a really good idea.

The book starts out by introducing the Greek mathematicians and their classification of the platonic solids, and then quickly moves on to Euler's discovery of the formula and shows a number of proofs. It was the first time that I had seen Legendre's proof using spherical geometry, and the author covers the proof really well. There is also a good discussion of the types of polyhedron that the formula applies to, and a discussion of how it was some time before the correct underlying concepts were discovered. The book uses the polyhedron formula as a way to introduce some graph theory, knot theory and more general topological theorems such as the hairy ball theorem.

I thought this was a really well written book that covers a lot of really interesting material.

1. [http://www.amazon.co.uk/Eulers-Gem-Polyhedron-Formula-Topology/dp/0691126771/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1298059198&sr=8-1](http://www.amazon.co.uk/Eulers-Gem-Polyhedron-Formula-Topology/dp/0691126771/ref=sr_1_1?ie=UTF8&s=books&qid=1298059198&sr=8-1)
-

## **5.2.2 If it's broken, fix it. (2011-02-28 20:06)**

[1]Reality is Broken: Why Games Make Us Better and How They Can Change The World by Jane McGonigal

I bought this book to read on the plane while flying to the US. It was a fun read. I don't claim to know anything about psychology, but the author makes a good case for the idea that game designers have found ways to make virtual environments that motive and entertain us, and keep us more engaged with an activity than the real world. The astonishing amounts of time that players across the world have voluntarily spent on games like World of Warcraft, probably shows that there is a fantastic untapped resource that could be used for many good purposes other than escapist entertainment, if only these projects could be moulded more into a game-like form.

The book is full of interesting summaries of what makes good games and details many experiments concerning new styles of game.

1. [http://www.amazon.co.uk/Reality-Broken-Games-Better-Change/dp/0224089250/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1298918594&sr=8-1-spell](http://www.amazon.co.uk/Reality-Broken-Games-Better-Change/dp/0224089250/ref=sr_1_1?ie=UTF8&s=books&qid=1298918594&sr=8-1-spell)
- 

## **5.2.3 Can you say that a different way? (2011-02-28 20:08)**

The news that Reflector is going to cost money from version 7, has caused the creation of some projects with the aim of generating new versions of this tool. One interesting example is [1]ILSpy which is based on a degree computing project by [2]David Srbecky. The [3]first post points to his [4]dissertation which is a fairly interesting read. He uses SSA to convert the stack based IL into higher level language, and analyses the basic blocks to find the loops and conditionals using some graph transforms. I thought the idea of determining if a branch was needed by removing it and simulating the control flow to see if that branch target was reached, was a neat technique.

There's a post [5]here which summarises the decompilation technique.

The dissertation project didn't do higher level C # features such as lambda expressions and iterators, or even generics, but it will be interesting to see how far this project can go.

1. <https://github.com/icsharpcode/ILSpy>
  2. <http://community.sharpdevelop.net/blogs/dsrbecky/default.aspx>
  3. <http://community.sharpdevelop.net/blogs/dsrbecky/archive/2011/02/11/ilspy-decompiler.aspx>
  4. <https://github.com/icsharpcode/ILSpy/raw/master/doc/Dissertation/Dissertation.pdf>
  5. <http://community.sharpdevelop.net/blogs/dsrbecky/archive/2011/02/19/ilspy-ensuring-correctness.aspx>
-

## 5.2.4 No it's not an assembly language (2011-02-28 20:10)

I keep coming across articles that talk about C# being the assembly language of the .NET platform. Sadly though, C# is just a lens through which we look at the platform, and there are various places where C# doesn't express things quite as well as it might.

In particular, IL has a try...fault... construct, where the fault block is only executed when an exception is making its way out of the try body. This is much like

```
try
{
.... body ....
}
catch (Exception)
{
Console.WriteLine("Fault");
throw;
}
```

However, a change was made in the .NET 4 platform, to avoid unmanaged exceptions getting caught by C# code, which was probably not expecting them,

If we write some C code to generate an exception:

```
extern "C" int __declspec(dllexport) CauseAccessViolation()
{
int *x = 0;
return *x;
}
```

and call it from C#

```
[DllImport("CauseAccessViolation.dll")]
static extern int CauseAccessViolation();

static void Main(string[] args)
{
try
{
CauseAccessViolation();
}
catch (Exception)
{
Console.WriteLine("Fault");
throw;
}
}
```

Under .NET 2, the “Fault” will be printed, but under .NET 4 it won’t. Confusingly, replacing the unmanaged call with a throw of the exception that it raises, `AccessViolationException`, both platforms print “Fault”. This behaviour can be configured using the application’s configuration file, but to me it feels dirty that some Exceptions are being treated specially.

---

## 5.3 March

### 5.3.1 Yes, but can I rely on that? (2011-03-05 12:04)

I've read a couple of quite interesting papers this week.

There's an [1]interesting paper on memory models with some associated discussion on [2]Lambda The Ultimate. Current memory models are either badly underspecified or are hard to relate to a user's code, and many models appear to have undesirable side-effects. The paper discusses some of these issues, and potential ways forward.

Cliff Click's [3]blog has a link to some [4]slides from one of his presentations which discusses Azul System's aims via its Managed Runtime Initiative to get better support inside Linux for their high throughput garbage collector.

1. <http://rsim.cs.illinois.edu/Pubs/10-cacm-memory-models.pdf>
  2. <http://lambda-the-ultimate.org/node/4211>
  3. <http://www.azulsystems.com/blog/cliff/2011-02-17-fosdem-brussels-and-new-blog-software>
  4. [http://www.azulsystems.com/blog/wp-content/uploads/2011/02/2011\\_FOSDEM\\_OpenSrc.pdf](http://www.azulsystems.com/blog/wp-content/uploads/2011/02/2011_FOSDEM_OpenSrc.pdf)
- 

### 5.3.2 What a conference, what a night! (2011-03-12 17:41)

I recently attended the [1]QCon London conference, and I have to say that there were a number of really interesting and thought-provoking talks.

I very much enjoyed Bart De Smet's talk on [2]"LINQ, take two". This talk consisted of some observations on the current state of LINQ. In particular, the speaker pointed out that you don't need to use interfaces when implementing LINQ as this forces the implementation of error raising methods corresponding to functionality that you don't provide, which leads to completions in Visual Studio which aren't really of value. Instead you can use the way that the compiler uses syntactic transforms and straightforward overloading to make it easier to use statement completion on queries for your own provider. The talk then moved on to IQbservable interface and later on to the scheduling of the execution of the LINQ query as the third dimension of query execution. This was the last talk of the conference and it overran by 15 minutes, but it was well worth attending.

[3]Secure distributed programming on ECMASCIRIPT 5 was a talk by Mark Miller which covered how Javascript mash-ups can be secured by using capability based authorisation instead of the usual identity based authorisation. Managed implementations of object systems have a built-in way of implementing capabilities. An object A can only get a reference to an object B if some third party, which already has a pointer B passes it to A via a method call. This is the first step to security. The next version of Javascript allows objects to be frozen, preventing subsequent modification – this means that a base set of objects can be secured against modification by

attackers. The author and collaborators have implemented a Javascript library on top of this which implements a number of membranes – a way to manipulate objects as they are passed through to someone else, in such a way that access can be later revoked. When ECMASCIPT5 is available, it will be possible to use a library to secure the base system by freezing the base objects and patching functions such as eval, and then use these capability mechanisms to allow mash-ups without the danger of a loaded script interacting with another part of the application.

The talk then moved from the local to the remote case, by talking about an implementation of distributed communication using promises. There was an interesting example here of writing a queue in Javascript – the single threadedness of the language makes it necessary to implement many structures using callbacks to avoid the need to block the thread.

[4]“Webmachine: A practical executable model of HTTP” was really interesting from two angles. First it talked about something that was implemented in Erlang. This meant we got to see a small amount of Erlang code, and the implementation had a functional programming flavour with no global state and no mutable shared state. The main point of the talk was that this was a way to do REST correctly. [5]Webmachine is an implementation of the state machine that governs HTTP requests. From this point of view it is interesting in itself, as it highlights some of the corner cases of the specification that are not often seem. The idea behind Webmachine is that this state machine calls out of user defined functions at various points. These functions can change the interaction by returning the appropriate values.

The presenter showed how easy it was to write a simple REST server. The defaults have been chosen to make this easy, and the user defined functions are passed the appropriate arguments, so that if the bodies only depend on the parameters, it is really easy to ensure that the server behaves as it ought.

Webmachine has been ported to a number of languages, and I’m sure it will make its way to other languages in the near future.

[6]“Advanced Reflection” was a talk by JB Evain that highlighted the use of the Cecil library. This is a library for manipulating .NET assemblies. It provides an object model over the assemblies, allowing them to be modified and re-saved. The talk emphasised the use of the library for Aspect Oriented programming - one main example was adding checks to the prologue of functions to check for null parameters.

The current implementation allows a method’s code to be modified at the IL level, and the presenter talked about his future hopes of decompiling the IL into expression trees and allowing the modification at this level.

Cecil (and associated utility libraries) is a really useful tool which we have used at work for reading and adjusting assemblies.

[7]“Node.js: Asynchronous I/O for fun and profit” was an introductory talk on Node.js. Javascript is a really powerful language, and the V8 engine implementation has made it easy to plug in a DOM which can be manipulated using Javascript. Node.js uses event based programming, running a single worker thread (at the user program level) and using the operating system’s select equivalent to find I/O requests that need a response. Requests are plugged into the user program via callbacks, leading to a CPS (continuation passing) style of programming that is alien to many people, but which has been used for a long time in the functional programming community.

Node.js is very scalable due to its use of the operating system to queue the work. A single thread means that there isn’t the memory overhead of many blocked threads, and performance is good as there is little context switching or scheduling overhead. Load balancing happens at the level of having multiple Node.js instances instead of at the threadpool level.

On the Wednesday, I attended two talks on The Java track. [8]“Java’s youthful maturity” talked about the soon to be released version of Java, emphasising the new VM support for dynamic languages via the [9]invokedynamic bytecode. There was also discussion of some concurrency support via the ForkJoin classes which support the ForkJoin pattern. During this discussion we got to see the new lambda syntax in Java, and a potential mechanism for extending existing classes. This latter facility adds default implementations into existing classes to support backwards compatibility.

[10]“The future of Java EE” was very much about getting Java to work better with the cloud. During the talk the presenter used the glassfish application server to deploy an application into the cloud.

I also attended an [11]“Introduction to iOS development” which was a quick tour of the iOS platform and it’s development tools. The introduction to Objective-C was really good, though as I’m a fan of Smalltalk it was easier to grasp the + and - to denote class and instance methods and the parameter syntax. The presenter gave brief guidelines for when an HTML5 application should be written in preference to a native application.

Both days also had relatively interesting keynote presentations, and the conference was really worthwhile.

1. <http://qconlondon.com/>
2. [http://qconlondon.com/london-2011/presentation/LINQ%2C+take+two+%E2%80%93+realizing+the+LINQ+to+Everythin  
g+dream](http://qconlondon.com/london-2011/presentation/LINQ%2C+take+two+%E2%80%93+realizing+the+LINQ+to+Everything+dream)
3. [http://qconlondon.com/london-2011/presentations/show\\_presentation.jsp?oid=2796](http://qconlondon.com/london-2011/presentations/show_presentation.jsp?oid=2796)
4. <http://qconlondon.com/london-2011/presentation/Webmachine%3A+a+practical+executable+model+of+HTTP>
5. <http://webmachine.basho.com/>
6. <http://qconlondon.com/london-2011/presentation/Advanced+Reflection>
7. [http://qconlondon.com/london-2011/presentations/show\\_presentation.jsp?oid=2778](http://qconlondon.com/london-2011/presentations/show_presentation.jsp?oid=2778)
8. <http://qconlondon.com/london-2011/presentation/Java%27s+Youthful+Maturity>
9. <http://java.sun.com/developer/technicalArticles/DynTypeLang/index.html>
10. <http://qconlondon.com/london-2011/presentation/The+future+of+Java+EE>
11. <http://qconlondon.com/london-2011/presentation/Introduction+to+iOS+Software+Development>

---

### 5.3.3 Nobody said it was easy, nobody said it would be that fast (2011-03-19 09:57)

We had an interesting issue at work this week concerning a Visual Studio add-in that was failing but only on versions of Windows that were not in an English locale and only in Visual Studio SP1. We had reports of the failure in Germany and a couple of other European countries. The problem turned out to involve a method that would work normally, but not when called in certain dynamic contexts, though from the definition it isn’t immediately obvious that context would be involved. I’d been thinking a little about program proofs after a colleague had mentioned a talk on Code Contracts he’d attended at QCon. It struck me that the problem we had would have been very hard to find using theorem proving (without getting a lot of false positives).

To set the scene, add-ins in Visual Studio use COM for the communication between the add-in and the Visual Studio code. When an add-in is started, its OnConnect method is called

via COM. COM has a default timeout of something like 3 minutes for calls, so the add-in will be deemed to have failed if it takes longer than this to initialize itself. When we attached windbg to the hanging Visual Studio and loaded the SOS debugger extension, we found that a SecurityPermission exception was being raised a great number of times, and this was causing the OnConnect to time out.

The add-in called the

```
new IsolatedStorageFileStream(...)
```

constructor to try to open a previously saved file. This had a try...catch... block wrapped around it to catch the exception if the file didn't exist. Now, it turns out that Isolated Storage isolates things by asserting permission to access files only in a safe directory.

```
[1]FileIOPermission permission =
new FileIOPermission([2]FileIOPermissionAccess.[3]AllAccess, this.[4]m_isf.[5]RootDirectory);
permission.[6]Assert();
permission.[7]PermitOnly();
```

This is all well and good. While this permission is in effect, the system tries to open the local storage file. If the file doesn't exist, then an exception is thrown. But while the exception is being constructed, the system needs to localize the message string so it needs to look for resource dlls in various places. This involves resolving some assemblies, and it turns out that VS 2010 installs an AppDomain resolve handler which has code of the form:

```
[8]Assembly assembly = null;
foreach ([9]string str in [10]AssemblyPaths(name))
{
try
{
if ([11]File.[12]Exists(str)) {
```

The dynamic context means that the File.Exists is going to fail as it's going to be trying to deal with files that aren't inside the Isolated Storage. Of course, raising another Exception is going to require a localized string which is going to require file access.

All of these exceptions don't crash the system. The Resource resolution code takes a lot of care to avoid crashes by running inside a CER, but it does lead to a lot of exceptions. When I looked, the initial attempt to open the file stream caused 22 exceptions, but the more add-ins that have been previously loaded, the more assemblies the above code tries to access. With one add-in, I got to 400 exceptions before I gave up.

I struck me how hard this would be to verify. The handler is only dangerous in a certain dynamic context, and there isn't actually a crash. There is just a dramatic slowdown which needs to happen in the context of a COM method with a timeout to cause a problem.

It's these kind of issues that make developing software a fun job!

1. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Security.Permissions.FileIOPermission>
2. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Security.Permissions.FileIOPermissionAccess>
3. <http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Security.Permissions.FileIOPermissionAccess/AllAccess>

```
4. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.IO
   .IsolatedStorage.IsolatedStorageFileStream/_m_isf:System
5. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.IO
   .IsolatedStorage.IsolatedStorageFile/property:RootDirec
6. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Se
   curity.CodeAccessPermission/Assert()
7. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Se
   curity.CodeAccessPermission/PermitOnly()
8. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Re
   flection.Assembly
9. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.St
   ring
10. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://Microsoft.VisualStudio.Platform.AppDomainM
    anager:10.0.0.0:b03f5f7f11d50a3a/Microsoft.VisualStudio.
11. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.IO
    .File
12. http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.IO
    .File/Exists(String):Boolean
```

---

### 5.3.4 Skills do matter (2011-03-19 10:04)

Yesterday I attended the [1]Skillsmatter Functional Programming Exchange where there was a sequence of really interesting talks.

The day started with Simon Peyton Jones giving a good introduction to the issues around multi-core concurrency and parallelism. The talk was really entertaining and brought up a great number of interesting points. The presenter obviously emphasised Haskell as an ideal language for experimenting with various solutions to the concurrency problem. He had some great remarks concerning locks and how hard they are to use. “Implementing a double ended queue on a uniprocessor is an undergraduate problem; on a multi-processor it was until recently a conference paper”. GHC has been extended to offer all kinds of solutions – low overhead threads and an \_epoll implementation to allow large numbers of outstanding requests, software transactional memory (where he emphasised that Microsoft’s CLR STM failed owing to the nature of the runtime with the complication of too much mutable state), and also Erlang like isolation models to control the interaction between agents and allow graceful handling of failure. The talk covered the compilation of a subset of Haskell to run on a GPU and also covered data parallelism and nested data parallelism. In the case of the latter, there is a lot of research going on in this area. All very interesting and thought-provoking.

Another talk that mentioned Erlang was the talk on the [2]Akka, a scalable Actor library implemented as a library in Scala. The talk emphasised that the Actor model allows scaling, both within the processes and across machines using remote Actor references. Actors can be linked in the style of process linking in Erlang, – this allows processes to fail at which point other linked processes are notified and recover from the failure.

Adam Granicz also did a really good talk on [3]WebSharper. This product allows you to write F # code inside Visual Studio. The F # compiler will compile it and then the WebSharper tools

can translate some of the code into Javascript allowing it to run on the client. Applications mark which functions should run on the client and which should run on the server, and the WebSharper libraries will take care of the necessary remoting calls between the two sets of code. Too many questions meant that the presentation had to be cut slightly short, but the application looked really powerful, with composable formlets making it easy to abstract over pieces of Web GUI.

There were several other good talks, including a [4]talk by Simon Cousins on how his company started using F #, seamlessly integrating it into an existing C # application.

1. <http://skillsmatter.com/event/scala/functionalpx-2011>
  2. <http://akka.io/>
  3. <http://www.websharper.com/>
  4. <http://www.simontylercousins.net/journal/2011/3/18/functional-programming-exchange-2011.html>
- 

## 5.4 April

### 5.4.1 Get back, he's got a book (2011-04-08 20:01)

[1]Professional F # 2.0 by Ted Neward, Aaron Erickson, Talbott Crowell, Richard Minerich

This book seeks to be an introduction to F # and functional programming for C # programmers. Maybe it is because the book was written by so many people, but I found some parts of the book really good and some parts really uninteresting and not at all illuminating.

The first chapter is a primer which starts with a simple C # example and gradually converts it into equivalent F # code, stripping away all of the ceremony that you are forced to add in an object oriented language. This chapter introduces the notion of a function as a first class object, and shows how the type inference keeps the code looking a lot cleaner. Personally, I'm not sure that it really works, though the eventual contrast between the initial C # and the final F # is certainly a good advertisement for the brevity of F #.

The next chapters introduce the various datatypes and primitives of the F # language. There are a few annoying errors... approximating the distribution of three rolls of a die using a random number in the range 3-18 is not something you really want to do, as the latter has a completely different distribution. The coverage of classes and interfaces, followed by the subject of active patterns is really good.

Again there are a couple of annoying typos, in particular the derivative of a function is defined as

$(f(x+h) - f(x-h)) / 2 * h$

instead of

$(f(x+h) - f(x-h)) / (2 * h)$

There are also several illustrative pieces of code which are not explained and which I couldn't link to the discussion at that point in the tutorial.

There are good chapters on linking F # and C # code, and short chapters on using F # in practical scenarios such as ASP.NET MVC and implementing web services.

I've done a fair amount of F # in the past, and this book contained some really good explanations. However, the style and content was a little variable, presumably depending on the author of that particular part of the book, which made it hard to read.

1. [http://www.amazon.co.uk/Professional-2-0-Wrox-Programmer/dp/047052801X/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1302291749&sr=8-1](http://www.amazon.co.uk/Professional-2-0-Wrox-Programmer/dp/047052801X/ref=sr_1_1?ie=UTF8&s=books&qid=1302291749&sr=8-1)
- 

#### **5.4.2 Call me when you've done the second version (2011-04-23 07:57)**

There are various interesting videos and blog posts out on the new Async CTP. The underlying pattern has changed for efficiency reasons, and the implementation fixes a number of bugs including some race conditions. This new version works on the Windows Phone and the licensing allows it to be used for real.

[1]Stephen Toub covers the new pattern and [2]Lucian Wischik talks about the reasons for the various changes which he also covers in a series of [3]blog posts.

This is really powerful stuff that makes it so much easier to express programs that run in the new asynchronous world of Silverlight.

1. <http://channel9.msdn.com/Shows/Going+Deep/Stephen-Toub-Inside-Async-CTP-SP1-Refresh-Technical-Overview-and-Building-Awaitable-Types>
  2. <http://channel9.msdn.com/Shows/Going+Deep/Lucian-Wischik-Async-Compiler-Bug-Fixes-Updates-and-Core-Improvements>
  3. <http://blogs.msdn.com/b/lucian/>
- 

#### **5.4.3 No more heroes any more (2011-04-25 16:25)**

[1]Hackers – heroes of the computer revolution by Steven Levy

I'd been meaning to read this book for a long time. My early career revolved around the Lisp language, and lots of the early development in this space happened at the MIT AI lab in the 70s and 80s. This lab takes pride of place in the first hacker age and is covered in some detail in this book, as are the later days in which Lisp moved from software into hardware with the commercialization of the Lisp Machines from LMI and Symbolics.

The book goes into some detail about the characters involved in the early computing industry, and this social aspect is quite interesting. Before reading the book, I didn't know the story of early games development shops such as Sierra On-Line and some of the stories of the famous Homebrew Computer Club.

I must admit though that I found the book quite slow going. A seemingly never-ending list of new characters made it quite hard to follow, and after a while all of the characters blurred

into one. Despite the interesting stories that are present in the text, I had several goes at reading the book and then ended up putting it down and starting again from where I left it the previous time.

1. [http://www.amazon.co.uk/Hackers-Heroes-Computer-Revolution-Anniversary/dp/1449388396/ref=sr\\_1\\_2?ie=UTF8&qid=1303726014&sr=8-2](http://www.amazon.co.uk/Hackers-Heroes-Computer-Revolution-Anniversary/dp/1449388396/ref=sr_1_2?ie=UTF8&qid=1303726014&sr=8-2)
- 

#### **5.4.4 Check (2011-04-25 16:25)**

There's a great new series on Channel9 where Eric Meier interviews various developers at Microsoft. There have been several good episodes. One in particular was the [1]discussion with Jeff Wilcox who talks about getting Silverlight running on the Windows 7 Phone.

1. <http://channel9.msdn.com/Shows/Checking-In-with-Erik-Meijer/Checking-In-Jeff-Wilcox-Writing-the-WP7-App-Platform-in-C-and-C>
- 

#### **5.4.5 It certainly brings out the artist in you (2011-04-25 17:05)**

[1]Hackers and Painters: Big ideas from the computer age by Paul Graham

This was another book that I just happened to pick up in Cambridge Public Library. I'd obviously heard of Paul Graham in my Lisp programming past. His Viaweb company, which was eventually sold to Yahoo, used Common Lisp as the underlying technology which gave his start-up an advantage over the other players in the web store area.

This book is a very interesting collection of essays on a number of subjects. There are obviously some essays on Common Lisp, and why the language is so great. Graham lists the ideas that make a language great, and then shows how many other languages only have a subset of these features. Lisp, even though it dates so far back, has many features that are only fairly recently making their way into more mainstream languages, and still has a few features that haven't yet migrated across. For example, lexical closures took a while to make it into the C # language, and features like LINQ rely on compiler source-to-source transforms without giving the ability for the user to define some features as macros for themselves.

There are many essays on lots of disparate subjects. Why the school system is failing nerds, why start-ups are more effective at getting products to market and loads of other thought-provoking pieces. All of the essays are really easy to read and it's hard to put the book down.

1. [http://www.amazon.co.uk/Hackers-Painters-Big-Ideas-Computer/dp/1449389554/ref=sr\\_1\\_1?ie=UTF8&qid=1303750472&sr=8-1](http://www.amazon.co.uk/Hackers-Painters-Big-Ideas-Computer/dp/1449389554/ref=sr_1_1?ie=UTF8&qid=1303750472&sr=8-1)

---

## 5.5 May

### 5.5.1 Where is the string that Theseus laid? (2011-05-07 19:16)

Mention Haskell and IO in the same paragraph and you'll always find the word Monad in close proximity. Monads are the construct (pattern) that Haskell uses to thread the world state through a series of computations, but when we're dealing with input and output we need a bit more – and that's rarely mentioned in tutorial articles. In order to guarantee that output happens in the right order, interactions governed by the IO monad need to throw away some laziness; some parts of the computation need to be strict. This [1]recent blog post is the best explanation I have read concerning this issue and is well worth a read.

1. [http://blog.ezyang.com/2011/05/unraveling-the-mystery-of-the-io-monad/?utm\\_source=feedburner&utm\\_medium=fed&utm\\_campaign=Feed%3A+ezyang+%28Inside+245s%29](http://blog.ezyang.com/2011/05/unraveling-the-mystery-of-the-io-monad/?utm_source=feedburner&utm_medium=fed&utm_campaign=Feed%3A+ezyang+%28Inside+245s%29)

---

### 5.5.2 What's on your mind? (2011-05-07 19:16)

Anders Hejlsberg answers some questions on C # in this [1]recent Channel 9 video. I was hoping it would go into more detail about [2]Roslyn – sadly, it doesn't.

1. <http://channel9.msdn.com/Shows/Going+Deep/Anders-Hejlsberg-Questions-and-Answers>
2. <http://blogs.msdn.com/b/ericlippert/archive/2010/12/16/hiring-for-roslyn.aspx>

---

### 5.5.3 No assembly required (2011-05-16 23:30)

I've been programming on .NET for quite some time but only recently came across the following [1]method which allows an object to be created with all of the fields zeroed out. This can't be done at the C # language level, but this functionality is required by various parts of the serialisation code.

1. <http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatterservices.getuninitializedobject.aspx>

---

#### **5.5.4 Crash bang (2011-05-22 09:34)**

The first game I played on the Play Station was Crash Bandicoot. There's a series of blog posts starting [1]here, where Andy Gavin describes how the game was designed, and how they worked around the limitations in the processing power of the console.

1. <http://all-things-andy-gavin.com/2011/02/02/making-crash-bandicoot-part-1/>
- 

#### **5.5.5 You looked better from the other side (2011-05-22 09:35)**

[1]Why Beauty is Truth: A History of Symmetry by Ian Stewart

Another really good book by Ian Stewart. Like lots of other books around this area, it gives a basic introduction to group theory and discusses how Galois and others used this to show that there is no general solution to the quintic. As well as providing historical data about the characters involved, this book does a really good job at describing the mathematics behind the proof in a really easy to understand way. Better still, rather than finishing at this point, the book then goes on to discuss Lie groups and their relationship to relativity and quantum mechanics.

1. [http://www.amazon.co.uk/Why-Beauty-Truth-History-Symmetry/dp/0465082378/ref=sr\\_1\\_1?ie=UTF8&qid=1306052999&sr=8-1](http://www.amazon.co.uk/Why-Beauty-Truth-History-Symmetry/dp/0465082378/ref=sr_1_1?ie=UTF8&qid=1306052999&sr=8-1)
- 

#### **5.5.6 Mix it up (2011-05-22 09:49)**

Channel 9 have posted a great set of videos from Mix 11 covering a range of subjects from Javascript to Windows Phone to HTML 5.

[1]Doug Crockford covers the new ECMAScript 5 standard and then takes part in a [2]panel discussion on Javascript. The latter contains some good discussion on the direction that the language should take in the future. One future extension will definitely be a notion of a module - currently, people end up using functions for encapsulation, leading to the rather odd looking pattern where a function is defined and then immediately applied.

(function () ... ) ()

Two other related talks are one on the [3]Rx extensions for Javascript, in which a team has taken Rx, a library which makes it easy to deal with asynchronous events, and ported it to work in

Javascript, and a talk on [4]Script #, a project which compiles a subset of C # into Javascript. For the latter project, the author argues that C # is a better language for programming the large, and says that several Microsoft projects have used Script # to get the benefits while still generating code that runs natively client-side in the browser. My worry about such tools is that there is no debugging support, so you start with code that is written in C # but end up debugging it as code that is written in Javascript. To make this practical, the transforms that happen as part of the translation need to be easy to understand to allow the person doing the debugging to understand how they relate to one another.

[5]Miguel de Icaza discusses the Mono variants than run on iPhone and Android. C # is definitely a well designed programming language that makes it easy to write complicated applications in a managed environment, and it seems to be a good idea to make it available on various portable devices. Currently there doesn't seem to be a unifying GUI framework (such as Silverlight) across the two platforms, so you end up getting a C # wrapper around the native libraries, but this seems to be a great way to improve productivity when writing for these devices.

HTML5 is going make the browser a much richer platform for client side application development. One of the important technologies is certainly Canvas, and there's a good discussion of this [6]here. Canvas may well make it possible to avoid the need for Flash or Silverlight in the future, and there's a good talk contrasting the two [7]here. SVG is another useful part of the new offerings [8]covered too at the conference.

IE now has some [9]debugging extensions that look a lot like those of FireBug.

Now that people want to access data from many different places, Identity is becoming a very important issue. Mix had a talk on [10]identity in the extended web.

One other set of talks was on the future of Windows Phone. The most interesting was on the [11]architecture of the phone, which also covers the extensions that are going to be made available in the next release, "Mango". This release is going to offer fast application switching – when an application is terminated, it will not necessarily be flushed from memory, but will be left around in case the user switches back to it, at which point it can be quickly restarted. Windows Phone applications already get notifications that they are about to be terminated at which point they persist their state (tombstoning). This is a simple extension to the initialization mechanism where an event argument is passed telling the application that the current instance is simply restarting and hence may not need to reload all of its state. This talk covers this aspect [12]in more detail. There were some other really interesting talks on Windows Phone at TechEd which I'll cover in more detail in another post.

1. <http://channel9.msdn.com/Events/MIX/MIX11/EXT13>
2. <http://channel9.msdn.com/Events/MIX/MIX11/RES02>
3. <http://channel9.msdn.com/Events/MIX/MIX11/HTM07>
4. <http://channel9.msdn.com/Events/MIX/MIX11/HTM16>
5. <http://channel9.msdn.com/Events/MIX/MIX11/EXT03>
6. <http://channel9.msdn.com/Events/MIX/MIX11/HTM03>
7. <http://channel9.msdn.com/Events/MIX/MIX11/HTM14>
8. <http://channel9.msdn.com/Events/MIX/MIX11/HTM15>
9. <http://channel9.msdn.com/Events/MIX/MIX11/HTM18>
10. <http://channel9.msdn.com/Events/MIX/MIX11/SVC03>
11. <http://channel9.msdn.com/Events/MIX/MIX11/DVC19>
12. <http://channel9.msdn.com/Events/MIX/MIX11/DVC19>

---

## 5.6 June

### 5.6.1 A tale of recursion (2011-06-06 17:21)

You spend a lot of time learning about tail recursion, and how it is a way of writing loops while keeping the code in a functional style. So it's a shame that the .NET platform doesn't always guarantee tail call optimisation, in particular for self tail calls. The problem is, of course, that you'll write something and it will work on all sorts of simple tests and then fail on large customer examples owing to a greater depth in the recursion at runtime. There are no end of examples of C # code which manipulates binary trees, and which would fall prey to this trap.

Take, for example, the following program:

```
class Program
{
    static void Main(string[] args)
    {
        TailMe(1000000, 0);
    }

    static int TailMe(int x, int y)
    {
        if (x == 0)
            return y;
        Console.WriteLine(x);
        return TailMe(x - 1, y + 1);
    }
}
```

Compile it in debug mode, and you get the following output:

```
...
984112
984111
```

Process is terminated due to StackOverflowException.

Compile for "Any Cpu" release mode and run on an x64 machine:

```
3
2
1
```

Compile it for "x86" release mode:

```
871186
871185
871184
```

Process is terminated due to StackOverflowException.

What is interesting is that even though the IL doesn't use a tail prefix on any of the IL instructions (which is a hint to the jit that it would be good to tail call in this particular case)

```
.method private hidebysig static int32
TailMe(int32 x,
int32 y) cil managed
{
// Code size 23 (0x17)
.maxstack 8
IL_0000: ldarg.0
IL_0001: brtrue.s IL_0005

IL_0003: ldarg.1
IL_0004: ret

IL_0005: ldarg.0
IL_0006: call void [mscorlib]System.Console::WriteLine(int3 2)
IL_000b: ldarg.0
IL_000c: ldc.i4.1
IL_000d: sub
IL_000e: ldarg.1
IL_000f: ldc.i4.1
IL_0010: add
IL_0011: call int32 TailMe.Program::TailMe(int32,
    int32)
IL_0016: ret
} // end of method Program::TailMe
```

in the x64 release case, the code has been jitted to have a jmp to implement the self tail call.

```
0:004> u 000007ff00140160
000007ff'00140160 53 push rbx
000007ff'00140161 57 push rdi
000007ff'00140162 4883ec28 sub rsp,28h
000007ff'00140166 8bfa mov edi,edx
000007ff'00140168 8bd9 mov ebx,ecx
000007ff'0014016a 660f1f440000 nop word ptr [rax+rax]
000007ff'00140170 85db test ebx,ebx
000007ff'00140172 750c jne 000007ff'00140180
000007ff'00140174 8bc7 mov eax,edi
000007ff'00140176 eb4b jmp 000007ff'001401c3
000007ff'00140178 0f1f840000000000 nop dword ptr [rax+rax]
000007ff'00140180 48b8f010de1200000000 mov rax,12DE10F0h
000007ff'0014018a 488b00 mov rax,qword ptr [rax]
000007ff'0014018d 4885c0 test rax,rax
000007ff'00140190 750e jne 000007ff'001401a0
000007ff'00140192 b101 mov cl,1
000007ff'00140194 e8e7b79fe5 call mscorelib _ni+0x3db980 (000007fe'e5b3b980)
000007ff'00140199 0f1f800000000000 nop dword ptr [rax]
000007ff'001401a0 49bbf010de1200000000 mov r11,12DE10F0h
000007ff'001401aa 4d8b1b mov r11,qword ptr [r11]
000007ff'001401ad 498b03 mov rax,qword ptr [r11]
000007ff'001401b0 4c8b4060 mov r8,qword ptr [rax+60h]
```

```
000007ff'001401b4 8bd3 mov edx,ebx
000007ff'001401b6 498bcb mov rcx,r11
000007ff'001401b9 41ff5028 call qword ptr [r8+28h]
000007ff'001401bd ffcb dec ebx
000007ff'001401bf ffc7 inc edi
000007ff'001401c1 ebad jmp 000007ff'00140170 <<<< self tail call
000007ff'001401c3 4883c428 add rsp,28h
000007ff'001401c7 5f pop rdi
000007ff'001401c8 5b pop rbx
000007ff'001401c9 c3 ret
```

---

### 5.6.2 Thanks for the memory (2011-06-06 20:25)

You often hear people talking about the CLR's generation structure and how it allows the runtime to do really efficient bump-pointer allocation.

The idea here is that gen0 is a chunk of memory into which various threads are going to allocate. We'd like to have a pointer to the start of the free space, and every time we allocate we'd like to increment this pointer by the size of the allocation (ie bump the pointer) and place the new object into the space we just walked over, until we reach the end of the area. The trouble is that if we are going to allow multiple threads to allocate into the same area, it looks like we might need to use a lock or atomic operation around the bumping of the pointer, which sounds inefficient.

Using WinDbg and a simple program, we can see how the CLR does this.

```
class Program
{
    private static AutoResetEvent s_Event1 = new AutoResetEvent(false);
    private static AutoResetEvent s_Event2 = new AutoResetEvent(false);

    static void Main()
    {
        new Thread(() =>
        {
            while (true)
            {
                s_Event1.WaitOne();
                new Program();
                s_Event2.Set();
            }
        }).Start();
        new Thread(() =>
        {
            while (true)
            {
```

```

s_Event1.Set();
s_Event2.WaitOne();
new Program();
}
}).Start();
}
}

```

We can find the code using WinDbg

```

0:006> u 004401a8
004401a8 55 push ebp
004401a9 8bec mov ebp,esp
004401ab 8b0dec1f4a03 mov ecx,dword ptr ds:[34A1FECh]
004401b1 8b01 mov eax,dword ptr [ecx]
004401b3 8b402c mov eax,dword ptr [eax+2Ch]
004401b6 ff500c call dword ptr [eax+0Ch]
004401b9 b954381800 mov ecx,183854h
004401be e85d1ed3ff call 00172020
004401c3 8b0df01f4a03 mov ecx,dword ptr ds:[34A1FF0h]
004401c9 3909 cmp dword ptr [ecx],ecx
004401cb e808c3f255 call mscorlib _ni+0x1dc4d8 (5636c4d8)
004401d0 ebd9 jmp 004401ab

```

The allocation happens by loading the handle 183854h into ecx and jumping into the allocation routine.

```

00172020 8b4104 mov eax,dword ptr [ecx+4]
00172023 648b15400e0000 mov edx,dword ptr fs:[0E40h]
0017202a 034248 add eax,dword ptr [edx+48h]
0017202d 3b424c cmp eax,dword ptr [edx+4Ch]
00172030 7709 ja 0017203b
00172032 894248 mov dword ptr [edx+48h],eax
00172035 2b4104 sub eax,dword ptr [ecx+4]
00172038 8908 mov dword ptr [eax],ecx
0017203a c3 ret
0017203b e9c1cca157 jmp clr!CoUninitializeEE+0xb59 (57b8ed01)

```

Setting a breakpoint and running to it.

```

0:006> bp 0017202a
0:006> g
Breakpoint 0 hit
eax=0000000c ebx=00000000 ecx=00183854 edx=00528150 esi=024abc20 edi=034a101c
eip=0017202a esp=04d4f44c ebp=04d4f450 iopl=0 nv up ei pl nz na po nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000202
0017202a 034248 add eax,dword ptr [edx+48h] ds:002b:00528198=0267ab4c
0:005> dd 00528150
00528150 57c764b8 0000b020 00000001 04d4f6cc
00528160 00000000 00000000 00000000 00000004
00528170 00000000 00528178 00528178 00528178
00528180 00000000 00000000 00000000 004eb520
00528190 7efa9000 00000000 0267ab4c 0267c054

```

```
005281a0 00ae2058 00000000 00000000 00000000
005281b0 00000000 5816cffc 005217b8 00000000
005281c0 00000000 00000000 00000000 00000000
0:005> g
Breakpoint 0 hit
eax=0000000c ebx=00000000 ecx=00183854 edx=00526660 esi=024abb30 edi=034a101c
eip=0017202a esp=04ecf71c ebp=04ecf720 iopl=0 nv up ei pl nz na po nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000202
0017202a 034248 add eax,dword ptr [edx+48h] ds:002b:005266a8=02678b58
0:004> dd 00526660
00526660 57c764b8 0000b020 00000001 04ecf99c
00526670 00000000 00000000 00000000 00000003
00526680 00000000 00526688 00526688 00526688
00526690 00000000 00000000 00000000 004eb520
005266a0 7efac000 00000000 02678b58 0267a054
005266b0 00ae3020 00000000 00000000 00000000
005266c0 00000000 0000003a 005217a0 00000000
005266d0 00000000 00000000 00000000 00000000
```

We can see that the thread local storage gives access to two separate allocation areas on the two threads. In one it is 02678b58 - 0267a054 and in the other it is 0267ab4c - 0267c054. A clever trick.

---

### 5.6.3 That's not the kind of thing I provide (2011-06-07 20:25)

I keep coming across speculation about the forthcoming Type Providers in the next version of F#. The best speculation I have seen so far is Robert Pickering's [1]blog post where a Type Provider is expected to be an assembly implementing an interface that allows it to participate in the compiler's pipeline, allowing it to inject code via an expression tree. Tomas Petricek has a [2]set of slides from a recent talk about Accessing Loosely Structured Data.

1. <http://strangelights.com/blog/archive/2010/11/15/the-future-of-f-type-providers.aspx>
  2. <http://www.slideshare.net/tomaspfb/data-8109295>
- 

### 5.6.4 That had slipped my memory (2011-06-11 06:04)

I recently spent a week at work putting together some material on .NET memory management. It's always interesting trying to put material together for a talk as it's guaranteed that you'll notice links between things that you've not noticed before. The slides that I produced are

available [1]here.

- 
1. <http://cid-3f21df299c355e7f.office.live.com/self.aspx/Public/SomeMemoryMisconceptions.pptx>

### **5.6.5 Watch out for the dragons (2011-06-11 06:09)**

[1]Professor Stewart's Hoard of Mathematical Treasures by Ian Stewart

Yet another very interesting and entertaining book by Ian Stewart. This one is a collection of puzzles together with a number of quick (several page) discussions on various mathematical topics such as sphere inversions and gear linkages. The book is really entertaining and it covers such a wide area that it is guaranteed that everyone will find something in the book.

1. [http://www.amazon.co.uk/Professor-Stewarts-Hoard-Mathematical-Treasures/dp/1846683467/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1307772263&sr=8-1](http://www.amazon.co.uk/Professor-Stewarts-Hoard-Mathematical-Treasures/dp/1846683467/ref=sr_1_1?ie=UTF8&s=books&qid=1307772263&sr=8-1)
- 

### **5.6.6 Before you know it you're up to version 4 (2011-06-11 06:21)**

[1]C # 4.0 How-To by Ben Watson

This book consists of a series of short how-tos covering a vast range of topics. Each how-to consists of something like a half page of text followed by some example C # code.

The book covers lots of material, starting out with some C # language ideas, highlighting new features in C # 4 as well as offering advice around constructs that have been in the language for a long time, for example giving advice on exception classes and serialization. There are more how-tos on handling data which covers database and file system access, user interaction covering ASP.NET and WPF/Silverlight and a final section on advanced C # which includes information on threads.

The book was good to read. At times it felt like it might be more appropriate as a reference book that you'd go to when you have a problem rather than as a book you read from end to end (as I did), but I must admit that I learned a lot from reading it so it was worth persevering.

1. [http://www.amazon.co.uk/C-4-0-How--Ben-Watson/dp/0672330636/ref=sr\\_1\\_1?ie=UTF8&qid=1307772649&sr=8-1](http://www.amazon.co.uk/C-4-0-How--Ben-Watson/dp/0672330636/ref=sr_1_1?ie=UTF8&qid=1307772649&sr=8-1)
-

## **5.6.7 All sparks die out in the end (2011-06-14 18:56)**

Those clever Haskell people have been doing more work on parallel Haskell programs, this time using a [1]monad to encapsulate the deterministic parallelism which allows the choice of different scheduling strategies. [2]Tomas Petricek extends this in a blog post to implement the unamb operator. Simon Marlow has a great tutorial on GHC's parallelism [3]here.

1. <http://research.microsoft.com/en-us/um/people/simonpj/papers/parallel/monad-par.pdf>
  2. <http://tomaspetricek.com/blog/SpeculativeParMonad.aspx>
  3. <http://community.haskell.org/~simonmar/par-tutorial.pdf>
- 

## **5.6.8 My generation would have done it differently (2011-06-19 07:32)**

It appears that Mango, the latest release of Windows Phone, is going to have a generational garbage collector. There's a post [1]here that gives some of the details, and some other related posts [2]here and [3]here. Of these latter two posts, the first discusses why the generational GC is going to make a performance difference, and the second discusses the infamous question of whether it is necessary to manually clear local variables to allow the garbage collector to allow objects early.

It seems pretty amazing to me that they have only just started using a generational collector given the rich history of such technology. Generational collectors add memory overheads in the form of tables which track the inter-generational references and extra instructions which keep these tables up to date, but I'd have imagined that the performance gains would have made this all worthwhile years ago.

1. <http://blogs.msdn.com/b/abhinaba/archive/2011/06/14/wp7-mango-the-new-generational-gc.aspx>
  2. <http://blogs.msdn.com/b/abhinaba/archive/2011/06/08/wp7-mango-mark-sweep-collection-and-how-does-a-generational-gc-help.aspx>
  3. <http://blogs.msdn.com/b/abhinaba/archive/2011/01/04/wp7-when-does-gc-consider-a-local-variable-as-garbage.aspx>
- 

## **5.6.9 I won't make a hash of that (2011-06-19 21:03)**

Occasionally something like [1]this talk comes along which covers my favourite parts of computer science - algorithms, multi-processor scaling, memory models and clever design tricks. Cliff Click of Azul Systems talks about his design for a highly scalable hash table implemented in Java. Azul Systems produces hardware and software that allows a system to run many Java threads concurrently. The talk quotes numbers in the region of 1000 concurrent threads, and starts with a discussion of how the standard concurrent hash table doesn't scale to this number

of concurrent accesses. The standard hash table uses striping to spread multiple locks (16 by default) across of the hash table and hence alleviate some of the lock contention that you'd get with a single lock. This runs into trouble with a very large number of threads, and hence the need for a new algorithm.

This new algorithm doesn't use locks, but uses a combination of weak-CAS and some memory barriers at certain places in the resize code. A CAS, a check and set instruction is available as an instruction on all modern architectures and is the primitive on which most of the higher level concurrency primitives such as locks and semaphores are built. For example CLR locks are implemented using [2]two CAS operations. On some architectures (such as x86) a CAS also acts as a memory barrier, forcing the caches on multiple processors to synchronise, to flush pending stores and cache invalidations, though this algorithm allows the CAS to be of the weaker non-barrier form which simply requires the executing processor to gain exclusive use on the relevant cache line.

The basic table, get() and put() with no resizing which we'll come to in a moment, works by implementing the hash table as a closed table, currently sized at a power of two, with sequential probing. The argument for this being that sequential probing is more cache friendly – when we've loaded the cache line for a particular key value pair (which we expect to be next to each other in memory), we've likely also got the next key and value in the same cache line. If at any point we find we need to do too many secondary probes, the hash table will be resized to get the keys to spread out better. To allow the use of weak-CAS, when a key has been inserted into the table, we will never remove it. If an item is deleted, the value is replaced by a tombstone value, which can be subsequently replaced by some other value in some later put() operation.

Resizing works by having two tables for storing the data. Values will be migrated from the old table to the new table, usually by having threads do a small part of this work whenever they try to use the table. When the key has been re-homed into the new table, the value in the old table is replaced with a value representing "look me up in the new table". This need for a field to hold a new table is the point at which a memory barrier needs to be used, to ensure that this forward pointer can be seen by people who read the forward value. get() operations try the old table whereas put() operations all write their effects into the new table.

This resizing mechanism is the code that flushes keys that are associated with tombstone values. These need not be copied into the forwarding table.

The whole algorithm is really neat, and the author uses a correctness proof that works using the possible state transitions that the threads can see. There are [3]slides here that accompany the talk. The slides show measurements of the performance which is really good.

1. <http://video.google.com/videoplay?docid=2139967204534450862>
2. <http://www.bluebytesoftware.com/blog/2009/01/09/SomePerformanceImplicationsOfCASOperations.aspx>
3. [http://www.azulsystems.com/events/javaone\\_2007/2007\\_LockFreeHash.pdf](http://www.azulsystems.com/events/javaone_2007/2007_LockFreeHash.pdf)

## 5.7 July

### 5.7.1 Please keep your focus! (2011-07-06 17:31)

I learned something that I didn't know about WindowsForms the other day. It all started while debugging a problem with an application in which there's a web browser pane hosted in an otherwise WindowsForms application. The problem concerned the focus. If you gave the focus to the web browser pane, then shifted focus to another application, and then clicked the title bar of the WindowsForms application, the focus would no longer be in the embedded web browser, but would have shifted to the last control that had been given focus in the WindowsForms.

I got out Spy++ and watched the windows messages that were being sent around. Sure enough, the web browser element was being given the focus, but when the WindowsForms application was reactivated, the WM\_FOCUS message was being sent to the last control of the WindowsForms controls that had been given the focus.

I spent a while looking around for clues, and it was only when I saw that the Form supported an ActiveControl property that I understood that WindowsForms itself must be tracking the focus. It was then easy to put things together. Control, in System.Windows.Forms, subclasses the various WindowsForms controls, overriding the WndProc method. If you look inside that method, you'll notice that focus changes are propagated up to the container control that contains the element that was given the focus. This container is then responsible for tracking who has the focus. This explained the issue I was seeing. In that case, Windows has given the focus to a control that hasn't been subclassed by WindowsForms, and hence nothing is told that the focus has moved, leaving the tracked focus on the last WindowsForms control that had it.

Why does WindowsForms track the focus? So that it can do all kinds of useful things for the application writer. For example, it can ensure that the control with the focus is visible in the GUI. As an example of this, try the following small program. It sets up a Form that contains a scrollable panel, and the panel contains two text boxes only one of which is visible. After a suitable delay, we set the focus to the textbox which is currently outside the viewing area. This message is handled by the WindowsForm code, which notifies the container which can then take care of scrolling the focussed element into view.

```
[DllImport("user32.dll", CharSet = CharSet.Auto, ExactSpelling = true)]
public static extern IntPtr SetFocus(HandleRef hWnd);
```

```
[STAThread]
static void Main(string[] args)
{
    var form = new Form { Height = 150, Width = 150 };
    var panel = new Panel { Height = 100, Width = 100, AutoScroll = true };
    var txt1 = new TextBox { Location = new Point(0,0) };
    var txt2 = new TextBox { Location = new Point(500,500) };
    form.Controls.Add(panel);
    panel.Controls.Add(txt1);
    panel.Controls.Add(txt2);
    form.Load += delegate {
        HandleRef txt2Handle = new HandleRef(txt2, txt2.Handle);
        SynchronizationContext context = SynchronizationContext.Current;
        ThreadPool.QueueUserWorkItem(delegate
```

```
{  
    Thread.Sleep(5000);  
    context.Post(delegate { SetFocus(txt2Handle); }, null);  
});  
};  
Application.Run(form);  
}
```

I hadn't realised before that WindowsForms offered all of this functionality. As long as you stay in its domain, all of this magic happens without you having to think about it. However, when you interact with elements that are not under its control, then things get a little more confusing.

---

### 5.7.2 You can catch lots in your .NET (2011-07-09 13:23)

[1]Pro DLR in .NET 4 by Chaur Wu

I found this book in the book store at Tech Ed. The DLR is a really interesting area of .NET 4, though aside from a few articles in MSDN magazine, I've never really seen any in-depth information about how it works and the reasons behind the design. This was exactly the complaint of the author of IronJS in a [2]recent podcast. However, dynamic objects are finding a place in the .NET world, including active record implementations like Simple.Data that was [3]recently featured on Herding Code.

I didn't find this the easiest book to read. Sometimes, the author includes a lot of tightly packed text without inline code examples, which made it quite hard to follow. I decided to give the book a chance by reading it twice, and this certainly seemed to help as the second time through I found the material much easier to digest.

The book covers several areas. The first, and most interesting aspect to me, was the support the DLR offers for efficient execution of late bound operations. This includes the DLR expression trees which offer variable binding, lambdas, labels and goto statements, giving a very expressive language, and the multi-level cache design in the notion of call sites and language binders. Having looked at this as a means of understanding the implementation of the dynamic type in C #, I didn't fully understand how these ideas fitted together to give the interoperability story between C # and dynamic languages such as Ruby and Python. This book covers this really well, showing examples of interworking between all of these languages. The other part of the interoperability story I neglected to study in the past is the set of hosting APIs that the DLR offers. The idea here is that global information can be set up for a language, before any code executes in that language. This is needed to bootstrap one language into the world of another. Again the book offers numerous examples of the languages working together.

Part two of the book covers the uses of the DLR. The first chapter gives a brief introduction to aspect oriented programming (AOP), which I thought was good, and then goes on to demonstrate how you could add aspects to objects in C #. The difficulty here is that the C # classes have to implement IDynamicMetaObjectProvider which they need to implement to return an

AopMetaObject which can be implemented in a library to add the necessary point-cut processing. When compared against Spring.net's way to doing AOP by intercepting object creation and returning a dynamic proxy, this need to modify the classes looked less than convincing to me.

The second use of the DLR is in a chapter on meta-programming, where the author writes a class which handles dynamic data access. Dynamic objects can be effectively extended to add fields corresponding to columns in a dynamically provided data table. This chapter had some great code examples of manipulating expression trees to get powerful results, and covers the kind of ground where Simple.Data is making a name for itself.

Other uses of the DLR are in implementing a scripting language that processes the definition of a DSL for hosting (and running) multiple source languages, and a very interesting chapter on using the DLR from Silverlight. This latter chapter covers a project named Gestalt which allows script blocks written in multiple languages to be embedded inline into an HTML page. Gestalt takes care of finding these blocks and feeding them into an instance of the DLR which then runs them. This allows the user to write a single HTML page that contains script written in a number of languages. The examples of this were very cool, though [4]Gestalt doesn't appear to have much development for some time.

Even though Microsoft is no longer actively developing Iron Python and Iron Ruby, at least the DLR has been included into the latest .NET to support the dynamic type that was added to C#. It isn't clear if C# is going to get any more dynamic features in the future, but this book serves as a good introduction to the kinds of things that you can do with dynamic objects and how they can be efficiently implemented.

1. [http://www.amazon.co.uk/Pro-DLR-NET-Experts-Voice/dp/1430230665/ref=sr\\_1\\_1?ie=UTF8&s=books&qid=1309871258&sr=8-1](http://www.amazon.co.uk/Pro-DLR-NET-Experts-Voice/dp/1430230665/ref=sr_1_1?ie=UTF8&s=books&qid=1309871258&sr=8-1)
  2. <http://www.hanselman.com/blog/HanselminutesPodcast271InsideIronJSACompleteJavaScriptECMAScriptOpenSourceImplementationOnTheNETDLR.aspx>
  3. <http://herdingcode.com/?p=305>
  4. <http://gestalt.codeplex.com/>
- 

### 5.7.3 Clojure becomes more portable (2011-07-24 06:48)

There has been a flurry of activity on [1]Twitter after Rich Hickey's recent [2]announcement of ClojureScript. Clojure was initially developed for the Java Virtual Machine, and although there has been a project to port it to the Microsoft CLR (with only limited success), this development looks like it will radically extend the usefulness of the language. ClojureScript is a subset of Clojure, and I assume they'll take the opportunity to get rid of parts of the language that aren't really used in production applications. For example, use of eval is typically frowned upon by Lisp developers.

For me, the highlights are that the ClojureScript compiler is written in Clojure. The standard Clojure implementation has a compiler written in Java and a library that is written mainly in Clojure requiring work in two languages to extend it. Writing the compiler in the language itself means that improvements in the language can lead to improvements to the compiler too.

The ClojureScript system also uses the Google Closure compiler to post-process the generated Javascript. The Closure compiler is used by Google to optimise its various Javascript applications such as Gmail. It does a number of sophisticated compiler optimisations including tree shaking to get rid of code that will never be used by the application, leading to very small executables. There are several blog posts about this, including the start of a series by one of the developers [3]here.

There have been other languages that compile down to Javascript, including the seminal [4]GWT which has been successfully used to write large applications. I imagine the only difficulty is the debugging story - as a programmer who wrote the application in one language, it may be hard to understand the mapping to another language in order to debug the final application.

Clojure will now be able to run on top of various App Engines such as node.js, and there is even a REPL that runs on top of Javascript engine Rhino. When running in the browser, ClojureScript applications will now be able to use Javascript libraries such as jQuery.

It will be interesting to see how far [5]ClojureScript is developed, and where this leaves the standard Java implementation. There is some discussion on [6]Hacker News.

1. <http://twitter.com#!/search?q=%23ClojureScript>
  2. <http://blip.tv/clojure/rich-hickey-unveils-clojurescript-5399498>
  3. <http://blog.fogus.me/2011/07/21/compiling-clojure-to-javascript-pt1/>
  4. <http://code.google.com/webtoolkit/>
  5. <https://github.com/clojure/clojurescript>
  6. <http://news.ycombinator.com/item?id=2787851>
- 

#### **5.7.4 Please mind the gap to the platform (2011-07-26 14:41)**

Not so long ago there was a flurry of panic in the .NET space after Microsoft announced that [1]Windows 8 was going to be an HTML5 platform, without making it clear as to how existing .NET features such as Silverlight were going to fit into the picture. At around this time I happened to come across the book: [2]HTML5 Up and Running by Mark Pilgrim.

This is a very slim book, with only 199 pages, but was a very informative read. The introduction is a very concise history of the development of web standards, explaining the W3C's aborted detour in its attempt to make HTML into an XML vocabulary. The book then goes through the various parts of the HTML5 specification, covering topics such as canvas, local storage, the new semantic elements and offline applications, as well as covering in depth topics such as video and geo-location. The chapter on video covered more information on container formats and the various video and audio encodings than I've ever seen in one place before, even explaining the utilities to use to do the various encodings. The author covers each of the major browsers, and discusses how well they support the various specifications.

All of the chapters contain examples of the technology, and the author emphasises libraries such as [3]Modernizr which allow applications to dynamically determine which parts of the HTML5 specification are supported by the hosting browser, allowing the application to use a plug-in such as [4]Google Gears to support the missing functionality or simply degrade the

user experience. The only technology that didn't get a lot of coverage was [5]Web Workers for which there is a lot of tutorial material available elsewhere.

This was a brilliant compact read for the busy developer and I highly recommended it.

I should also point to a Mix video which offers a [6]deep dive into canvas.

Application development for mobile applications is an area where it seems that web applications, with their new HTML5 ability that allows them to run offline, might just be a good thing. For certain types of application, writing in HTML5 would allow an application writer to produce applications that run on a number of different mobiles, without the need to produce different versions for the different Gui libraries and more specifically the different languages that the platforms use for development. While looking into the topic of application development using HTML, CSS and JavaScript I came across the online book: [7]Building iPhone Apps With HTML, CSS and JavaScript by Jonathan Stark

This book uses the [8]PhoneGap open-source library to produce the final native application. PhoneGap is a framework that uses a native application to launch an embedded [9]Webkit based browser which can then run a portable HTML/JS application. The advantage of this type of development is that the JavaScript based application can be debugged in the normal desktop browser before it is moved on to the mobile platform. The disadvantage is that in order to get a native looking application, the CSS has to be written for each of the platforms to get the correct look and feel, though some of this CSS comes with the PhoneGap platform (though it does contain Webkit CSS extensions for various kinds of gradient fill etc). PhoneGap does allow the JavaScript to call out into native code via a url based protocol, so a portable application can go native in places to implement things that require the speed or closeness to the platform of native code.

[10]PhoneGap is also making its way onto the Windows Mobile, and it will be interesting to see at what point it is sensible to switch from a native application written in Silverlight from one written in HTML/JS. There is a (very biased in my opinion) article comparing the two platforms [11]here.

1. <http://www.itwriting.com/blog/4447-considering-windows-8-as-an-html-platform.html>
2. [http://www.amazon.co.uk/HTML5-Up-Running-Mark-Pilgrim/dp/0596806027/ref=sr\\_1\\_1?ie=UTF8&qid=1311512486&sr=8-1](http://www.amazon.co.uk/HTML5-Up-Running-Mark-Pilgrim/dp/0596806027/ref=sr_1_1?ie=UTF8&qid=1311512486&sr=8-1)
3. <http://www.modernizr.com/>
4. <http://code.google.com/apis/gears/>
5. <http://www.html5rocks.com/en/tutorials/workers/basics/>
6. <http://channel9.msdn.com/Events/MIX/MIX11/HTM03>
7. <http://ofps.oreilly.com/titles/9780596805784/index.html>
8. <http://www.phonegap.com/>
9. <http://www.webkit.org/>
10. <http://www.phonegap.com/presentations>
11. <http://blogs.msdn.com/b/eternalcoding/archive/2011/06/13/html-5-vs-silverlight-5.aspx>

---

## 5.7.5 Camera! Action! (2011-07-31 07:12)

[1]Silverlight 4 in Action by Pete Brown

Silverlight is certainly a large topic to cover and this book stretches to around 750 pages. At many points you get the feeling that there is more that the author would like to say but is forced by size constraints to only give a brief introduction. The chapter on MVVM is a really good example of this.

The book is really well organised into three sections. The first section can be broadly divided into two parts, with the initial part covering the application model, XAML syntax, integration with the browser and the new ability of Silverlight to go out-of-browser (which offers a pretty good deployment story for in-house applications). The second part of the first section covers the rendering model, layouts and panels, controls and user controls, and input. There's enough detail in these parts to give you a good understanding of what's going on when a Silverlight application runs.

The second section, entitled "Structuring Your Application", covers data binding – though this is so fundamental that it is surprising it is not in the first section of the book, forms and grids (the staple GUI elements for business applications), input validation, navigation within the application, MVVM for structuring the application, and network and communications which includes a large amount on WCF RIA services which allow the front end Silverlight application to get its data from a backend server. This section is really the guide on writing typical business applications using this technology.

The third section, "Completing the experience", offers a whole series of extras, covering graphics and effects, printing, media display and capture, and working with bitmaps, all of which is intermixed with some more WPF-like material on animation and behaviours, resources, styles and control templates, and information on creating your own layouts and panels.

The book has such broad coverage that it is a pity that the author can't go into more depth in certain places. For example, he describes the rendering pipeline really well, and it would be really good to have even more detail about this subsystem.

Throughout the book, the examples are all well chosen and illustrate beautifully the points that are covered in the various sections. This book is certainly a really informative read and set me up nicely to go to other books for more in-depth material on the various topics.

1. [http://www.amazon.co.uk/Silverlight-4-Action-Pete-Brown/dp/1935182374/ref=sr\\_1\\_1?ie=UTF8&qid=1312093871&s=r=8-1](http://www.amazon.co.uk/Silverlight-4-Action-Pete-Brown/dp/1935182374/ref=sr_1_1?ie=UTF8&qid=1312093871&s=r=8-1)
- 

## 5.8 August

### 5.8.1 Pieces of eight (2011-08-09 16:46)

I've only just found the Windows Weekly podcast and have only listened to a couple of episodes, but there were a lot of reasonable predictions concerning Windows 8 in the [1]first episode that I listened to.

Microsoft haven't released an agenda yet for the coming [2]Build conference which has led to a lot of [3]rumours about the future of .NET and Silverlight so it was good to hear a plausible

sounding position while we wait for the conference and the full story.

1. <http://twit.tv/ww211>
  2. <http://www.buildwindows.com/>
  3. <http://azurecoding.net/blogs/brownie/archive/2011/06/11/silverlight-the-rumors-of-my-death-have-been-greatly-exaggerated.aspx>
- 

### 5.8.2 I'll need to reflect on that (2011-08-10 16:51)

I do most of my programming these days in C #, though I have ended up learning a little Visual Basic over the years. It never ceases to amaze me how IL manages to support two languages with very different features.

I was surprised to find out the other day that VB allows you to branch from an exception handler back into the code that the handler is guarding, allowing you to write an infinite loop of the form.

```
Sub Main()
Try
Console.WriteLine("start")
restart:
Throw New Exception()
Catch ex As Exception
Console.WriteLine("Handled")
GoTo restart
End Try
End Sub
```

C # doesn't allow that kind of branching.

```
static void Main(string[] args)
{
try
{
Console.WriteLine("start");
Restart:
throw new Exception();
}
catch (Exception)
{
Console.WriteLine("Handled");
goto Restart; // Invalid
}
}
```

---

### **5.8.3 No live animals here (2011-08-11 16:53)**

There's a talk [1]here that contains some material on Roslyn, the compiler as a service feature that will be exposed in a future version of Visual Studio. The talk focuses on the potential for Refactoring, but this feature offers a lot more.

1. <http://research.microsoft.com/apps/video/default.aspx?id=152281>
- 

### **5.8.4 This C is Kilim me (2011-08-12 16:55)**

Memory models are just too complicated at the [1]user level and even compiler implementers have a hard time getting their [2]JITs to work correctly.

Structuring code as series of single threaded objects, typically using Agents, has always seemed like a really good idea, though writing code in a suitable style to handle the inversion of control has always been a problem.

[3]Kilim is a Java library that has a couple of novel solutions to the problems of using Actors. First, a post-processor recognises annotations in the Java code and rewrites the byte code to handle the inversion of control aspect. @Pausable methods are rewritten to take an extra parameter of type Fiber (a lightweight thread) which is used for storing the current state when the current execution needs to wait, allowing the computation to be restarted at the later point. This rewriting implements a kind of coroutine, allowing lightweight threading and also allowing an OS thread to be shared by many computations with very low overhead. This is the kind of transformation that Microsoft are doing in their C # and VB compilers to implement the new async feature for C # 5, though in the Microsoft case the signatures of the methods need to be modified to return a Task of the appropriate type and the waiting is carried out by adding new contextual keywords to the language.

The second novel idea is to ensure that messages are not being shared between Actors, effectively isolating them. Messages are instances of Java classes, but are treated specially by the system which analyses method annotations to ensure that fields within the message are not being shared between threads. The idea here is that a method can be annotated to say what it does with a message. This annotation can be checked by the post processor and the system can therefore ensure that when a message has been passed into another method, the current method can no longer access it.

There's a Google Tech talk on Kilim [4]here and a paper on the [5]design. Some discussion on this library and whether it solves any problems (or just introduces more) can be found [6]here.

1. <http://www.rdrop.com/users/paulmck/scalability/paper/whymb.2010.06.07c.pdf>
2. <http://research.microsoft.com/pubs/76524/tr-2008-171-latest-03-11-09.pdf>
3. <http://www.malhar.net/sriram/kilim/>
4. <http://www.youtube.com/watch?v=37NaHRE0Sqw>

- 
5. [http://www.malhar.net/sriram/kilim/kilim\\_ecoop08.pdf](http://www.malhar.net/sriram/kilim/kilim_ecoop08.pdf)
  6. <http://brixomatic.wordpress.com/2010/11/26/java-actor-framework-kilim-a-solution-without-a-problem/>

### **5.8.5 You say tomato, I say tomato (2011-08-13 17:04)**

HTML5 means lots of different things to different people. There's a really good blog post [1]here that puts these interpretations into perspective.

1. <http://blog.n0ise.net/?p=375>
- 

### **5.8.6 V1, V2, V3, V4, ... (2011-08-14 17:06)**

V8 seems to doing great things for Javascript performance, and there is an interesting series of blog posts on how it does its stuff [1]here. V8 borrows a lot of techniques from the implementation of Self, a prototype based language from the mid-80s. The Self compiler was very focussed on inlining type specialised methods into other methods in order to gain performance, basing this on the observation that for a typical program, most calls happen with a fixed set of types. This leads to the need to de-optimise code when a call is made that doesn't use the specialised types, and some of the posts in the series looks at the [2]de-optimisation at the [3]assembly language level.

There is also an interesting discussion of [4]JIT security implications and the [5]relationship between SSA and CPS forms of intermediate languages.

1. <http://wingolog.org/archives/2011/08/02/a-closer-look-at-crankshaft-v8s-optimizing-compiler>
  2. <http://wingolog.org/archives/2011/06/08/what-does-v8-do-with-that-loop>
  3. <http://wingolog.org/archives/2011/06/20/on-stack-replacement-in-v8>
  4. <http://wingolog.org/archives/2011/06/21/security-implications-of-jit-compilation>
  5. <http://wingolog.org/archives/2011/07/12/static-single-assignment-for-functional-programmers>
- 

### **5.8.7 What do you attribute that to? (2011-08-17 08:00)**

Attributes are used in .NET as an extension mechanism, allowing methods and classes to have extra information added to them, which can be picked up by other tools. Some of these attributes affect that runtime itself - for example, some are used to control the marshalling behaviour during calls to functions in other unmanaged dlls.

They sound like a really good idea, but the Reactive extensions team have [1]run into a problem when their library is used on Windows Phone. They need to use thread local storage, which they do by adding an attribute to some static fields. Unfortunately Windows Phone doesn't respect this and just silently ignores the attribute, leading to the field being shared between competing threads.

The moral: it's ok to ignore attributes, but if you're the runtime it would be useful to check for those that you don't support and warn or throw an exception when such an assembly is loaded.

1. <http://social.msdn.microsoft.com/Forums/en-US/rx/thread/e70022a6-9b71-4d44-8c0f-c2c450eab01d/>
- 

### **5.8.8 Follow the trail to the source (2011-08-23 08:00)**

Javascript is being used as a compiler target language in loads of existing compilers. This is all very well, but the resulting applications need to be debugged. If you're written an application in a particular language you'd really like to see the stack state in the original language. This is exactly what the [1]source map technology provides. This is [2]currently supported only by a FireFox extension, but it is likely that more browsers will support it in the future.

1. [https://docs.google.com/document/d/1U1RGAehQwRypUTovF1KRLpiOFze0b-\\_2gc6fAHOKY0k/edit?pli=1](https://docs.google.com/document/d/1U1RGAehQwRypUTovF1KRLpiOFze0b-_2gc6fAHOKY0k/edit?pli=1)
  2. <http://www.infoq.com/news/2011/08/debug-languages-on-javascript-vm>
- 

## **5.9 September**

### **5.9.1 Behind you! (2011-09-04 08:16)**

There have been quite a number of blog posts lately on the subject of WebGL, an embedding of OpenGL into web browsers.

I started trying to learn about this by reading the [1]excellent article by Bartek Drozdz in .NET magazine. Of course, I wanted to put some of this into practice, and I noticed an [2]add-in for IE that is supposed to implement a large chunk of the WebGL specification. Unfortunately I couldn't get this to work. In fact some of the documentation is wrong; for example, the WebHelper code is in a Scripts directory of the main site and not at the top level which is where the Javascript sample of the Developer page links to.

After another failed attempt to get it going in Firefox, I eventually settled on Chrome as the browser to use for experimenting with this stuff. At around the same time, I found the impressive tutorials at [3]learningwebgl.com which take you through both OpenGL and how to use it

via WebGL with code in Javascript. I'd not really used OpenGL before, but these tutorials are aimed at programmers in exactly this position, and they are really good.

There are, of course, security repercussions associated with exposing OpenGL to web pages. The attack surface is increased as clients can now target flaws in the device drivers, and attempt to get data off the graphics card which is associated with another operating system process. One security company has posted some initial work in the following blog posts [4]here and [5]here.

Microsoft has DirectX positioned as a competitor to OpenGL, and [6]this blog post covers the history of the history of the development of these technologies and why they came about.

1. <http://www.netmagazine.com/tutorials/get-started-webgl-draw-square>
  2. <http://www.iewebgl.com/>
  3. [http://learningwebgl.com/blog/?page\\_id=1217](http://learningwebgl.com/blog/?page_id=1217)
  4. <http://www.contextis.com/resources/blog/webgl2/>
  5. <http://www.contextis.com/research/blog/webgl/>
  6. <http://m.tomshardware.com/reviews/opengl-directx,2019.html>
- 

### 5.9.2 You made me... now tell me what to do (2011-09-05 17:54)

The question came up at work as to why a warning was issued against the call to DoSomething in the following code.

```
public class A{public A(){DoSomething();} protected virtual void DoSomething(){}
}
```

The answer is, of course, that the virtual call can end up calling back into a derived class before it has fully finished initialising. For example, the following derived class

```
public class B : A
{
    int x;
    public B()
    {
        x = 33;
    }
    protected override void DoSomething()
    {
        Console.WriteLine(x);
    }
}
```

when called as

```
var b = new B();
```

ends up printing 0, as the virtual class runs one of the methods inside B before the constructor has finished running.

This brings up the interesting point that generally constructors should prepare a class for use, but should probably leave complicated actions such as subscribing to events or complicated setup to a subsequent initialization method call. Indeed, if a constructor throws an exception, then it is easy for a finaliser action to see a partially constructed object causing the finaliser to fail in interesting ways.

---

### 5.9.3 Down to the MIL (2011-09-05 21:50)

I've never really spent a lot of time looking at WPF, but the recent discussions about HTML5 and whether Silverlight still has a place in the application world have encouraged me to have a closer look at it. Declarative GUI has got to be the future, so I thought it would be interesting to see how much of WPF is implemented in managed code and to get an idea of where the boundary is between the managed classes and the unmanaged code which drive DirectX.

In a previous job, where I worked on a Common Lisp programming environment, there was a portable GUI library named [1]CAPI which allowed the programmer to write cross-platform GUIs. The trick was to implement most of layout and widgets in the managed language (Common Lisp) and then have various backends that interacted with the unmanaged world in the form of the various graphics system. This worked well for Motif, Windows, Cocoa and GTK+.

I was therefore interested to see if WPF has the same pattern - implement the front end of the library in managed code, and then pass data off to the unmanaged world to actually do the rendering. I took my copy of [2]Reflector Pro, decompiled the various assemblies and started stepping through a simple code example.

The world of WPF is slightly different. WPF doesn't run on top of another windowing system, and therefore doesn't need to translate the application windows into a series of widgets that run on top of particular GUI library. Instead it does all of the rendering itself, allowing a series of much better effects such as real transparent backgrounds, hence allowing irregular shaped items. The front end code in C# takes care of managing the layout and items on the screen, and passes off the rendering and composition to an unmanaged component called the Media Integration Layer, shortened to MIL in the code base. The MIL does still know about a series of widget type objects, and in WPF, front end and back end link together via the Visual class. If we take the example of a Grid that contains a button and a TextBox,

```
<Window x:Class="WpfApplication6.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Button Content="Button" Height="22" HorizontalAlignment="Left" Margin="180,176,0,0"
            Name="button1" VerticalAlignment="Top" Width="68" Click="button1_Click" />
        <TextBox Height="23" HorizontalAlignment="Left" Margin="84,52,0,0"
            Name="textBox1" VerticalAlignment="Top" Width="120" />
    </Grid>
</Window>
```

Using Reflector Pro to trace the execution, we can see that a proxy for the Button is generated when the CreateOrAddRefOnChannel method is used in the System.Windows.Media.Composition.VisualProxy class. This method takes three parameters. The first is the instance which is the Button object, the third is the resourceType which has value DUCE.resourceType.TYPE\_VISUAL, and the communication is being passed along a DUCE.Channel which is the remaining argument. The resource is represented using an instance of the DUCE.ResourceHandle struct.

This is all driven by the RenderRecursive method in System.Windows.Media.Visual, which is triggered by a need to resize the hosting window.

```
System.Windows.Media.Composition.VisualProxy.CreateOrAddRefOnChannel
System.Windows.Media.Visual.AddRefOnChannelCore
System.Windows.Media.Visual.System.Windows.Media.Composition.DUCE.IResource .AddRef-
OnChannel
System.Windows.Media.Visual.RenderRecursive
System.Windows.Media.Visual.UpdateChildren
System.Windows.Media.Visual.RenderRecursive
System.Windows.Media.Visual.UpdateChildren
System.Windows.Media.Visual.RenderRecursive
System.Windows.Media.Visual.UpdateChildren
System.Windows.Media.Visual.RenderRecursive
System.Windows.Media.Visual.UpdateChildren
System.Windows.Media.Visual.RenderRecursive
System.Windows.Media.Visual.UpdateChildren
System.Windows.Media.Visual.RenderRecursive
System.Windows.Media.Visual.UpdateChildren
System.Windows.Media.Visual.RenderRecursive
System.Windows.Media.Visual.Render
System.Windows.Media.CompositionTarget.Compile
System.Windows.Media.CompositionTarget.System.Windows.Media.ICompositionTar-
get.Render
System.Windows.Media.MediaContext.Render
System.Windows.Media.MediaContext.RenderMessageHandlerCore
System.Windows.Media.MediaContext.RenderMessageHandler
System.Windows.Media.MediaContext.Resize
System.Windows.Interop.HwndTarget.OnResize
System.Windows.Interop.HwndTarget.HandleMessage
System.Windows.Interop.HwndSource.HwndTargetFilterMessage
```

The HandleMessage function in the above is processing windows messages on the main window, and then dispatching on the message. In the above case we are handling the WM\_SIZE message.

On subsequent runs, the various properties of the unmanaged object are updated by calls in the RenderRecursive method which contains the code:

```
this.UpdateCacheMode(channel, @null, none, isOnChannel);
this.UpdateTransform(channel, @null, none, isOnChannel);
this.UpdateClip(channel, @null, none, isOnChannel);
this.UpdateOffset(channel, @null, none, isOnChannel);
this.UpdateEffect(channel, @null, none, isOnChannel);
this.UpdateGuidelines(channel, @null, none, isOnChannel);
this.UpdateContent(ctx, none, isOnChannel);
this.UpdateOpacity(channel, @null, none, isOnChannel);
```

```

this.UpdateOpacityMask(channel, @null, none, isOnChannel);
this.UpdateRenderOptions(channel, @null, none, isOnChannel);
this.UpdateChildren(ctx, @null);
this.UpdateScrollableAreaClip(channel, @null, none, isOnChannel);
this.SetFlags(channel, false, VisualProxyFlags.None |
VisualProxyFlags.IsSubtreeDirtyForRender );

```

There's one other complication in the WPF world. WPF allows the user to override the rendering that the system does in order to do user drawn objects around the standard drawing. We can see the MIL in operation at this level too.

```

public class MyTextBox : TextBox
{
protected override void OnRender(DrawingContext drawingContext)
{
base.OnRender(drawingContext);
drawingContext.DrawEllipse(Brushes.Bisque,
new Pen(Brushes.Bisque, 2.0), new Point(0, 0), 3.0, 5.0);
}
}

```

and modify the TextBox in the earlier example to

```

<WpfApplication6:MyTextBox Height="23" HorizontalAlignment="Left"
Margin="84,52,0,0" x:Name="textBox1" VerticalAlignment="Top"
Width="120" />

```

If we now add a button handler, and add the following code into it.

```

private void button1_Click(object sender, RoutedEventArgs e)
{
DrawingGroup group = VisualTreeHelper.GetDrawing(textBox1);
string output = System.Windows.Markup.XamlWriter.Save(group);
}

```

we see the textual representation of the rendering we did within the drawing context.

```

<DrawingGroup xmlns=[5]http://schemas.microsoft.com/winfx/2006/xaml/presentation>
<DrawingGroup.Children>
<GeometryDrawing Brush="#FFFFE4C4">
<GeometryDrawing.Pen>
<Pen Brush="#FFFFE4C4" Thickness="2" />
</GeometryDrawing.Pen>
<GeometryDrawing.Geometry>
<EllipseGeometry RadiusX="3" RadiusY="5" Center="0,0" />
</GeometryDrawing.Geometry>
</GeometryDrawing>
</DrawingGroup.Children>
</DrawingGroup>

```

What the VisualTreeHelper does in this case is quite interesting. The system goes to the GetDrawing method defined in System.Windows.UIElement, which calls into DrawingServices.DrawingGroupFromRenderData. This uses an object to type DrawingContextDrawingContextWalker which walks a data structure that encodes the extra drawing commands we did on top of the button.

How does the RenderData arrive in place? The DrawingContext passed into our OnRender method is of type RenderDataDrawingContext, and each of the drawing methods that can be called are responsible for allocating a data structure that records (in the managed world) the details of the drawing command, and moreover record this in a form that can be passed as a stream of bytes to the unmanaged layer.

```
public override unsafe void DrawEllipse(  
Brush brush, Pen pen, Point center, double radiusX, double radiusY)  
{  
this.VerifyApiNonstructuralChange();  
if ((brush != null) || (pen != null))  
{  
this.EnsureRenderData();  
MILCMD_DRAW_ELLIPSE milcmd_draw_ellipse =  
new MILCMD_DRAW_ELLIPSE(  
this._renderData.AddDependentResource(brush),  
this._renderData.AddDependentResource(pen),  
center, radiusX, radiusY);  
this._renderData.WriteDataRecord(  
MILCMD.MilDrawEllipse, (byte*) &milcmd_draw_ellipse, 40);  
}  
}
```

The RenderData produces a structure that records the information about the rendering, and buffers it in a byte array in the RenderData.WriteDataRecord method. This is pushed out to the rendering system in the call to this.UpdateContent which we saw above, which calls RenderData.MarshalToDUCE. This is quite tidy – the data can be passed to the unmanaged world as extra drawing commands, but stays around in the managed world and can be walked using a visitor pattern to reconstruct the drawing as we did in the code above.

In fact, RenderContext is used by more than just the user code. If you take the first version of the code which simply used a WPF Button, you'll find that a TextBlock is used to hold the textual part of the button. When the TextBlock has its UpdateContent method called in the RenderRecursive (which we saw above), the

```
this.UpdateContent  
calls through to the method  
private void UpdateContent(  
RenderContext ctx, VisualProxyFlags flags, bool isOnChannel)  
{  
if ((flags & VisualProxyFlags.IsContentDirty) != VisualProxyFlags.None)  
{  
this.RenderContent(ctx, isOnChannel);  
this.SetFlags(ctx.Channel, false, VisualProxyFlags.IsContentDirty);  
}  
}
```

which uses the RenderContent on UIElement. This again uses a stored DrawingContext

```
internal override void RenderContent(RenderContext ctx, bool isOnChannel)  
{  
DUCE.Channel channel = ctx.Channel;
```

```

if (this. _drawingContent != null)
{
    DUCE.IResource resource = this. _drawingContent;
    resource.AddRefOnChannel(channel);
    DUCE.CompositionNode.SetContent(this. _proxy.GetHandle(channel),
        resource.GetHandle(channel), channel);
    base.SetFlags(channel, true, VisualProxyFlags.IsContentConnected);
}
else if (isOnChannel)
{
    DUCE.CompositionNode.SetContent(this. _proxy.GetHandle(channel),
        DUCE.ResourceHandle.Null, channel);
}
}

```

which is placed there when `OnRender` method of `System.Windows.Controls.TextBlock` is called. This method uses the various drawing primitives to render the text block.

In contrast, Silverlight has a different set of objects that work across the boundary. `MS.Internal.ManagedPeerTable` maintains a link between `IntPtr` from the unmanaged world and the managed objects that they represent. In the Silverlight case the unmanaged equivalent of the `Button` is generated in the constructor of the `Button`.

```

MS.Internal.ManagedPeerTable.Add
System.Windows.DependencyObject.DependencyObject
System.Windows.Controls.ContentControl.ContentControl
System.Windows.Controls.Primitives.ButtonBase.ButtonBase
System.Windows.Controls.Button.Button
[Managed to Native Transition]
System.RuntimeType.CreateInstanceSlow
System.Activator.CreateInstance
MS.Internal.XamlManagedRuntimeRPIInvokes.CreateInstance
[Managed to Native Transition]
MS.Internal.XcpImports.Application_LoadComponent
System.Windows.Application.LoadComponent
SilverlightApplication7.MainPage.InitializeComponent
SilverlightApplication7.MainPage.MainPage
SilverlightApplication7.App.Application_Startup
MS.Internal.CoreInvokeHandler.InvokeEventHandler
MS.Internal.JoltHelper.FireEvent

```

The initial cause of the instantiation is the incoming event at the bottom of the stack.

```
internal static uint FireEvent(IntPtr unmanagedObj, IntPtr unmanagedObjArgs, int argsTypeIndex, int actualArgsTypeIndex, [MarshalAs(UnmanagedType.LPWStr)] string eventName)
```

The event comes into the system as a string "M@5" - this is parsed to get the value 5 which is converted into an appropriate delegate. In this case we find something

```

instanceEventDelegate
{System.Windows.StartupEventHandler }
[System.Windows.StartupEventHandler]: {System.Windows.StartupEventHandler }
_methodBase: null

```

```
_methodPtr: 87408664  
_methodPtrAux: 0  
_target: {SilverlightApplication7.App }
```

The dispatching to the handler is done in MS.Internal.CoreHandler.InvokeEventHandler which uses the typeIndex to decide on an action. In this case we just call into the Application \_Startup of our application.

In the Silverlight world, more of the rendering has been moved into the unmanaged code. In WPF, there are lots of places where RenderData is generated to describe the content, allowing user and system rendering to happen for a given element. This kind of customisation isn't available in the Silverlight world. This lack of flexibility means that the amount of managed code is a lot smaller.

In the days of the Common Lisp system that I worked on, we actively tried to move as much code as possible into the managed world. Common Lisp is in a fairly unique position in that the system could be patched dynamically, and the more code that was managed, the more code we could change by issuing patches, rather than having to redeploy the managed and unmanaged parts.

In some ways it is a shame that all of the composition code is unmanaged. I assume this was done to allow the MIL to be shared with other unmanaged components like the window manager. Some of the Channel9 videos hint that there was an intention to make the MIL (unmanaged) interface accessible by user code, though I don't believe that has happened yet. It is quite impressive to see the buffering of the RenderData inside the managed world inside byte vectors that can be efficiently pushed into the unmanaged world, but Silverlight needs to be small and fast, so this flexibility has been removed from the platform.

1. <http://www.lispworks.com/documentation/lw60/CAPUG-U/html/capiuser-u-6.htm#pgfId-884168>
  2. <http://www.reflector.net/vspro/>
  3. <http://schemas.microsoft.com/winfx/2006/xaml/presentation>"
  4. <http://schemas.microsoft.com/winfx/2006/xaml>"
  5. <http://schemas.microsoft.com/winfx/2006/xaml/presentation>
- 

#### 5.9.4 What type of event were you expecting? (2011-09-07 22:10)

We looked last time at how the rendering flowed between the managed and unmanaged worlds when running WPF and Silverlight. For any sort of useful application, of course, there's also the question of events and how events are captured and routed.

Taking the same XAML as last time.

```
<Window x:Class="WpfApplication6.MainWindow"  
       xmlns="*[1]http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
       xmlns:x="*[2]http://schemas.microsoft.com/winfx/2006/xaml"  
       Title="MainWindow" Height="350" Width="525">  
<Grid>  
<Button Content="Button" Height="22"
```

```

HorizontalAlignment="Left" Margin="180,176,0,0" Name="button1"
VerticalAlignment="Top" Width="68" Click="button1_Click" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="84,52,0,0"
Name="textBox1" VerticalAlignment="Top" Width="120" />
</Grid>
</Window>

```

This example contains a button event handler, and when we breakpoint the click handler click the button we can see the following stacktrace.

```

WpfApplication6.MainWindow.button1_Click
System.Windows.RoutedEventArgs.InvokeHandler
System.Windows.EventRoute.InvokeHandlersImpl
System.Windows.UIElement.RaiseEventImpl
System.Windows.UIElement.RaiseEvent
System.Windows.Controls.Primitives.ButtonBase.OnClick
System.Windows.Controls.Button.OnClick
System.Windows.Controls.Primitives.ButtonBase.OnMouseLeftButtonUp
System.Windows.UIElement.OnMouseLeftButtonUpThunk
System.Windows.Input.MouseEventArgs.InvokeEventHandler
System.Windows.RoutedEventArgs.InvokeHandler
System.Windows.RoutedEventArgs.InvokeHandler
System.Windows.RoutedEventArgs.InvokeHandlerInfo.InvokeHandler
System.Windows.EventRoute.InvokeHandlersImpl
System.Windows.UIElement.ReRaiseEventAs
System.Windows.UIElement.OnMouseUpThunk
System.Windows.Input.MouseEventArgs.InvokeEventHandler
System.Windows.RoutedEventArgs.InvokeHandler
System.Windows.RoutedEventArgs.InvokeHandler
System.Windows.RoutedEventArgs.InvokeHandlerInfo.InvokeHandler
System.Windows.EventRoute.InvokeHandlersImpl
System.Windows.UIElement.RaiseEventImpl
System.Windows.UIElement.RaiseTrustedEvent
System.Windows.UIElement.RaiseEvent
System.Windows.Input.InputManager.ProcessStagingArea
System.Windows.Input.InputManager.ProcessInput
System.Windows.Input.InputProviderSite.ReportInput
System.Windows.Interop.HwndMouseInputProvider.ReportInput
System.Windows.Interop.HwndMouseInputProvider.FilterMessage
System.Windows.Interop.HwndSource.InputFilterMessage
MS.Win32.HwndWrapper.WndProc
System.Windows.Threading.ExceptionWrapper.InternalRealCall
MS.Internal.Threading.ExceptionFilterHelper.TryCatchWhen
System.Windows.Threading.Dispatcher.InvokeImpl
MS.Win32.HwndSubclass.SubclassWndProc
[Native to Managed Transition]
[Managed to Native Transition]
System.Windows.Threading.Dispatcher.PushFrameImpl
System.Windows.Threading.Dispatcher.PushFrame
System.Windows.Threading.Dispatcher.Run
System.Windows.Application.RunDispatcher
System.Windows.Application.RunInternal
System.Windows.Application.Run

```

```

System.Windows.Application.Run
WpfApplication6.App.Main
[Native to Managed Transition]
[Managed to Native Transition]
System.AppDomain.ExecuteAssembly
Microsoft.VisualStudio.HostingProcess.HostProc.RunUsersAssembly
System.Threading.ThreadHelper.ThreadStart _Context
System.Threading.ExecutionContext.Run
System.Threading.ExecutionContext.Run
System.Threading.ThreadHelper.ThreadStart()
[Native to Managed Transition]

```

We are processing the button up message from the mouse. The message trickles into the system from the message filter handling the standard windows WM\_LBUTTONDOWN message.

```

case WindowMessage.WM_LBUTTONDOWN:
{
int num20 = MS.Win32.NativeMethods.SignedLOWORD(IParam);
int num19 = MS.Win32.NativeMethods.SignedHIWORD(IParam);
handled = this.ReportInput(hwnd, InputMode.Foreground,
this._msgTime,RawMouseActions.Button1Release, num20, num19, 0);

```

The call makes its way through to the RaiseEvent on System.Windows.UIElement. At this point the system has created a RoutedEventArgs instance to hold the data and a flag to determine if the event was user initiated. In Silverlight, for example, certain actions are only allowed if they are result of a user action. That jumps through into RaiseTrustedEvent which marks the RoutedEventArgs as user initialized.

We drop through to RaiseEventImpl on System.Windows.UIElement. This is responsible for calculating the event route.

```

internal static void RaiseEventImpl(DependencyObject sender, RoutedEventArgs args)
{
EventRoute route = EventRouteFactory.FetchObject(args.RoutedEvent);
...
try
{
args.Source = sender;
BuildRouteHelper(sender, route, args);
route.InvokeHandlers(sender, args);
args.Source = args.OriginalSource;
...

```

The actual navigation along the route is done in System.Windows.EventRoute.InvokeHandlerImpl. The code goes along the route, firing each handler in turn. We eventually get to System.Windows.UIElement.OnMouseLeftButtonUp which gets us into the Click code in the PresentationFramework. This then generates another Routed event which is routed to the various handlers, finding our code in button1\_Click.

Silverlight handles things in a much simpler way.

```

SilverlightApplication7.MainPage.button1_Click
System.Windows.Controls.Primitives.ButtonBase.OnClick
System.Windows.Controls.Button.OnClick

```

```
System.Windows.Controls.Primitives.ButtonBase.OnMouseLeftButtonUp
System.Windows.Controls.Control.OnMouseLeftButtonUp
MS.Internal.JoltHelper.FireEvent
[Appdomain Transition]
```

FireEvent is called with following signature.

```
internal static uint FireEvent(
IntPtr unmanagedObj, IntPtr unmanagedObjArgs,
int argsTypeIndex, int actualArgsTypeIndex,
[MarshalAs(UnmanagedType.LPWStr)] string eventName)
```

This quickly finds out the information about the event, and calls into OnMouseLeftButtonUp on System.Windows.Controls.Control. This jumps into the ButtonBase code which generates the OnClick event which is processed by our event handler.

The event handling on Silverlight is so much simpler. WPF behaves like a highly customizable version of the WIN32 event processing, When the left button goes down on the button, it grabs the mouse focus and waits for the left button to go up again. All of this is handled in managed code, making it easy to implement things that diverge in certain places from this behaviour. In Silverlight, a button is fixed to behaving in a much simpler fashion, and the unmanaged code basically calls straight into the Click event. In WPF, the click event is only triggered after the left button up has been handled (and finding this handler requires the event to be bubbled until something suitable is found), and is raised in a second phase after the code that tries to find a suitable handler.

1. [http://schemas.microsoft.com/winfx/2006/xaml/presentation"](http://schemas.microsoft.com/winfx/2006/xaml/presentation)
  2. [http://schemas.microsoft.com/winfx/2006/xaml"](http://schemas.microsoft.com/winfx/2006/xaml)
- 

### 5.9.5 Ask your Dad! (2011-09-10 18:29)

Something that puzzles me every time I read articles about WPF DependencyObjects, is the fact that they don't mention how property value inheritance is implemented. Inheritance is used all over the place so property values such as fonts trickle down from parents to their children. While looking at the code in [1]Reflector, I finally twigged how this happens.

Take a simple test type of object,

```
class TestType : DependencyObject
{
    private static readonly DependencyProperty s_Property =
        DependencyProperty.Register("Property", typeof(string), typeof(TestType),
        new FrameworkPropertyMetadata("boo", FrameworkPropertyMetadataOptions.Inherits));

    public string Property
    {
        get
    }
```

```
return (string)GetValue(s _Property);
}
set
{
SetValue(s _Property, value);
}
}
}
```

Notice the `FrameworkPropertyMetadataOptions`, as that turns out to be important. Let's make two instances, which we'll assume represent a parent and a child.

```
var parent = new TestType();
var child = new TestType();
```

Of course, currently the two instances are unrelated, so the parent value isn't inherited even though the value isn't set on the child.

```
parent.Property = "hello";
Console.WriteLine(child.Property); // "boo"
```

Once we call one of the internal functions to relate the two objects (using Reflection), the inheritance starts happening.

```
MethodInfo method =
typeof(DependencyObject)
.GetMethod("SynchronizeInheritanceParent", BindingFlags.Instance | BindingFlags.NonPublic);
method.Invoke(child, new object[] { parent });
Console.WriteLine(child.Property); // "hello"
```

If you use Reflector, you can see that these internal functions are used in the WPF to set up the relevant relationships to get the values inheriting properly.

1. <http://www.reflector.net/>

---

## 5.9.6 COM is coming home (2011-09-21 15:51)

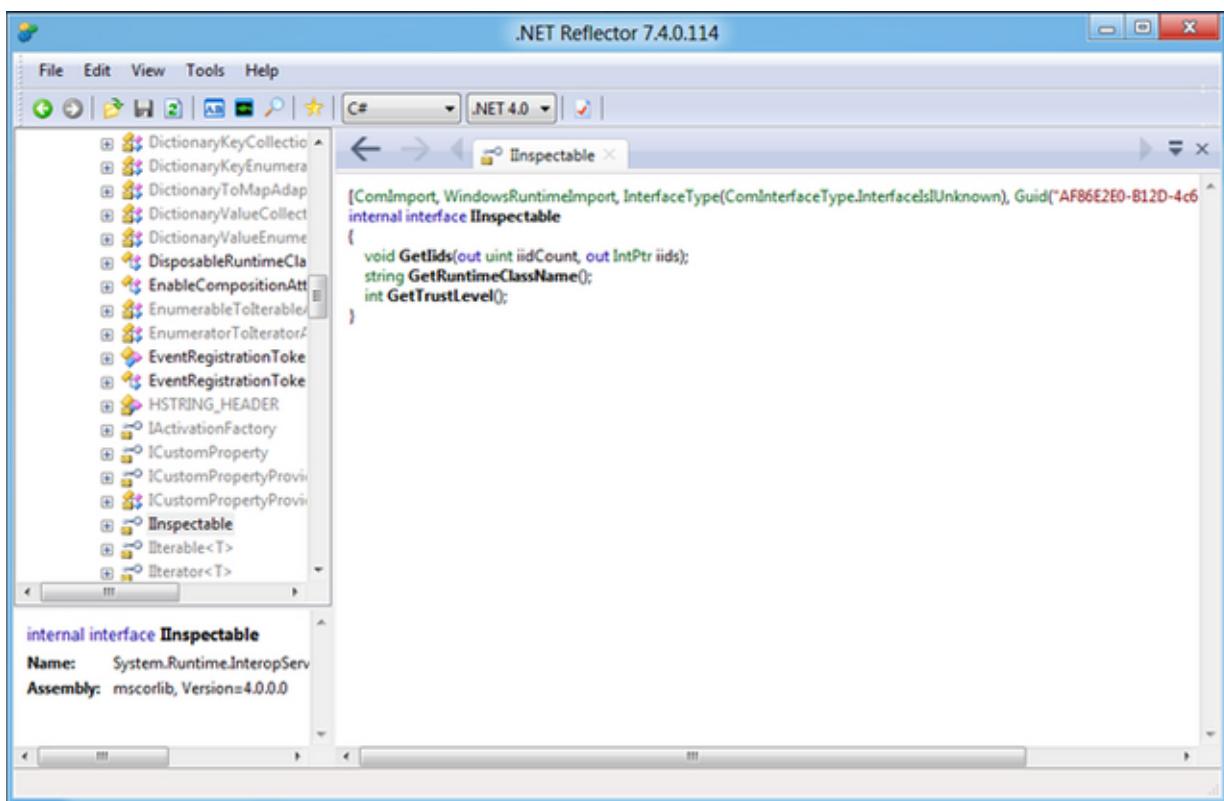
The recent [1]//BUILD/ conference changed a few things in the Windows Space.

A long time ago in the 1990s, COM was the mechanism that allowed disparate applications to communicate with each other. At the turn of the century, the focus moved towards using managed runtimes, and having multiple applications running as separated components within a single garbage collected heap, with .NET remoting and WCF allowing communication between separated heaps. Both of these technologies sat on Win32, the interface through which the operating system offered its services to applications.

In Windows 8, for metro style applications, the communication to the operating system happens via the Windows runtime, WinRT. This offers a set of COM interfaces as the means for applications to communicate with the operating system. This is not quite the COM we knew and loved in the old days; this is a COM with component metadata. Whereas the COM of old was

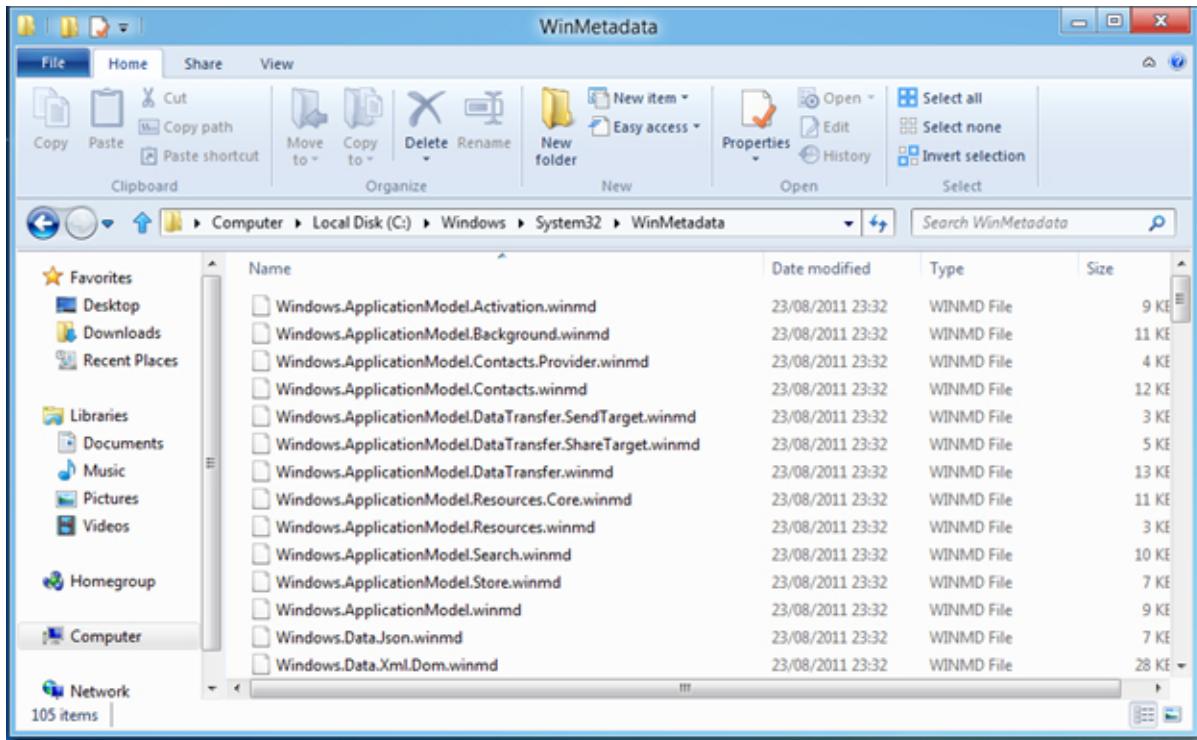
based on the `IUnknown` interface, which offered simple discoverability using `QueryInterface`, and component life cycle management via the `AddRef` and `Release` methods, the interfaces of the WinRT components are based on the `IInspectable` interface. This extends `IUnknown` with three extra methods that get the trust level for the component, get the runtime class name of a component and which ask a component to list all of the interfaces that it supports. In the old days, you could ask a component if it supported a given interface, but the basic contract didn't demand that a component could always tell you the full list of interfaces that it supported. This made it hard for applications to install a proxy in front of a given component, as you could only find out the full set of interfaces of a component by asking for all possible interfaces one by one (unless the component supported other interfaces which allowed you to query the component for type information).

[2]



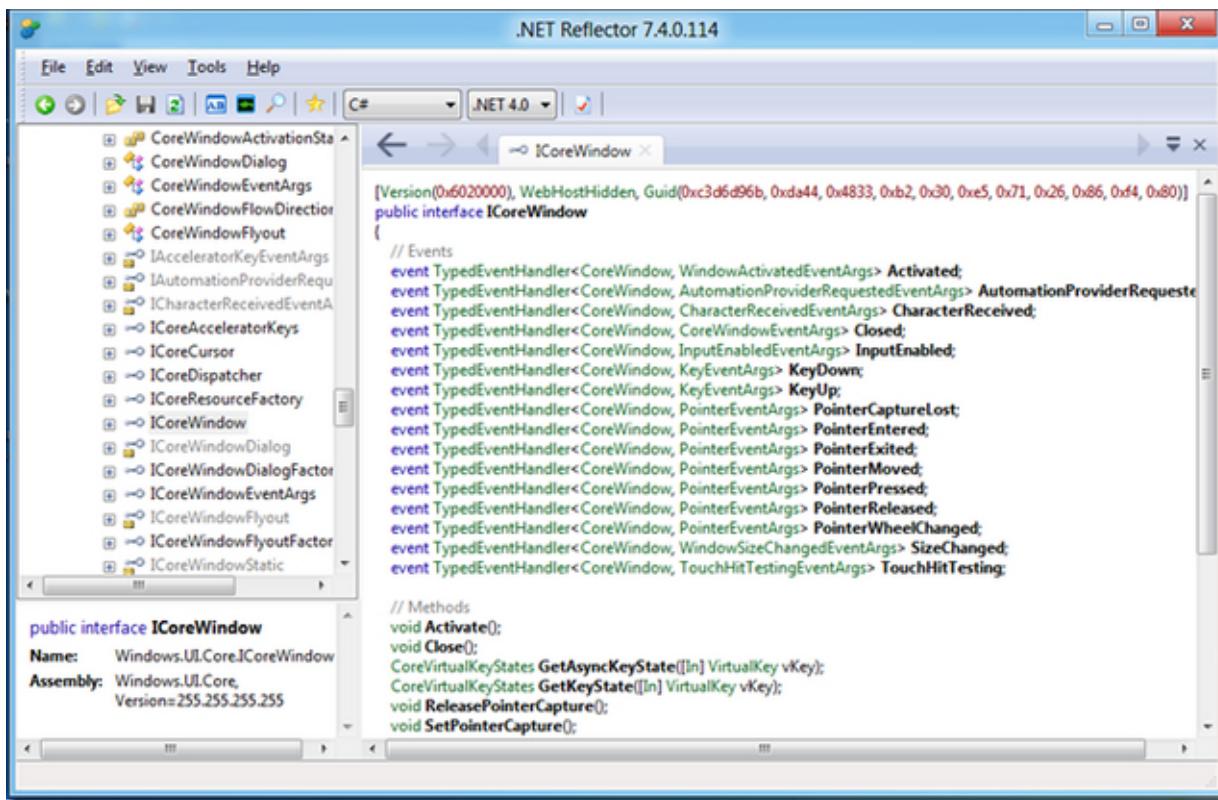
It has always been possible to use COM components from managed code, but in Windows 8 the system has been changed to make it even easier. The metadata for all of the built-in WinRT components is made available in a form that other tools can read. This allows “projection” of these components into the various languages, where the interfaces can be presented as standard classes.

[3]



This information is stored in a file format that corresponds to slightly modified CLR metadata, which can be read by a tool such as Reflector.

[4]



The difficulty with playing with this from inside Visual Studio is that the `IInspectable` type is internal to mscorelib. However, that doesn't really stop us as we can use Reflector to find its

definition and then redefine it ourselves. Note that we are going to have some problems here – the GetRuntimeClassName method is specified as returning a string, but in reality it is returning another special string type which is built into WinRT, an HString. We'll come to that later, but will deal with it by accepting a raw pointer back from calls into the interface.

[5]

```
[System.Runtime.InteropServices.ComImport,
System.Runtime.InteropServices.InterfaceType(System.Runtime.InteropServices.ComInterfaceType.InterfaceIsIUnknown),
System.Runtime.InteropServices.Guid("AF86E2E0-B12D-4c6a-9C5A-D7AA65101E90")]
internal interface IMyInspectable
{
    void GetIids(out IntPtr iidCount, out IntPtr iids);
    IntPtr GetRuntimeClassName();
    int GetTrustLevel();
}
```

We make an instance of one of the WinRT standard components. We convert it into a standard COM IUnknown and then force that to be an instance of our redefined IInspectable (IMyInspectable). We can then call the standard methods on that.

Our main difficulty is that the GetRuntimeClassName returns a WinRT string, an HString. We can find this in the hstring.h header file where we're told that the string header takes 24 bytes on x64 (and 20 on x86). We therefore get the name by reading characters from this offset into the string.

The three IIDs, the interface IDs that the File Picker supports, are written as consecutive bytes starting at the location pointed to by iidData.

[6]

```
var fileOpenPicker = new Windows.Storage.Pickers.FileOpenPicker();

var inspectable =
    (IMyInspectable)Marshal.GetTypedObjectForIUnknown(
        Marshal.GetIUnknownForObject(fileOpenPicker),
        typeof(IMyInspectable));

IntPtr result = inspectable.GetRuntimeClassName();
string theName = System.Runtime.InteropServices.Marshal.PtrToStringUni(result + 24);

int trustLevel = inspectable.GetTrustLevel();

IntPtr countIids;
IntPtr iidData;
inspectable.GetIids(out countIids, out iidData);

var guids = new List<string>();
int offset = 0;
for (int count = 0; count < countIids.ToInt32(); count++)
{
    var guid = GetGuidFromOffset(iidData, offset);
    offset += 16;
    guids.Add(guid.ToString());
}
```

We can see that the file picker supports three COM interfaces, one of which is IInspectable. Using the registry, we can look up the other IIDs to find the names \_\_x\_Windows\_CStorage\_CPickers\_CIFileOpenPicker and IWeakreferenceSource.

[7]

Watch 1		
Name	Value	Type
theName	"Windows.Storage.Pickers.FileOpenPicker"	string
trustLevel	1	int
guids	Count = 3	System.C
Capacity	4	int
Count	3	int
Static members		
Non-Public members		
_items	{string[4]}	string[]
[0]	"AF86E2E0-B12D-4C6A-9C5AD7AA65101E90"	string
[1]	"00000038-0000-0000-C000000000000046"	string
[2]	"2CA8278A-12C5-4C5F-897794547793C241"	string

Note that we should be releasing the unmanaged memory and the component references that we have generated. We'd do the latter using the standard Marshal.Release method and the former using special API functions.

I used the following code for generating the string form of the Guids.

[8]

```
private StringBuilder GetGuidFromOffset(IntPtr iidData, int offset)
{
    var guid = new System.Text.StringBuilder();
    NextBlock(guid, iidData, offset + 3, 4, -1);
    guid.Append("-");
    NextBlock(guid, iidData, offset + 5, 2, -1);
    guid.Append("-");
    NextBlock(guid, iidData, offset + 7, 2, -1);
    guid.Append("-");
    NextBlock(guid, iidData, offset + 8, 8, 1);
    return guid;
}

private void NextBlock(StringBuilder guids, IntPtr data, int offset, int size, int direction)
{
    for (int k = 0; k < size; k++)
    {
        guids.AppendFormat("{0:X2}", Marshal.ReadByte(data, offset));
        offset += direction;
    }
}
```

I think it is quite fascinating how XAML used to be processed by managed code and then converted into a set of unmanaged objects, which I covered in some earlier blog posts. In this new setup, the unmanaged code parses the XAML and then it is the managed code which hooks up with the unmanaged objects.

So, for example, the InitializeComponent of a Metro Silverlight application looks like

[9]

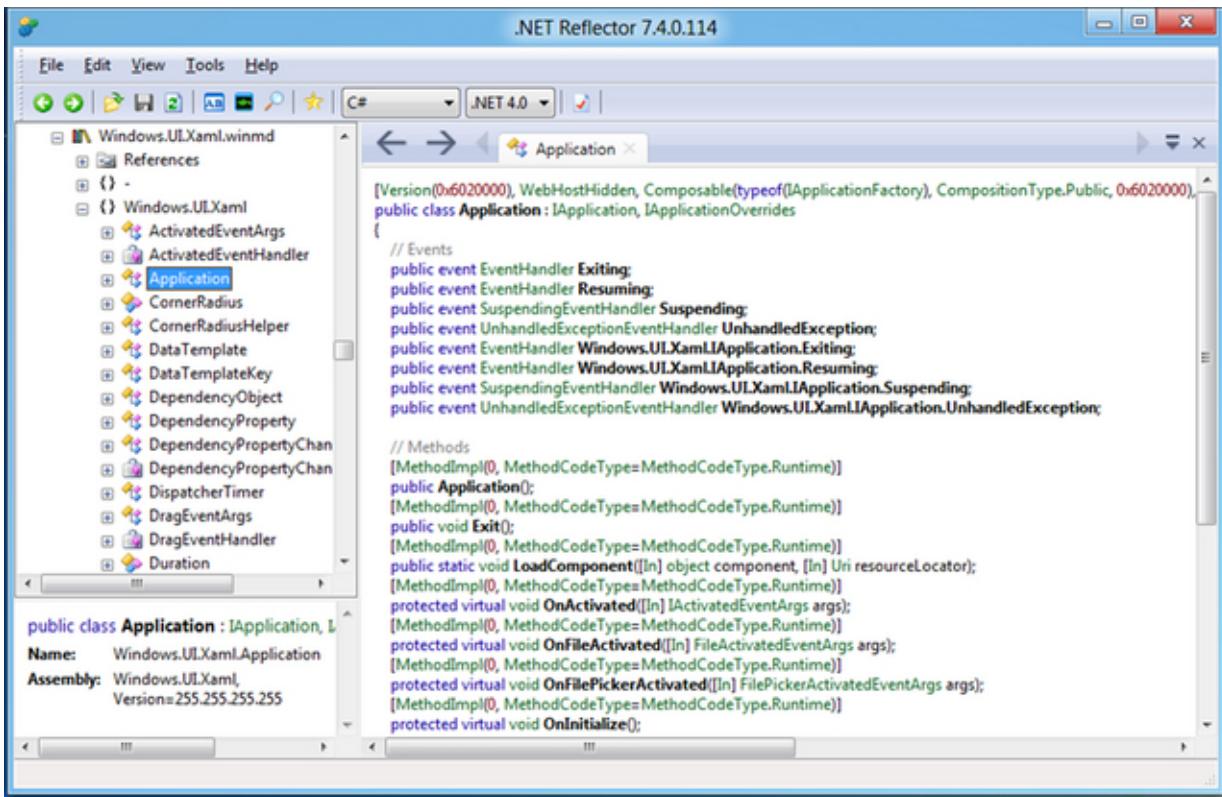
```
public void InitializeComponent()
{
    if (_contentLoaded)
        return;

    _contentLoaded = true;
    Application.LoadComponent(this, new System.Uri("ms-resource://application4/Files/MainPage.xaml"));

    LayoutRoot = (Windows.UI.Xaml.Controls.Grid)this.FindName("LayoutRoot");
}
```

where it is making use of the WinRT Application object

[10]



Two excellent talks on the Windows Runtime are available [11]here and [12]here.

1. <http://www.buildwindows.com/>
2. <http://clivetong.files.wordpress.com/2011/09/pic1.png>
3. <http://clivetong.files.wordpress.com/2011/09/pic2.png>
4. <http://clivetong.files.wordpress.com/2011/09/pic3.png>
5. <http://clivetong.files.wordpress.com/2011/09/pic4.png>
6. <http://clivetong.files.wordpress.com/2011/09/picc8.png>
7. <http://clivetong.files.wordpress.com/2011/09/pic9.png>
8. <http://clivetong.files.wordpress.com/2011/09/pic10.png>
9. <http://clivetong.files.wordpress.com/2011/09/pic11.png>
10. <http://clivetong.files.wordpress.com/2011/09/pic12.png>
11. <http://channel9.msdn.com/Events/BUILD/BUILD2011/PLAT-874T>
12. <http://channel9.msdn.com/Events/BUILD/BUILD2011/PLAT-875T>

### 5.9.7 It's all about the Type of the data (2011-09-22 19:50)

There have been a lot of rumours, and lots of demos that gave hints about the type providers that were to become available in F# 3.0. At last, a developer preview is here, and they do indeed look like a very powerful technology.

Type providers allow the user to register a provider, which will be called by the F # compiler at compile-time in order for the compiler to get information about types in the F # code that it is compiling. We've had tools in the past that you can point at a data source and which generate a .NET assembly of suitable types, but this has always been disconnected from the compiler running. With F # type providers, it is the compiler that drives the discovery process, making it possible to dynamically generate types of the right shape and avoid the need to generate data about types that aren't called.

[1]This post contains the best example of a type provider implementation that I have seen so far, showing how vector types could be provided for n-dimensional vectors where the n is specified in the source code.

1. <http://www.mindscapehq.com/blog/index.php/2011/09/19/f-type-providers-as-if-by-magic/>

---

## 5.10 October

### 5.10.1 Anything you can do, I can do too. (2011-10-03 10:43)

There is a slight weakness in the WinRT diagram that Microsoft put out at //Build/. In the diagram, the managed languages are listed as supporting XAML , but only the C++/WinRT combination is listed as supporting DirectX. As far as I know, Microsoft are not going to be supporting XNA on WinRT, so there seems to be a space in the product diagram, with no particularly good reason for the gap.

The WinRT interfaces are projected into the various managed languages, so it was fairly easy to take the C++ DirectX project instance and convert the main initialisation code into C #.

[1]

```
static void Main()
{
    var myProviderFactory = new MyViewproviderFactory();
    Windows.ApplicationModel.Core.CoreApplication.Run(myProviderFactory);
}

class MyViewproviderFactory : IViewProviderFactory
{
    public IViewProvider CreateViewProvider()
    {
        return new MyViewProvider();
    }
}
```

The ViewProvider can then activate the main window, and sit there dispatching events until the user presses a key.

[2]

```

class MyViewProvider : IViewProvider
{
    Windows.UI.Core.CoreWindow m_Window;
    Windows.ApplicationModel.Core.CoreApplicationView m_ApplicationView;

    public void Initialize(Windows.UI.Core.CoreWindow window, Windows.ApplicationModel.Core.CoreApplicationView applicationView)
    {
        m_Window = window;
        m_ApplicationView = applicationView;
    }

    public void Load(string entryPoint)
    {
    }

    public void Run()
    {
        bool inputReceived = false;

        m_Window.Activate();
        m_Window.KeyDown += (a, b) => inputReceived = true;

        while (!inputReceived)
        {
            m_Window.Dispatcher.ProcessEvents(Windows.UI.Core.CoreProcessEventsOption.ProcessAllIfPresent);
        }

        m_Window.Close();
    }

    public void Uninitialize()
    {
    }
}

```

Of course, what I really wanted to do was to use DirectX to draw something more complicated to the screen. DirectX is a set of COM interfaces so in principle it should be straightforward to do this from managed code, once a set of suitable interface definitions has been written. While searching for such a set, I came across [3]SharpDX.

SharpDX claims to support the latest DirectX libraries and is already geared up to support Metro style apps under Win8. They take the DirectX C++ headers and generate the .NET wrappers from this. The Zip distribution file contains two examples. For me the simple example works fine, though that doesn't do any rendering to the screen. The example that does rendering (of a cube) throws an exception so there is obviously some work to do, though this library will eventually be able to fill the gap.

1. <http://clivetong.files.wordpress.com/2011/10/image.png>
2. <http://clivetong.files.wordpress.com/2011/10/image1.png>
3. <http://code.google.com/p/sharpdx/>

---

xoofx (2011-10-03 13:31:38)

Hi, About your bug with SharpDX and Win8, as it is working on my config, that would be interesting if you could fill a bug issue (<http://code.google.com/p/sharpdx/issues/list>) and explain a little bit more where does it crashes (you can try {} catch inside the method run and see where it stops). Thanks!

## 5.10.2 What a Functionally Perfect Day! (2011-10-19 08:53)

[1]Mark Dalgarno organised a fantastic one day conference in Cambridge last week – [2]FPDay.

To me, the day was a perfect mix of talks on functional programming together with 3 hours of hands-on time learning more about one of the three functional languages [3]Haskell, [4]Clojure and [5]F #.

The day started with a keynote from [6]Simon Peyton Jones who has devoted a large part of his career to researching functional programming ideas using the Haskell programming language. His talk was concerned with Haskell in the cloud, and was mainly concerned with adding [7]Erlang like concurrency features to Haskell (as a new library). Erlang is interesting in that it embraces failure, with processes expected to simply die if they get into an error state from which they can't recover. In order to implement this, processes are highly isolated, making the failure localised, and allowing the system as a whole to deal with the failure by having processes track the state of other processes. Isolation implies that objects are passed between the processes using object-by-value, and the talk covered how to get this working in a strongly typed languages. For most types of object, the typeclass mechanism of Haskell could be used to define serialize and deserialize functions that marshal the object into a `ByteArray` which could be passed across to the other process. The difficulty comes when you want to transfer a function between the two processes. In a functional language, the type of a function doesn't describe the types of the objects that it might close over. A function is really described by the pair of an address (of the code) together with an environment that describes the closed over variables, and it is this representation, which makes all the parts of the function explicit, which could be marshalled across to the target process. The talk discussed how this new type of object could be expressed in Haskell – there's sure to be a paper on this in the near future.

The second part of the day was a couple of 1.5 hour blocks where the attendees were divided into three streams, one for each of the functional languages listed above. I was lucky to go to the Haskell tutorial which was run by two people, Andres and Duncan, from [8]well-typed. This was a great tutorial consisting of roughly an hour of slides on the Haskell language, followed by the rest of the time working through an example application of which the attendees had to implement various parts. This example consisted of defining an agent which would ask a set of yes/no questions to guess a number that the user had chosen. The example required the attendees to understand abstract data types, and moved on to use the QuickCheck framework built into GHCi to test their code. In the second part of the example, the Shed web server was downloaded using cabal and the game was turned into a stateful web based game. In this section, the state was saved into the URL using a embedded DSL describing the grammar for the state, which could be used both for the marshalling and unmarshalling.

Haskell is a mainly pure language though input and output via the IO monad complicates the issue. The tutorial didn't have time to go into the IO monad in any detail, but there were some good discussions on IO and how it is specially built into the interpreter. IO has always confused me – it's a monad, but the monad part is really about structuring the code. The strictness of the IO monad is really orthogonal to the use of the monad, and this was the first time that I've really put all of that together. There's some good discussion in this [9]blog post.

At the end of the tutorial there was a Haskell case study, where an energy company showed how they'd used Haskell to allow them to define various cost models which they could then simulate.

The final talk of the day was by [10]Don Syme. It started with a great discussion of what made a language functional – having functions as first class objects is not really enough these days. Don had a set of 11 (I think) items that made a language functional, and I hope the slides get published at some point as this list was very good. The rest of his talk was about the new F # 3 type provider mechanism, a compiler extension that can be used by the system to dynamically define new F # types, and which can be used to do the kinds of things that we used to do by

code generation. For example, taking an OData source and mapping it into the language. The day was brilliant, with a load of like minded people enthusing about functional languages. I hope that it is repeated next year!

1. <http://blog.software-acumen.com/>
  2. <http://www.fpday.net/fpday2011/index.php>
  3. <http://haskell.org/haskellwiki/Haskell>
  4. <http://clojure.org/>
  5. <http://msdn.microsoft.com/en-us/fsharp/>
  6. <http://research.microsoft.com/en-us/people/simonpj/>
  7. <http://www.erlang.org/>
  8. <http://www.well-typed.com/>
  9. <http://blog.ezyang.com/2011/05/unraveling-the-mystery-of-the-io-monad/>
  10. <http://research.microsoft.com/en-us/people/dsyme/>
- 

### 5.10.3 Catch it if you can (2011-10-20 08:56)

[1]How Debuggers Work: Algorithms, Data Structures and Architecture by Jonathan B. Rosenberg

I've recently become interested in using the debugging API of the CLR, so I thought this book might be a useful read. The first thing to note is that this book is fairly old. Published in 1996 it covers 32 bit Windows and Java as if they are new technologies, and hence misses the improvements to their debugging interfaces from the last ten years.

That said, the book is a great introduction to how debuggers do their stuff, and covers beautifully the hardware support that some processors offer to the debugger, and why you might not want to use this hardware support. The book is written from the point of view of someone who has written a debugger, and covers the difficulties that the author of a debugger will face when dealing with optimising compilers and chips that have optimisation features such as delay slots in their branches.

A large swathe of material is covered, including how the debugger can deal with multithreaded applications (which were the future when the book was written). I thought that the chapter on compiler optimisations and how the debugger can try to map optimised code back to the initial source was really good – in particular, a given source line can map to multiple disparate sections of the final machine code, and the debugger has to deal with this.

The only thing I didn't find useful in the book was the algorithms that the author gives all over the place. The text gives enough detail to put the algorithms together oneself, and I found that they just disturbed the flow.

For a good overview of the low level implementation of debuggers I recommend this book, which can be obtained fairly cheaply second hand.

1. [http://www.amazon.co.uk/How-Debuggers-Work-Algorithms-Architectures/dp/0471149667/ref=sr\\_1\\_1?ie=UTF8&qid=1318926556&sr=8-1](http://www.amazon.co.uk/How-Debuggers-Work-Algorithms-Architectures/dp/0471149667/ref=sr_1_1?ie=UTF8&qid=1318926556&sr=8-1)

---

#### 5.10.4 There's no debugger for this situation (2011-10-21 08:57)

[1] Debugging the Development Process by Steve Maguire

This is a great read about things that are wrong with typical development processes. The author fills the book with anecdotes from the time he spent at Microsoft, first working on various teams and then being used as a trouble-shooter to improve the performance of failing teams.

Lots of good, simple advice that makes a lot of sense.

1. [http://www.amazon.co.uk/Debugging-Development-Process-Steve-Maguire/dp/1556156502/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1318927285&sr=1-1](http://www.amazon.co.uk/Debugging-Development-Process-Steve-Maguire/dp/1556156502/ref=sr_1_1?s=books&ie=UTF8&qid=1318927285&sr=1-1)
- 

#### 5.10.5 Tell Roslyn I've been waiting for her (2011-10-22 20:35)

I just got my hands on the Roslyn preview, and having six hours to wait at Seattle airport, I thought I'd have a play with it. On first impressions it is really good. You should note that the implementation is currently quite incomplete with lots of syntax not supported, but it does give a really good feel for what will be available in the future.

Roslyn is Microsoft's effort to rewrite their C# compiler in C# - I believe the original compiler is written in C++. It's a shame they have left it this long before doing so, as rewriting it in C# would seem to give a number of benefits. First, it proves that C# is a sufficiently good implementation language to write such things. This is no bad thing and used to be the first thing that you did in a new language. Writing the compiler allowed you to write the first big application in the language, allowing you to check out the idioms and richness of the language. Most importantly though, it makes it possible to use the compiler as a service as the managed code can be easily imported into another managed process, and if you expose enough APIs the compiler can easily be used as a service.

Let's try some simple experiments with this new API. First we need a syntax tree for a small amount of C#. We can use the textual representation of some C# and get the Roslyn code to parse it into a syntax tree for us.

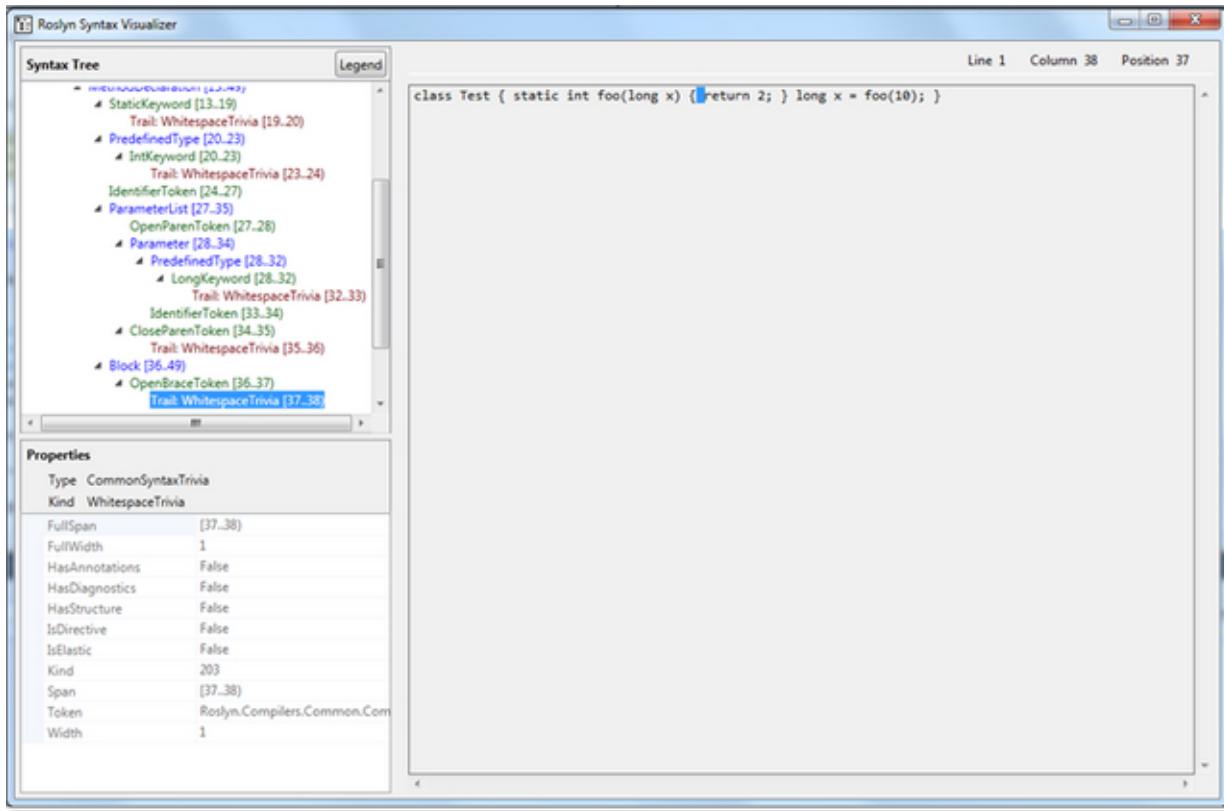
[1]

```
var fooDefn =
    SyntaxTree.ParseCompilationUnit("class Test { static int foo(long x) { return 2; } long x = foo(10); }");
```

If you compile up the debug visualizer that comes in the preview, and then install that in the Visualizers directory for the Visual Studio that you are running, you get a very nice representation of the parse tree. Roslyn preserves the whitespace by adding its details as fields in the parse

tree nodes. This allows your code to concentrate on the parse tree, while still allowing the original code to be regenerated. This is very useful for writing tooling of various kinds.

[2]



Roslyn is a lot more than a syntax analyser though. The focus here is on the semantics of the languages that it processes – both C # and VB.

In order to get semantic information, we need to package up the syntax tree together with the references that are needed to compile it. We're going to write the code out later, so we need to pass in some compilation options that say we're creating a standalone assembly.

[3]

```
var compilation = Compilation.Create("anyname", options: new CompilationOptions(assemblyKind: AssemblyKind.DynamicallyLinkedLibrary))
    .AddReferences(new AssemblyFileReference(typeof(object).Assembly.Location))
    .AddSyntaxTrees(fooDefn);
```

We access compilation information using the semantic model that is associated with the code.

[4]

```
var model = compilation.GetSemanticModel(fooDefn);
```

Before getting the model, you should check for errors in the code using the compilations diagnostic collection. I forgot to do this when experimenting and wondered why I kept getting error elements in the semantic model.

[5]

```
foreach (var diagnostic in compilation.GetDiagnostics())
{
    Console.WriteLine(diagnostic);
}
```

We can find the call to f. When this is compiled the compiler needs to insert some implicit conversions to convert the various int values into the longs that are needed. Using the managed C # compiler, we can get information to say that these implicit conversions are present.

We navigate the tree to find the call.

[6]

```
var classDecl = (ClassDeclarationSyntax) fooDefn.Root.ChildNodes().ElementAt(0);
var field = (FieldDeclarationSyntax) classDecl.Members.ElementAt(1);
var declaration = (VariableDeclaratorSyntax) field.Declaration.Variables.ElementAt(0);
var initializer = declaration.InitializerOpt;
var expr = initializer.Value;

var res = model.GetSemanticInfo((ExpressionSyntax)expr);
```

And then look at the semantic information associated with that part of the tree.

[7]

Name	Value
expr	InvocationExpressionSyntax InvocationExpression foo(10)
res	(Roslyn.Compilers.CSharp.SemanticInfo) None Count = 0 null
CandidateReason	"NamedType System.Int64"
CandidateSymbols	[ImplicitNumeric]
ConstantValue	false
ConvertedType	Count = 0
ImplicitConversion	"Method System.Int32 Test.foo(System.Int64 x)"
IsCompileTimeConstant	"NamedType System.Int32"
MethodGroup	
Symbol	
Type	
Static members	
Non-Public members	

Notice how the implicit conversion is recorded in the Implicit conversion field of the information object. Note too, the Type and ConvertedType fields which record the pre- and post- conversion types.

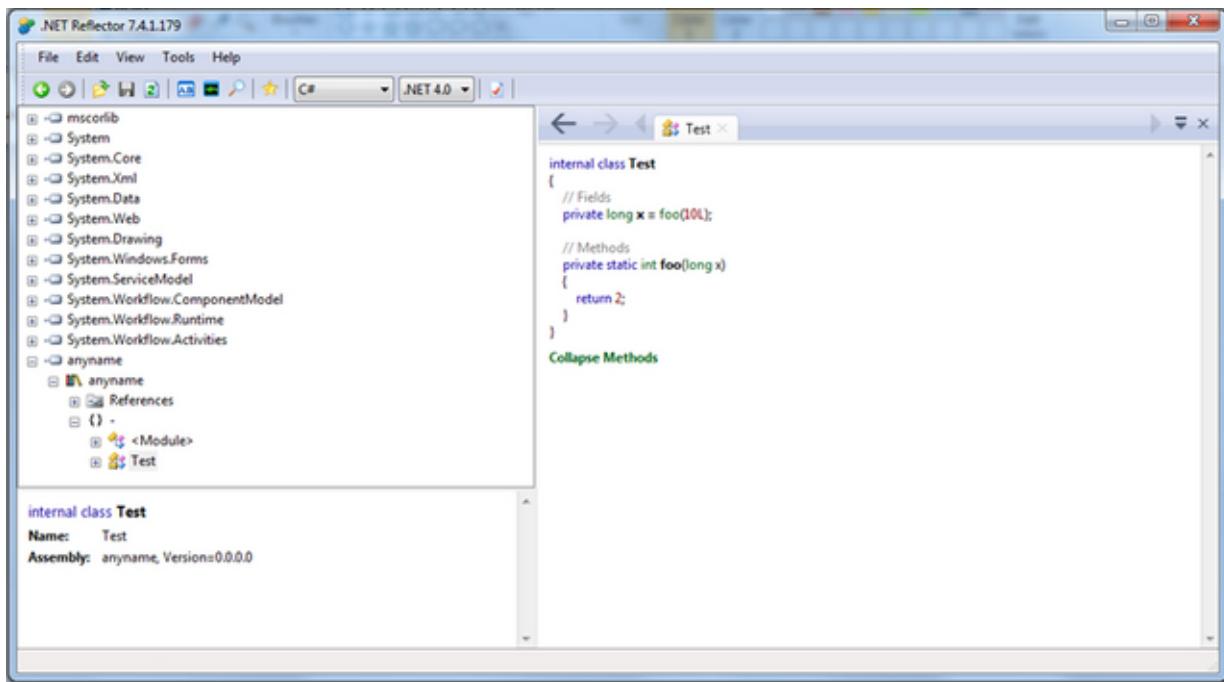
We have a full compiler here though, so we can actually get it to write out the compiled code as an assembly.

[8]

```
using (var targetStream = File.Open("c:/users/clive.tong/desktop/foo.dll", FileMode.Create))
{
    compilation.Emit(targetStream);
}
```

Using [9]Reflector, we can see the decompilation of the written module.

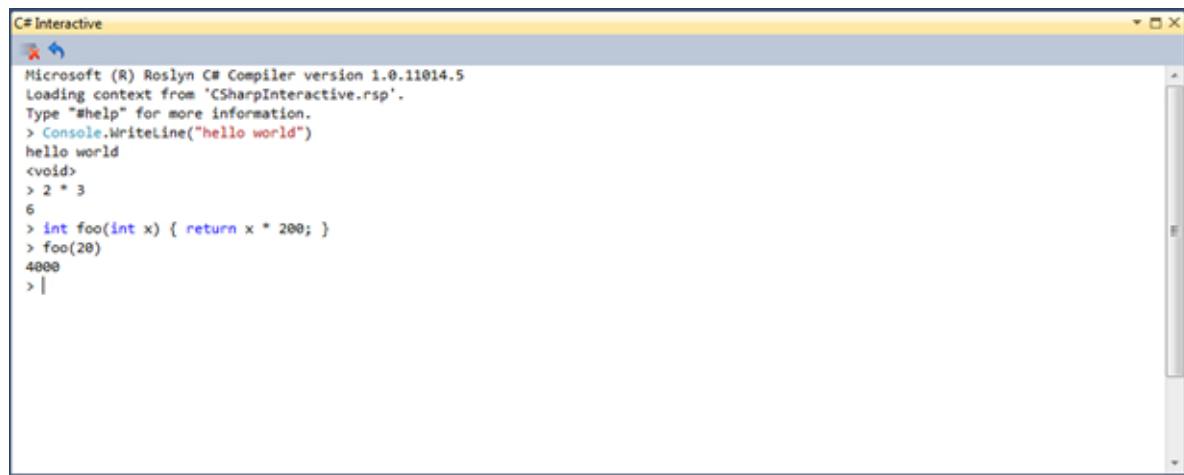
[10]



There are loads of samples in the preview that demonstrate the kinds of things you can do with this power. Code analysis, refactoring plus all kinds of meta-programming are going to be possible at the C# level. Meta-programming is usually done at the IL level these days using tools like Cecil. Working at the language level is going to make such transforms much easier to debug, and using the same compiler as Visual Studio itself is going to mean that your analysis tool and code generation tool agree on the syntax and semantics.

One other rather cool use of the managed C# compiler is to produce a REPL, including intellicense.

[11]



A lot of day to day programming is exploratory in nature and a REPL makes it so much easier for the user to try out ideas with instant feedback without having to go through a time consuming compile cycle (of the whole program). Understanding an API to some library class often requires some experimentation. Languages like Common Lisp have had this ability forever, and it will be interesting to see if the class-based syntax of C# keeps this a useful feature.

Unfortunately it won't be available in the next release of Visual Studio, dev 11, which will probably be released sometime in 2012, but at least we have a flavour of where C# is going

in the future.

1. <http://clivetong.files.wordpress.com/2011/10/pic1.png>
2. <http://clivetong.files.wordpress.com/2011/10/visualizer.png>
3. <http://clivetong.files.wordpress.com/2011/10/pic2.png>
4. <http://clivetong.files.wordpress.com/2011/10/pic3.png>
5. <http://clivetong.files.wordpress.com/2011/10/pic4.png>
6. <http://clivetong.files.wordpress.com/2011/10/pic5.png>
7. <http://clivetong.files.wordpress.com/2011/10/pic8.png>
8. <http://clivetong.files.wordpress.com/2011/10/pic6.png>
9. <http://www.reflector.net/>
10. <http://clivetong.files.wordpress.com/2011/10/pic7.png>
11. <http://clivetong.files.wordpress.com/2011/10/repl.png>

---

joakim (2011-12-12 12:41:29)

Cool and interesting!

### 5.10.6 An Englishman's home is his castle (2011-10-23 23:16)

[1]Channel 9 have recently published an [2]interesting video interview with some of the creators of Drawbridge. Various researchers at Microsoft Research have been doing work revolved around software isolated processes. This time they have a scheme which allows software isolation, with its associated security benefits, together with a means to allow an application to start executing on one OS and then have it migrated across to another. The scheme they use allows cheap virtualization and persistence of a running application, and at some point in the interview they mention how one could consider starting an application on a desktop machine, transferring it on to a mobile device for a train journey and migrating onto the cloud when the application needs to do a lot of processing.

The associated paper, found [3]here gives details of the work that they have done, and while it is a research paper, and hence not directly useful, the virtualisation exercise was carried out on a modern operating system (Windows 7) and the test applications that they have run include applications such as Excel and Notepad.

They reduce the traditional kernel down to a very small number of API calls, wrapping these calls in user mode code to simulate the traditional windows operating systems API. This kernel can be controlled using a security manager, and the smaller surface area makes it a lot easier to reason about the security of the application. In addition, the kernel API is stateless, which is what makes it possible to move the application from host to host. The isolated processes communicate using sockets, and they get rid of device drivers to the mouse and windowing system by using the [4]RDP protocol for the communication. To get this process to work they have to use a per-process simulation of the registry and the COM running object table, so the scheme will not work for all applications without more work. However, it is interesting in the video to see a process running Excel being persisted and then restarted in a new session.

Microsoft are in a transition phase at the moment, with some of Win32 being migrated into the new WinRT layer, a layer which can run be implemented on other platforms than the traditional

Wintel, so the work is timely.

For a general introduction to virtualization, [5]this article is a good read.

1. <http://channel9.msdn.com/>
  2. <http://channel9.msdn.com/Shows/Going+Deep/Drawbridge-An-Experimental-Library-Operating-System>
  3. <http://research.microsoft.com/apps/pubs/default.aspx?id=141071>
  4. [http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/\[MS-RDPBCGR\].pdf](http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-RDPBCGR].pdf)
  5. <http://www.anandtech.com/show/2480>
- 

### 5.10.7 Something for the weekend? (2011-10-30 22:17)

I haven't had time lately to read any computer books, mainly because I've been trying to finish a [1]book on basic Economics that I bought at the airport last week. This weekend though, I did get to read a couple of papers that I've had for a while.

First a blog post by Joe Duffy on the cost of [2]Generics in C #, mainly from the point of view of being able to ahead of time compile classes using Ngen and when a use of generics is likely to cause the jit to execute at run time. As usual, an interesting exposition on the issues, and the post contains a number of links to other interesting posts on the implementation of generics in the CLR. In the near future, I'm going to get out a debugger and have a close look at the code that is generated for some simple examples of generics.

I also took some time to read the [3]Roslyn Project Overview whitepaper which gives a good overview of what the Roslyn project is going to offer. The paper goes into some detail about the API that Roslyn offers, covering some points that I haven't seen before like the use missing tokens in the syntax tree to represent errors. The whitepaper talks about tree rewriting of immutable trees and I was surprised to see no mention of [4]Zippers, the usual datatype for such changes.

After these distractions, it is back to looking at Windows 8 and .NET 4.5 in the spare time that I have. In particular I want to find out a lot more about DirectX and its use from managed code.

1. [http://www.amazon.co.uk/What-Need-Know-About-Economics/dp/0857081144/ref=sr\\_1\\_1?ie=UTF8&qid=1320012321&sr=8-1](http://www.amazon.co.uk/What-Need-Know-About-Economics/dp/0857081144/ref=sr_1_1?ie=UTF8&qid=1320012321&sr=8-1)
  2. <http://www.bluebytesoftware.com/blog/2011/10/23/OnGenericsAndSomeOfTheAssociatedOverheads.aspx>
  3. <http://www.microsoft.com/download/en/details.aspx?id=27744>
  4. [http://en.wikipedia.org/wiki/Zipper\\_\(data\\_structure\)](http://en.wikipedia.org/wiki/Zipper_(data_structure))
-

## 5.11 November

### 5.11.1 Small is beautiful...and fast... and obfuscated... and (2011-11-23 21:00)

[1]Closure: The definitive guide by Michael Brolin

This is a really interesting book that offers lots of tips on Javascript as well as providing an in depth guide to the Closure Tools, both in terms of how to use them and, more importantly, the reasons behind the design. I got the book from the library because I'd heard about Closure from its use in the recently released Clojure to Javascript compiler. In this project the Closure compiler is used to minimise the Javascript, optimising it in the process. This allows the code generator of the compiler to generate relatively inefficient Javascript in the knowledge that it will be optimised by the Closure compiler. This simplifies the compiler greatly.

First, what is Closure? It is a package of five tools- a Javascript library, an HTML templating mechanism, a compiler (from JavaScript to Javascript), a testing framework and an inspector. It was produced by Google to help them maintain and build rich web applications like Gmail. Such applications need to be portable across a large range of browsers and also need to be written in maintainable code which can be loaded quickly and efficiently by the relevant browsers. Closure contains a portable library that isolates the code base from the tweaks that are often required to get the user interface to look good (or even work) on all of the various browsers.

First the compiler, the thing that drew me to this book. This takes Javascript code, parses it to an abstract syntax tree which can be transformed in a number of ways before it is output again. This transformation is extensible by the user, who can add their own rules and transforms. To get the most from the compiler, the user is expected to annotate their Javascript functions with JSDoc annotations giving the types of the various entities. The compiler can then type check the Javascript for you, getting rid of many potential errors before runtime. The compiler works best if your code uses the Clojure Library and its idioms for modularisation and class definition. The code can be split into separate groups of files corresponding to functionality, and by using provide and require the compiler can determine the dependencies between the groups. This information can be used to compress multiple Javascript files into a multiple result files, with these files set up so that code is demand loaded when a piece of functionality is used – think Gmail and the fact that you don't need to load the calendar code until the user goes to the calendar.

With relevant type information, which includes a notion of subtyping, the compiler can report type errors, and even a full program analysis to determine dead code which it can throw away. Most importantly, the compiler freely renames variables and folds constants to reduce the size of the code considerably, drastically improving the start up time for an application.

The Closure Library offers a standard platform to program against. It implements standard Javascript functions that aren't implemented natively in some browsers, and also provides a whole lot of extra utility functionality. For example, the object goog.array implements a whole lot of utility functions on arrays such as forEach and goog.object offers functions such as get and setIfUndefined. There are also functions for generating and parsing JSON, and a whole load of DOM manipulation functions such as goog.dom.getElementsByTagNameAndClass and goog.dom.getAncestorByTagNameAndClass. As mentioned above it implements a modularisation mechanism via goog.provide and goog.require which is built upon an accessible lower level dependency mechanism. There's also a whole mass of code for getting details about the current Javascript host (browser). A standard user extensible event abstraction is also provided for handling both bubbling and tunneling events which are abstracted across the many browser

implementations. Typically IE has different behaviour from the rest of the crowd, so the library needs to work around these differences.

Closure Library also offers a pattern for implementing standard class based object orientation in Javascript. There are standard patterns of class, inheritance, interfaces, override and calling base methods. If you follow the pattern, then the Closure Compiler can detect errors in the code much more easily.

As you'd expect, given the types of applications that Google produces, there's a lot of functionality concerned with implementing Ajax client-server applications. For example, `goog.net.XhrIo` is a wrapper over the basic `XMLHttpRequest` which offers events for completion and errors. `goog.net.XhrManager` allows the application to configure the timeouts and retries, and there's a bulk loader for firing off downloads and getting notified on completion. There's support for communication between independent iframes and support for COMET, the standard long running request trick which allows clients to avoid polling while still getting timely notifications from a server.

The Closure Library offers a standard set of widgets and a component model for writing more. This component model offers a lifecycle, with events being used to communicate transitions between the states. The model offers two ways for the component to affect the DOM – either by generating a tree of elements which can be patched into the DOM (rendering) or by attaching to existing DOM elements (decorating). This latter technique allows a web page to be loaded quickly, with the behaviour part only getting added later when the Javascript finishes loading. This technique makes it appear to the user that the page has loaded and is ready, using the delay before they start interacting with it for loading the code that handles the behaviour. The book goes into lots of details and implements a component to make everything very clear. The Library also implements Rich Text Editor, and there is a whole chapter in the book on how to configure and use it.

Closure Templates allow you to write a template for a particular pattern of HTML, avoiding the need to inline a lot of construction code which can be really hard to edit. The book covers this area really well.

This is followed by several chapters on implementing your own compiler extensions, followed by coverage of the Testing Framework. This framework allows you to write tests in Javascript and run them and display the results in a browser. There is good support for mocking and testing asynchronous methods. There's a chapter after this on the Closure Inspector, a Firefox add-in for debugging code that has been through the Closure Compiler. The compiler can generate information relating the output Javascript to the original source code, and the add-in allows you to seamlessly jump from the compressed code that you are debugging back to the original code.

This book covers everything really well, making it clear what the various features are designed to help with, and explaining why they were designed in that way. Moreover, the appendixes of the book, one on implementing inheritance in Javascript and the other on common Javascript pitfalls are a fantastic read. The latter makes some really good points on problems you can have with the difference between the `prototype` property and the object's `prototype` (which is accessible via the `_proto_` property).

The book is 500 pages long which requires some effort to read, but I learned a lot about the Closure Tools and during the course of the book a fair amount of material about writing rich internet applications too.

1. [http://www.amazon.co.uk/Closure-Definitive-Guide-Michael-Bolin/dp/1449381871/ref=sr\\_1\\_1?ie=UTF8&qid=1322037909&sr=8-1](http://www.amazon.co.uk/Closure-Definitive-Guide-Michael-Bolin/dp/1449381871/ref=sr_1_1?ie=UTF8&qid=1322037909&sr=8-1)

---

### 5.11.2 That's all I got from my inheritance? (2011-11-24 07:00)

WinRT has an object model that owes a great debt to COM, which has been part of the Windows operating system since the last century. Like COM servers of old, WinRT communicates with the application via an object that they share.

Taking a standard Silverlight application, the start up class is named App by the code that generates the Silverlight project. An instance of this class will be made and the Run method will be called on it to jump back into WinRT (see the main method in the second picture). WinRT can then finish the startup and call back into the application via its OnLaunched handler (or potentially via one of the other contracts for something like Search).

[1]



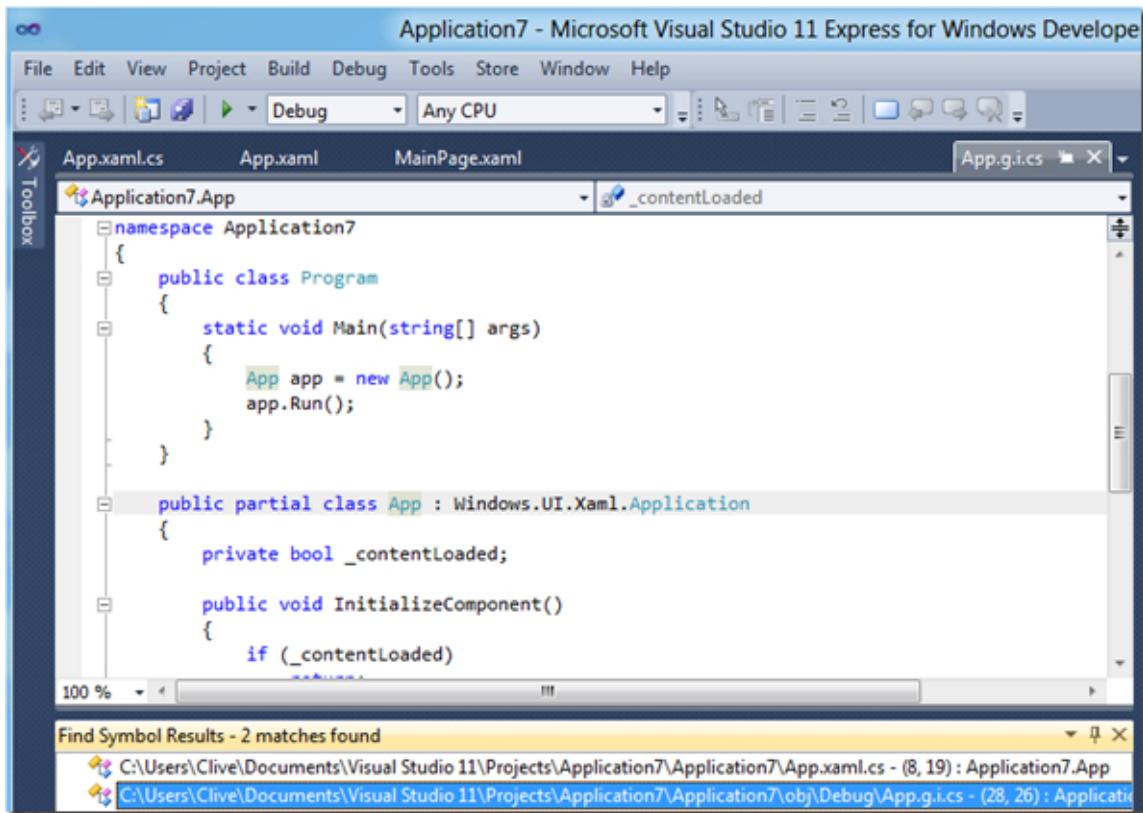
```
namespace Application7
{
    partial class App
    {
        public App()
        {
            InitializeComponent();
        }

        protected override void OnLaunched(LaunchActivatedEventArgs args)
        {
            Window.Current.Content = new MainPage();
            Window.Current.Activate();
        }
    }
}
```

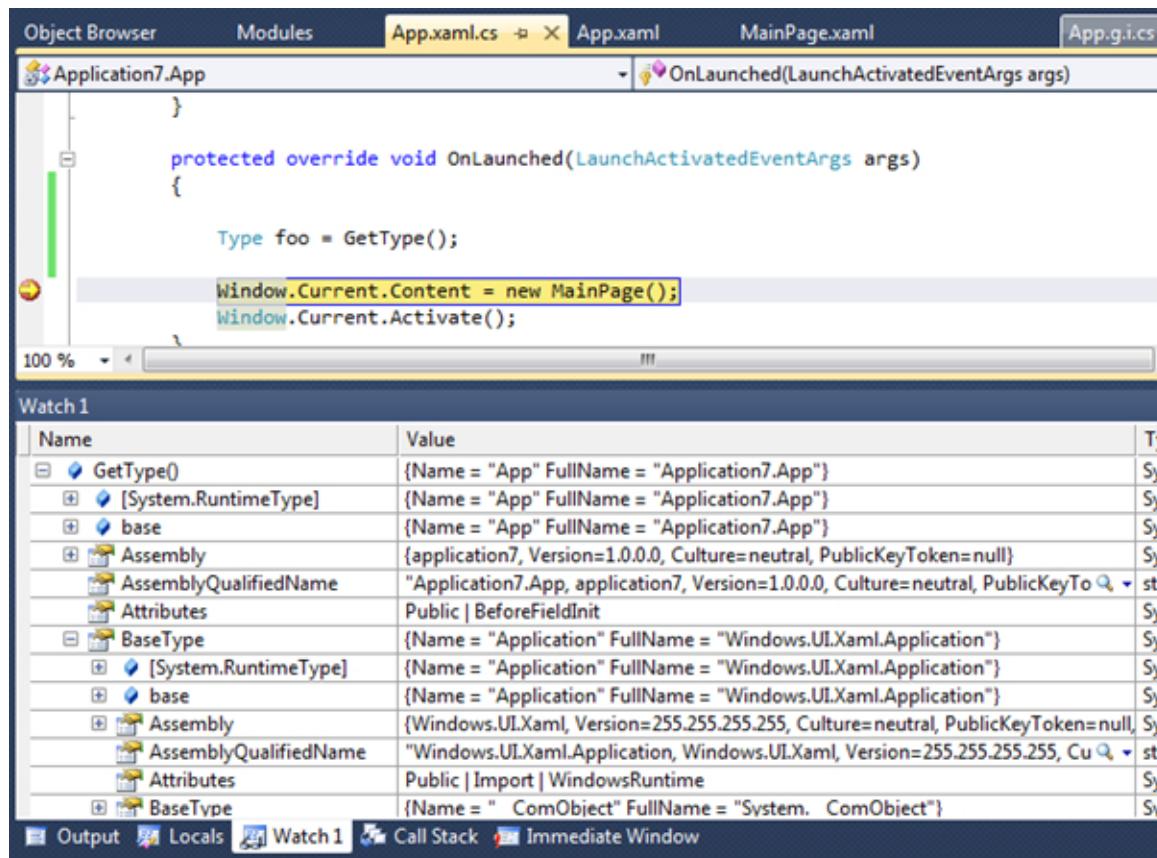
App is a partial class, and the user definitions are mixed together with a generated class in the App.g.i.cs file. It is this class which is marked as inheriting from Windows.UI.Xaml.Application. The latter class is something we can find in WinMD files, the metadata files that describe the services offered by the WinRT runtime. Moreover, the classes from the WinMD files are “projected” into the various languages, making them appear much like normal classes in those languages.

[2]

592



If we break inside the Initialize, we can check the inheritance of the App instance. It inherits from Windows.UI.Xaml.Application which inherits from \_\_ComObject.



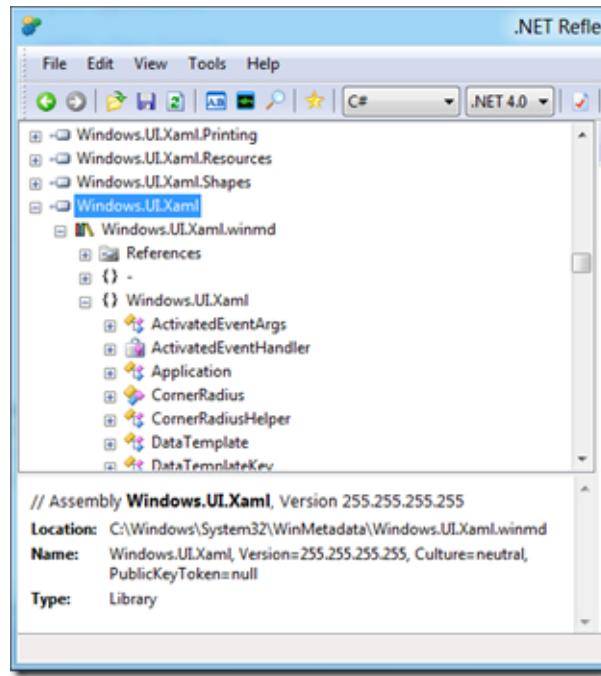
Using the “Make Object ID” functionality in the debugger, we can get hold of the BaseType and check the interfaces that it supports.

[4]

1#	{Name = "Application" FullName = "Windows.UI.Xaml.Application"} {1#}	object
((System.Type) 1#).GetInterfaces()	(System.Type[2])	System
[0]	{Name = "IApplication" FullName = "Windows.UI.Xaml.IApplication"}	System
[1]	{Name = "IApplicationOverrides" FullName = "Windows.UI.Xaml.IApplicationOverrides"}	System

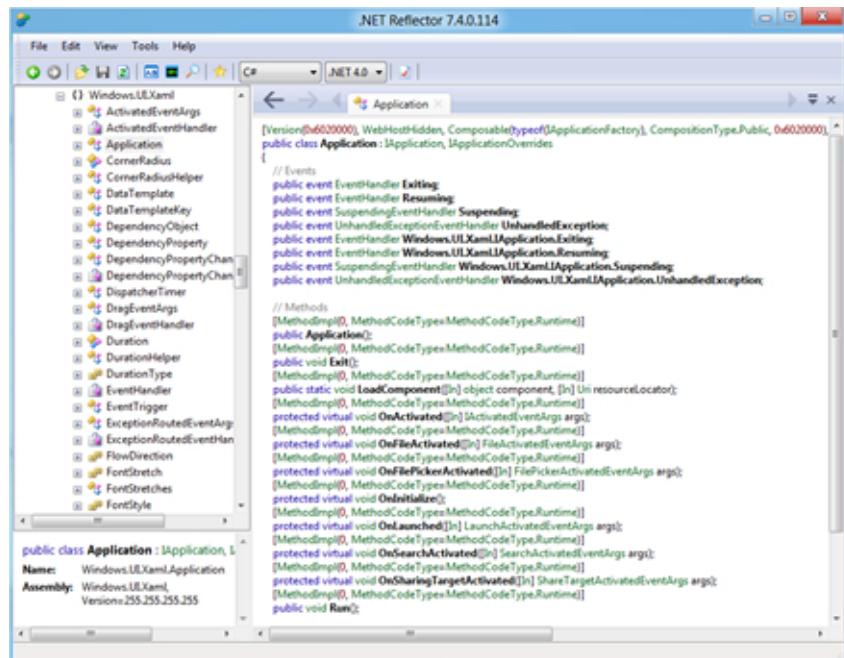
These interfaces are all described in metadata which you can find on the system. Moreover, the format of the metadata is derived from that used by the CLR, so you can inspect it (and search and analyse it) using [5]Reflector.

[6]



Application looks like this, and the following pictures show the information for the interfaces that it supports. The items supported by Application are split across these two interfaces.

[7]



[8]

```
[WebHostHidden, Version(0x6020000), Guid(0x74b861a1, 0x7487, 0x46a9, 0x9a, 110, 0xc7, 0x8b, 0x51, 0x27, 0x26, 0xc5), Ex]
internal interface IApplication
{
    // Events
    event EventHandler Exiting;
    event EventHandler Resuming;
    event SuspendingEventHandler Suspending;
    event UnhandledExceptionHandler UnhandledException;

    // Methods
    void Exit();
    void Run();

    // Properties
    ResourceDictionary Resources { get; [param: In] set; }
}
```

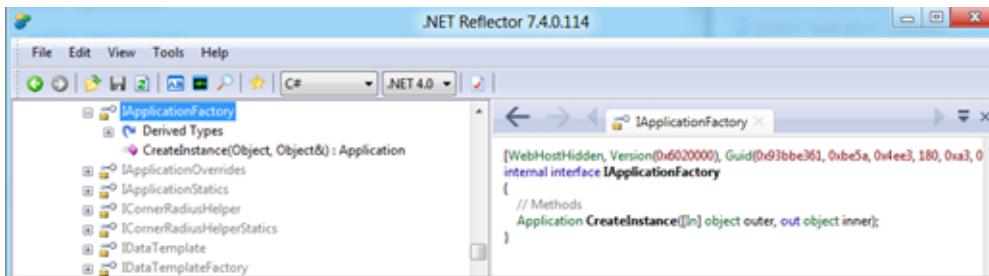
[9]

```
[ExclusiveTo(typeof(Application)), Guid(0x25f99ff7, 0x9347, 0x459a, 0x9f, 0xac, 0xb2, 0xd0, 0xe1, 0x1c, 0x1a, 15), Versid]
internal interface IApplicationOverrides
{
    // Methods
    void OnActivated([In] IActivatedEventArgs args);
    void OnFileActivated([In] FileActivatedEventArgs args);
    void OnFilePickerActivated([In] FilePickerActivatedEventArgs args);
    void OnInitialize();
    void OnLaunched([In] LaunchActivatedEventArgs args);
    void OnSearchActivated([In] SearchActivatedEventArgs args);
    void OnSharingTargetActivated([In] ShareTargetActivatedEventArgs args);
}
```

There's one other piece in the jigsaw, the IApplicationFactory interface.

Application supports two interfaces, one containing the application provided events and methods which can't be overridden (IApplication) and one that is designed to be overridden (IApplicationOverrides). The idea is that behind the scenes, the overrides are constructed into an object that implements the IApplicationOverrides interface. This is then passed to the factory, which combines the overrides with the system provided Application object, returning a new object which will call the original methods on the non-overrides, and which will call the user supplied overrides when a call is made on one of the methods from that interface. The factory also provides the original implementation of those methods – this returned implementation can be stitched into the overrides to make calls to the base functionality (and is the inner parameter in the method signature).

[10]



Since this is hidden behind the work that the projection mechanism does when it imports the interfaces into the managed world, you can really only explore what is really happening if you use a language where the project can be turned off. There's a really good post from Ian Griffiths [11] here which implements the top level application object in C++ without using any of projections which covers the items above in really good detail. Go and read it now!

1. <http://clivetong.files.wordpress.com/2011/11/appclass.png>
2. <http://clivetong.files.wordpress.com/2011/11/appclass2.png>

3. <http://clivetong.files.wordpress.com/2011/11/breakininitialize.png>
  4. <http://clivetong.files.wordpress.com/2011/11/breakininitialize2.png>
  5. <http://www.reflector.net/>
  6. <http://clivetong.files.wordpress.com/2011/11/winmd.png>
  7. <http://clivetong.files.wordpress.com/2011/11/application1.png>
  8. <http://clivetong.files.wordpress.com/2011/11/iapplication.png>
  9. <http://clivetong.files.wordpress.com/2011/11/iapplicationoverrides.png>
  10. <http://clivetong.files.wordpress.com/2011/11/applicationfactory.png>
  11. <http://www.interact-sw.co.uk/iangblog/2011/09/25/native-winrt-inheritance>
- 

## 5.12 December

### 5.12.1 I'll take a bite out of the existing code (2011-12-04 08:41)

At [1]Red Gate, we occasionally have a Down Tools Week – a week in which you can work on any code that you want. In the recent such week, I spent my time working on the code in [2]Reflector for writing out a modified object model of an assembly. I wrote up some of what I did [3]here.

1. <http://www.red-gate.com/>
  2. <http://www.reflector.net/>
  3. <http://www.reflector.net/2011/12/morrissey-byte-code-manipulation-using-reflector/>
- 

### 5.12.2 That was a lightening quick introduction! (2011-12-11 22:03)

I did a lightening talk at work on COM and how it is used in WinRT – the slides are available [1]here. I've also been poking around under the covers of Windows 8 using cdb and will blog about this soon. In the meantime, there's a good podcast with Ted Neward on the subject of Windows 8 [2]here.

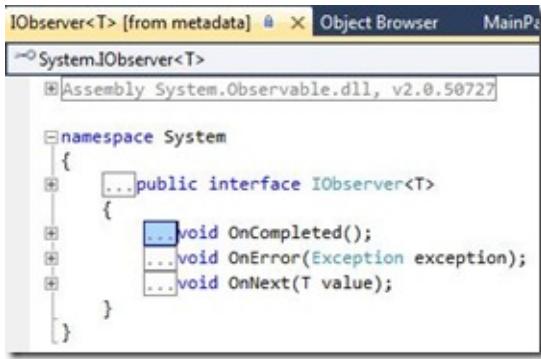
1. <https://skydrive.live.com/#!/view.aspx?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%21475>
  2. <http://thesoundof.net/Home/Play/1746>
-

### 5.12.3 Please don't call, I'm not ready (2011-12-12 09:11)

[1]Programming Reactive Extensions and LINQ by Jesse Liberty and Paul Betts

I must confess to being really disappointed with this book. A book with the subtext, "dive deep into the next important .NET technology" should surely contain more than 150 pages and shouldn't contain a chapter on the unrelated LINQ to SQL as one of the 9 chapters. The book could use up lots of the space that it devotes to listings of code to more of the traditional material on Rx - [2]marble diagrams for example which do a lot to explain what is really happening. Having said that, the book does contain some interesting material, in particular a chapter on the Reactive UI and testing reactive programs using virtual schedulers.

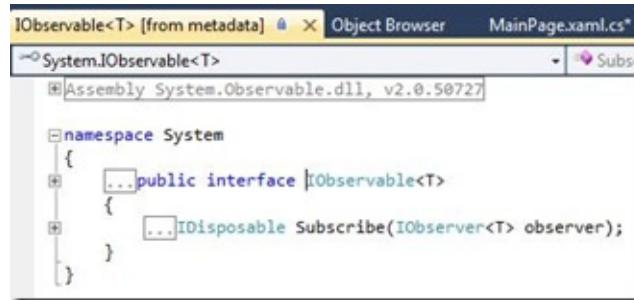
LINQ has been around for a while now, so we've all got used to the idea of sequences of values which we can process in interesting ways; filtering, projecting and merging to express computations in a very declarative fashion. `IEnumerable` is supported very nicely by C # 3 and its various language extensions. Rx turns this on its head by being based on the dual of the `IEnumerable` interface, the `IObservable`. Rather than the consumer pulling values and blocking if they are not available, in the Rx world, the consumer instead has values pushed at them when they become available via the `OnNext` method. If the source has finished then the `OnCompleted` warns the observer that the supply has terminated and they should unsubscribe. If an error happens, then `OnError` warns the consumer that there has been a problem.



[3]

In the same way that there are `IEnumerator` and `IEnumerable` interfaces, there is an `IObservable<.>` interface which allows the observer to subscribe to the data source.

[4]



This returns an `IDisposable`. The contract of the interface specifies that once the observer has called the `Dispose` on the returned interface, it will receive no more values. We could therefore write an example like the following; note that here the values are pushed when the `Run` method is called, but in most real cases the `OnNext` would be called from some other thread probably after some external stimulus.

[5]

```

class Generator : IObservable<int>
{
    private bool m_Finished = false;
    private IObserver<int> m_Observer;

    public IDisposable Subscribe(IObserver<int> observer)
    {
        m_Observer = observer;
        return new Subscription { Generator = this };
    }

    public void Run()
    {
        m_Observer.OnNext(1);
        if (m_Finished) return;
        m_Observer.OnNext(2);
        if (m_Finished) return;
        m_Observer.OnNext(3);
        if (m_Finished) return;
        m_Observer.OnCompleted();
    }

    class Subscription : IDisposable
    {
        public Generator Generator { get; internal set; }
        public void Dispose()
        {
            Generator.m_Finished = true;
        }
    }
}

```

A typical use by a consumer might be.

```

var source = new Generator();
using (source.Subscribe(new Handler()))
{
    source.Run();
}

```

[6]

This all sounds very simple, but of course there are issues involving threading models if we want to ensure that arriving values are processed in the correct order. It can be quite subtle ensuring that no more values are delivered once the subscription has been disposed. There are also complications in that observables can be “hot” or “cold”. Some observables only return values if you are subscribed at the point that they wish to push the value out to subscriptions, whereas others will push out their full sequence no matter when you subscribe.

Rx contains a rich set of operators to convert between the various types of observable as well as controlling the thread on which things happen.

These Rx sequences map well to the world of asynchronous function calls. The stream of values can turn out to have no elements, and waiting consumers can be notified of completion and errors in the generation, or indeed can be infinite in the sense that the stream is never terminated. Various Rx operators can convert the different types of .NET event patterns into observable streams of values. The book contains many examples of streams of mouse events or web events which follow the Async pattern. The book has good examples of processing values fetched from the web, and indeed includes the canonical “Google suggest” implementation and an implementation of drag and drop which are the common examples of Rx.

The book covers a Reactive UI library which looks very interesting, but is given very little coverage. There is also a great chapter on using virtual time to make it easy to write unit tests of asynchronous functionality, but again there is so much more that the book could have covered.

I suspect that most of the material in the book can be found fairly easily on the web, and given that the book has a lot of space taken up by source code listings, the actual value add by the authors feels very low. This is a shame as Rx is a really interesting area and deserves to be made more available to the public.

1. [http://www.amazon.co.uk/Programming-Reactive-Extensions-Jesse-Liberty/dp/1430237473/ref=sr\\_1\\_1?ie=UTF8&qid=1323555244&sr=8-1](http://www.amazon.co.uk/Programming-Reactive-Extensions-Jesse-Liberty/dp/1430237473/ref=sr_1_1?ie=UTF8&qid=1323555244&sr=8-1)
  2. <http://blogs.microsoft.co.il/bnaya/archive/2010/02/28/rx-for-beginners-part-5-marble-diagrams-select-and-where.aspx>
  3. <http://clivetong.files.wordpress.com/2011/12/iobserver.jpg>
  4. <http://clivetong.files.wordpress.com/2011/12/iobservable.jpg>
  5. <http://clivetong.files.wordpress.com/2011/12/example1.jpg>
  6. <http://clivetong.files.wordpress.com/2011/12/use.jpg>
-

# 2012

## 6.1 January

### 6.1.1 Someone needs to check the plumbing (for leaks) (2012-01-01 16:22)

[1]Monads are one of the most fascinating items that come up during all introductory articles on [2]Haskell, usually when the article turns to input/output and how you can actually get your Haskell program to interact with the outside world. The IO monad in Haskell is used to wrap all input and output operations and all too often it appears that it's the fact that IO is a monad which makes it suitable for IO – that's far from being the case and is something that I found really confusing when I first learned Haskell.

A Monad has a very useful property when we want to keep most of our program referentially transparent – it's an ADT that wraps a value, allowing access to it only via a function which returns a value that is also wrapped by an instance of the same Monad. So the bind function ( $>>=$ ) has the type

[3]

```
infixl 1  >>, >>=
class Monad m  where
    (>>=)           :: m a -> (a -> m b) -> m b
    (>>)            :: m a -> m b -> m b
    return          :: a -> m a
    fail             :: String -> m a

    m >> k           =  m >>= \_ -> k
```

This means that the monad is contagious, so that once your types start using anything from the IO system, the IO Monad is always going to appear in the types of those functions, and therefore in the types of the functions that call or use those functions too. Hence anything that depends on the outside world, the entity that the IO Monad wraps, is going to have the IO in its type signature, and hence you're going to be able to easily identify it.

What I found really confusing at first was how the following code works:

```
[4] testIt () = do
    putStrLn "Part 1"
    putStrLn "Part 2"
    x <- getLine
    putStrLn ("Hello " ++ x)
```

The “do” is syntactic sugar, which simply expands into use of the relevant Monad (in this case IO). The bind operation ( $>>=$ ) is going to be used in the expansion, to thread the state of the world that results from the use of the putStrLn into the next call to putStrLn and then into the argument to getLine. When we run this, we see the output that you might expect.

[5]

```
*Main> :load c:/users/clive/desktop/haskell/foo.hs
[1 of 1] Compiling Main           < C:\users\clive\desktop\haskell\foo.hs, int
erpreted
Ok, modules loaded: Main.
*Main> testIt<>
Part 1
Part 2
clive
Hello clive
```

To me the interesting question is this: in Haskell, the evaluation is lazy – the final output line only depends on the  $x$  that we read from the keyboard, so why are the two previous lines output before the keyboard is read. In other words, from the signature it looks as if the Monad is simply a plumbing construct used to feed the results of one evaluation into another evaluation, so where does the forcing needed to cause the output to the console in a timely fashion come from.

It took me a while to think through how Haskell goes about evaluating expressions, and what actually causes the evaluation to happen. We tend to define functions in Haskell using pattern matching, which actually is translated into a case expression. Most of the evaluation that happens is driven by the needs of this case expression in finding out which of the cases of a pattern is valid.

[6]GHC has a useful way to see what is being evaluated. We can use the trace function, which outputs a string (the first argument) when the expression is evaluated, resulting in the evaluation of the second argument. Note that evaluation here means evaluating the expression to WHNF (weak head normal form), a form in which the outermost item is fully evaluated but where items nested further in the expression are potentially still in the form of thunks, unevaluated expressions which hang around until their value is needed.

[7]

```
import Debug.Trace

foo 0 x = 0
foo 1 2 = 20

test () = foo (trace "first" $ 0) (trace "second" $ 2)
test2 () = foo (trace "first" $ 1) (trace "second" $ 2)
```

Evaluating test is going to require the system to evaluate the first argument to see if it is zero. As it is zero, then second argument can remain unevaluated. However, evaluating test2 results in the second argument also being evaluated in order to see if the second case expression matches.

```
*Main> test<>
first
0
*Main> test2<
first
second
20
```

[8]

To see that monads can come in “strict” and “non-strict” flavours, have a look at the definitions below.

[9]

```
data Foo x = F x
instance (Show a) => Show (Foo a) where
  show (F x) = "(" ++ show x ++ ")"
instance Monad Foo where
  F a >>= y = y a
  return x = F x

result =
  do x <- trace "item0" $ F (trace "item1" $ 1)
     y <- F (trace "item2" $ 2)
  return ()

data Foo2 x = G x
instance (Show a) => Show (Foo2 a) where
  show (G x) = "(" ++ show x ++ ")"
instance Monad Foo2 where
  G a >>= y = a `seq` y a
  return x = G x

result2 =
  do x <- trace "item0" $ G (trace "item1" $ 1)
     y <- G (trace "item2" $ 2)
  return ()
```

When we evaluate result, we need only the value bound to x to be in WHNF, so the Monad is lazy. In the case of result2, we use 'seq' in the bind function to evaluate more of the argument, leaving us with a strict monad.

```
*Main> result
item0
(<())
*Main> result2
item0
item1
item2
(<())
[10]
```

In summary, it's not really just the fact that we use a Monad for input and output in Haskell that makes this all work. The IO Monad makes it easy to identify the part of the program that is affected by the outside world, and the Monad controls access to the value it carries, but the ordering of input and output operations really depends on the IO Monad being strict, ensuring that evaluation happens appropriately during the bind operation to get the effects on the outside world to happen in the correct order.

1. [http://en.wikipedia.org/wiki/Monad\\_\(functional\\_programming\)](http://en.wikipedia.org/wiki/Monad_(functional_programming))
2. [http://en.wikipedia.org/wiki/Haskell\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Haskell_(programming_language))
3. <http://clivetong.files.wordpress.com/2011/12/monad.png>
4. <http://clivetong.files.wordpress.com/2011/12/code1.jpg>
5. <http://clivetong.files.wordpress.com/2011/12/code2.jpg>
6. <http://www.haskell.org/ghc/>
7. <http://clivetong.files.wordpress.com/2011/12/code3.jpg>
8. <http://clivetong.files.wordpress.com/2011/12/code4.jpg>
9. <http://clivetong.files.wordpress.com/2011/12/code5.jpg>
10. <http://clivetong.files.wordpress.com/2011/12/code6.jpg>

### **6.1.2 Bridge the gap! (2012-01-04 16:44)**

I'm just starting to get deeper into what Microsoft did for the Windows 8 preview release at //BUILD. In particular, their work to move the XAML processing into the runtime itself, implementing it as COM based C++ instead of managed .NET code, which has lead to a number of interesting new areas to explore. Something that has come to my attention several times in blogs, has been the automatic generation of efficient interop wrappers for working between .NET code and the unmanaged world.

The Mono guys have just released [1]CXXI, a tool for generating interop classes for C++, and when I was looking at earlier work to make DirectX available from C #, the [2]SharpDx implementor has a number of really interesting [3]blog posts on implementing an automatic converter which he uses to automatically generate this mapping. This includes some great work in order to get good performance across to bridge between the managed and unmanaged worlds.

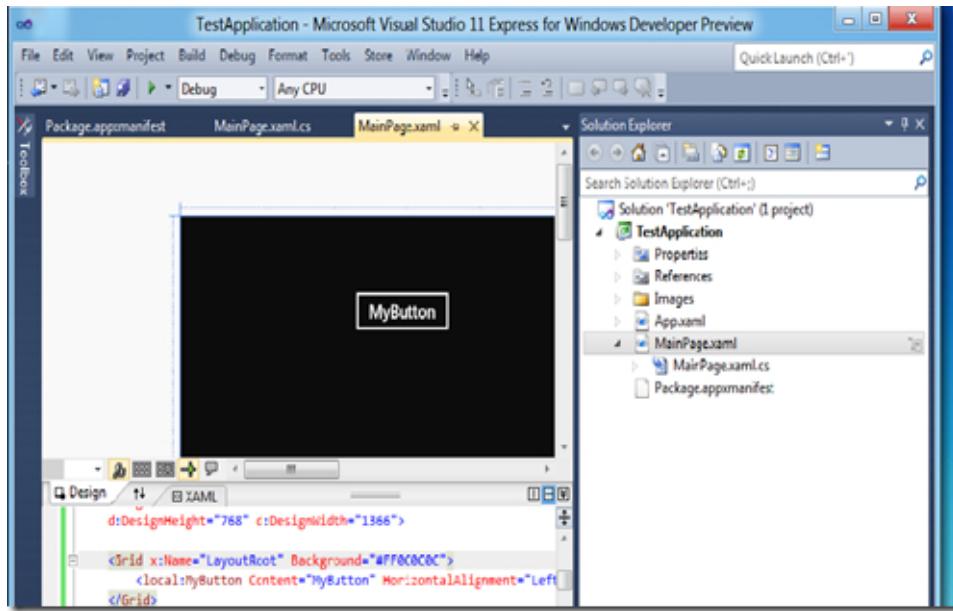
Incidentally, there's a [4]fantastic thread of comments associated with one of the [5]Channel9 GoingNative episodes which discusses the implementation strategy that Microsoft took.

1. <http://tirania.org/blog/archive/2011/Dec-19.html>
  2. <http://code.google.com/p/sharpdx/>
  3. <http://code4k.blogspot.com/2010/10/managed-netc-direct3d-11-api-generated.html>
  4. <http://channel9.msdn.com/Shows/C9-GoingNative/GoingNative-3-The-CCX-Episode-with-Marian-Luparu>
  5. <http://channel9.msdn.com/>
- 

### **6.1.3 Let's see how you run (2012-01-07 09:50)**

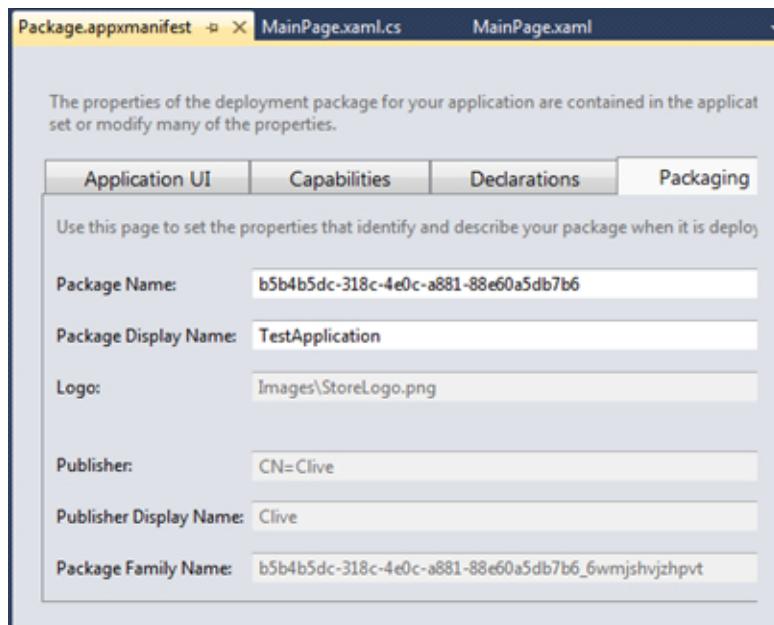
It's always good to be able to attach a debugger to a program to see how it does its work – animating code makes it far easier to understand than just reading through static source. While trying to get a better understanding of WinRT, I came across the [1]following tool which lets you attach a debugger to a system launched WinRT package from just after it has started up.

If we take a simple WinRT Silverlight application.



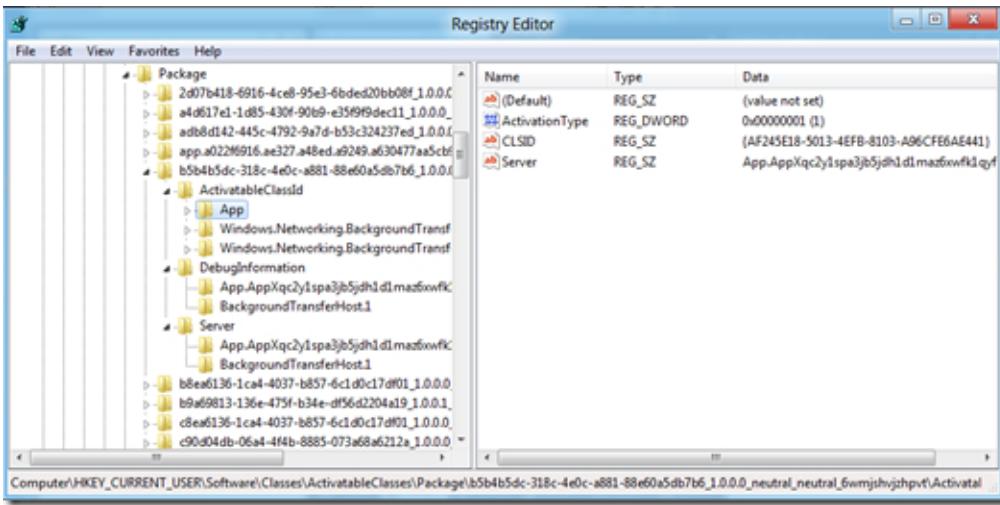
As we know, this type of application is compiled into a package. A package is identified using a guid.

[3]



It is registered into the registry of the machine.

[4]



Using the details about the package, the WinRT tool lets you associate a debugger with it. I downloaded cdb onto my virtual machine, and set that up as the associated debugger.

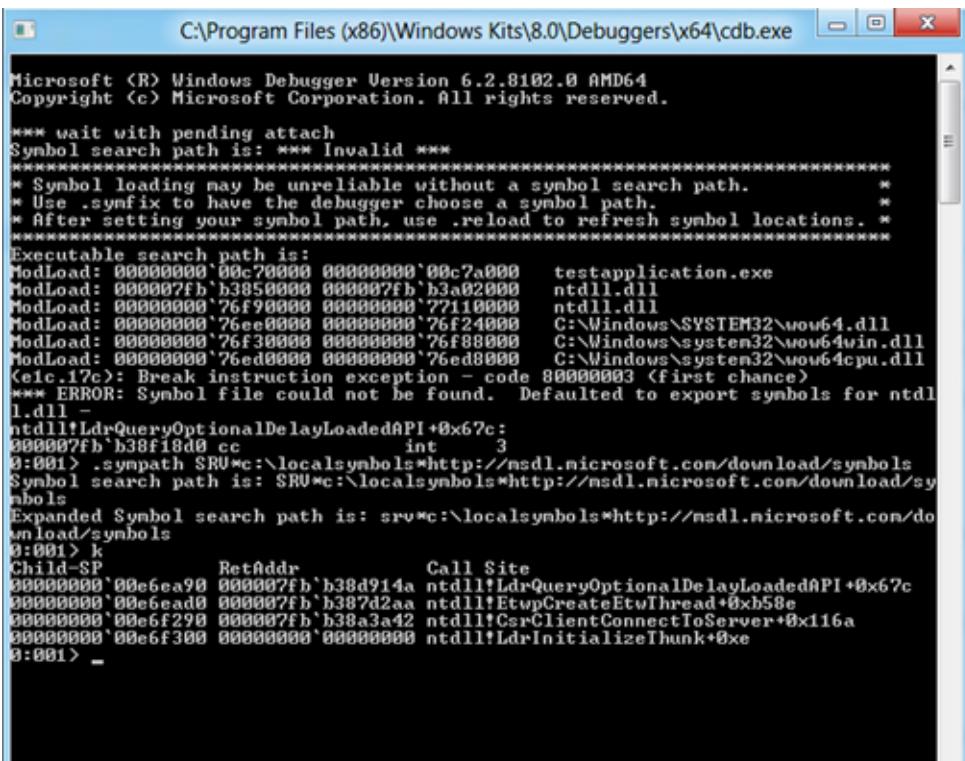
[5]

```
C:\Users\Clive\Desktop>WinRTDebug.exe -d b5b4b5dc-318c-4e0c-a881-88e60a5db7b6_1.0.0.0_neutral_neutral_6wmjshvzhpvt "C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x64\cdb.exe"
WinRTDebug v1.1 by Raffaele Rialdi, 2011 - http://www.iamraf.net
Project repository: http://WinRT.codeplex.com
This utility enable debugging the activation of WinRT Metro applications

Enable debugging
    b5b4b5dc-318c-4e0c-a881-88e60a5db7b6_1.0.0.0_neutral_neutral_6wmjshvzhpvt
    C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x64\cdb.exe
Command executed successfully
```

Launching the application in the Metro interface leads to an instance of cdb.

[6]



We can set some breakpoints to see when the Windows Runtime is being initialised.

[7]

```
0:001> bp combase!RoInitialize
Bp expression 'combase!RoInitialize' could not be resolved, adding deferred bp
```

We see this is happening as the CLR is starting up

[8]

```
Breakpoint 0 hit
combase!RoInitialize:
768ef3cd 8bff          mov     edi,edi
0:000:x86> k
ChildEBP RetAddr
006df5d0 72caa227 combase!RoInitialize
006df620 72b9005e clrThread::SetApartment+0x243
006df644 72cb4237 clr!SystemDomain::SetThreadApartmentState+0x8f
006df9d0 73d2c051 clr!CorHost2::ExecuteMain+0x248
006dfc44 73d12849 mscoreei!InvokeAppXMain+0x2bd
006dfc80 73f47f16 mscoreei!_CorExeMain+0x7e
006dfc90 73f44de3 MSCOREE!ShellShim__CorExeMain+0x99
006dfc98 76b99391 MSCOREE!_CorExeMain_Exported+0x8
006dfca4 26ff31b0 KERNEL32!BaseThreadInitThunk+0xe
006dfce4 76ff3183 ntdll!_RtlUserThreadStart+0x6f
006dfcf0 00000000 ntdll!_RtlUserThreadStart+0x1b
```

We can look at the start up environment of the dll that was generated for the application.

[9]

```
0:000:x86> .reload
Reloading current modules
-----
0:000:x86> !peb
Mod64 PEB32 at 7f41a000
InheritedAddressSpace: No
ReadImageFileExecOptions: No
BeingDebugged: Yes
ImageBaseAddress: 0000000000550000
Ldr 0000000077090180
*** _PEB_LDR_DATA type was not found...
*** unable to read Ldr table at 0000000077090180
SubSystemData: 0000000000000000
ProcessHeap: 0000000000000000
ProcessParameters: 0000000000000110
*** _CURDIR type was not found...
WindowTitle: '< Name not readable >'
ImageFile: '< Name not readable >'
CommandLine: '< Name not readable >'
DllPath: '< Name not readable >'
Environment: 0000000000000000
        Unable to read Environment string.

Mod64 PEB at 000000007f413000
InheritedAddressSpace: No
ReadImageFileExecOptions: No
BeingDebugged: Yes
ImageBaseAddress: 0000000000550000
Ldr 0000007fbb397c7e0
*** unable to read Ldr table at 0000007fbb397c7e0
SubSystemData: 0000000000000000
ProcessHeap: 0000000000040000
ProcessParameters: 00000000000a413a0
CurrentDirectory: 'C:\Windows'
WindowTitle: 'b5b4b5dc-318c-4e0c-a881-88e60a5db7b6_6umjshv,jzhput#app'
```

[10]

```
ImageFile: 'C:\Users\Clive\AppData\Local\Packages\b5b4b5dc-318c-4e0c-a881-88e60a5db7\h6US.Debug.AnyCPU.Clive\testapplication.exe'
CommandLine: '"C:\Users\Clive\AppData\Local\Packages\b5b4b5dc-318c-4e0c-a881-88e60a5db7\h6US.Debug.AnyCPU.Clive\testapplication.exe" -ServerName:App.AppXqc2yispa3jb5jdhidinaz6xwfk1qyfw0.mca'
DllPath: 'C:\Users\Clive\AppData\Local\Packages\b5b4b5dc-318c-4e0c-a881-88e60a5db7\b6US.Debug.AnyCPU.Clive'
Environment: 00000000000a408b0
    ALLUSERSPROFILE=C:\ProgramData
    APPDATA=C:\Users\Clive\AppData\Roaming
    CommonProgramFiles=C:\Program Files\Common Files
    CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
    CommonProgramFiles(x64)=C:\Program Files\Common Files
    COMPUTERNAME=TESTPC
    ComSpec=C:\Windows\system32\cmd.exe
    FZ_NO_HOST_CHECK=No
```

The real start up happens when an instance of the activation factory is created.

[11]

```

0:000:x86> bp combase!RoGetActivationFactory
0:000:x86> g
Breakpoint 0 hit
combase!RoInitialize:
760ef3cd 8bff        mov     edi,edi
0:002:x86> k
ChildEBP RetAddr
0472f868 72caa227  combase!RoInitialize
0472f8bc 72b772b9  clr!Thread::SetApartment+0x243
0472f8dc 72b58927  clr!Thread::DoExtraWorkForFinalizer+0x23
0472f924 72a92e7b  clr!WKS::GCHeap::FinalizerThreadWorker+0x222
0472f988 72c5ac35  clr!CrstBase::Enter+0x98
0472faa4 72c598d9  clr!Thread::HasStarted+0x453
0472fb44 76b99391  clr!Thread::intermediateThreadProc+0x4d
0472fb50 76ff31b0  KERNEL32!BaseThreadInitThunk+0xe
0472fb90 76ff3183  ntdll_76f90000!_RtlUserThreadStart+0x6f
0472fba8 00000000  ntdll_76f90000!_RtlUserThreadStart+0x1b
0:002:x86> g
ModLoad: 00000000 00b0a000  System.Runtime.dll
ModLoad: 00b10000 00b1a000  System.Runtime.dll
ModLoad: 6ce20000 6ce8c000  C:\Windows\Microsoft.NET\Framework\v4.0.30319\clrjit.dll
ModLoad: 676e90000 676eb000  C:\Windows\SysWOW64\WinTypes.dll
ModLoad: 00b30000 00b41000  image00b30000
ModLoad: 00e00000 00e11000  image00e00000
ModLoad: 00b30000 00b33000  image00b30000
ModLoad: 00b40000 00b43000  image00b40000
ModLoad: 00b30000 00b34000  image00b30000
ModLoad: 00c30000 00c34000  image00c30000
Breakpoint 1 hit
combase!RoGetActivationFactory:
760fe440 8bff        mov     edi,edi
0:000:x86> k
ChildEBP RetAddr
006ded7c 72d2d4fa  combase!RoGetActivationFactory
006df0b8 00b3013f  clr!StubHelpers::GetWinRTFactoryObject+0x277
WARNING: Frame IP not in any known module. Following frames may be wrong.
006df14c 72a93570  0xb3013f
006df150 72a9c590  clr!CallDescrWorker+0x34
006df180 72aa1df9  clr!SigParser::GetElemType+0x2f
006df2a0 ffffffff  clr!CallDescrWorkerWithHandler+0x2c
006df2b4 76ff1fa8  0xfffffff
006df3dc 72be9dia  ntdll_76f90000!RtlpAllocateHeap+0xf75
006df644 00000000  clr!Assembly::ExecuteMainMethod+0xfa

```

Now we have a tool for getting into the code before it runs, we have the ability to do more investigation of what is going on. I hope to blog more about that in the future.

The author of the WinRT tool has some really interesting blog posts on his initial reaction to WinRT: [12]why WinRT, [13]mixing languages, [14]packages, [15]registration and [16]contracts and activation.

1. <http://winrt.codeplex.com/>
2. <http://clivetong.files.wordpress.com/2012/01/project.png>
3. <http://clivetong.files.wordpress.com/2012/01/package.png>
4. <http://clivetong.files.wordpress.com/2012/01/registry1.png>
5. <http://clivetong.files.wordpress.com/2012/01/setfordebug.png>
6. <http://clivetong.files.wordpress.com/2012/01/debuggerstarted.png>
7. <http://clivetong.files.wordpress.com/2012/01/setbp.png>
8. <http://clivetong.files.wordpress.com/2012/01/roinitialize.png>
9. <http://clivetong.files.wordpress.com/2012/01/peb1.png>
10. <http://clivetong.files.wordpress.com/2012/01/peb2.png>
11. <http://clivetong.files.wordpress.com/2012/01/debug.png>
12. <http://www.iamraf.net/News/why-winrt-an-object-oriented-api>
13. <http://www.iamraf.net/News/mixing-languages-with-winrt>
14. <http://www.iamraf.net/News/essential-winrt-part-1-the-package>
15. <http://www.iamraf.net/News/essential-winrt-part-2-the-registration>
16. <http://www.iamraf.net/News/essential-winrt-part-3-contracts-and-activation>

#### **6.1.4 Crystal balls out (2012-01-10 10:00)**

At the start of the year, many bloggers start making their list of predictions for the coming year. One set of predictions I always love to read are those of [1]Ted Neward. I've read his predictions for many years, and all though they aren't always completely right, his well reasoned predictions are usually close. Also, I have a real soft spot for Lisp based languages, and so his number one prediction for this year obviously caught my attention. It will be really interesting to see if the .NET world ever moves to more dynamic languages. Ruby made a splash a few years back, but seems to have been forgotten in more recent times. Javascript is making its way on to the .NET platform in Windows 8, though it will only be one of the options.

1. <http://blogs.tedneward.com/2012/01/02/Tech+Predictions+2012+Edition.aspx>
- 

#### **6.1.5 Exceptional behaviour- I think not! (2012-01-12 13:25)**

Take a standard windows forms application in .NET, and modify the start up code to the following.

[1]

```
var form = new Form1();
form.Load += delegate { throw new ApplicationException(); };
Application.Run(form);
```

If you run the code on a x64 architecture you see the exception being thrown, but the Form is displayed nevertheless. If you run on x86, the exception causes the application to exit and moreover you can catch the exception if you wrap the call to Application.Run inside a try...catch...

I came across the explanation while I was researching Windows SEH on [2]this blog. The older versions of the OS had difficulty propagating structured exceptions across the re-entry from kernel mode which happens in order to raise the Load event. I guess the lesson to learn is to always keep an eye on the Output window when you are running your application in order to verify that you are seeing the exceptions you expect... if you don't, you might find exceptions being silently swallowed, which could lead to all kinds of unexpected corruption in your application's state.

1. <http://clivetong.files.wordpress.com/2012/01/exceptional.jpg>
  2. <http://blog.paulbetts.org/index.php/2010/07/20/the-case-of-the-disappearing-onload-exception-user-mode-callback-exceptions-in-x64/>
-

## 6.1.6 Lose the fat-it's slowing you down (2012-01-19 17:44)

### [1]The Lean Startup by Eric Ries

I haven't read a lot of business books, but one of the Red Gate CEOs bought over a hundred copies of the book which he offered to the employees to read. I eventually got hold of one of the copies – they disappeared quickly – and read it at the start of the Xmas vacation.

The book is targeted at technology companies with Ries' thesis being that product development should happen in a more scientific way. Using modern technologies it is easy to quickly develop product variants, and with the use of automated testing we can quickly verify that the variant can be safely launched to a subset of the customer base. We are therefore in a position to test out theories about what our customers want. Customers are notoriously bad at knowing what it is they want a product to do. If you ask them, then sure they will describe all kinds of requirements, but until you put the software in front of them they find it very hard to imagine what it will really be like to use that product.

In this new development model, we can generate a hypothesis, build variants of the product to test this (with data from each user's experience automatically sent back to the product developer) and then deploy this to a subset of our customer base. These experiments allow us to completely change the direction of our product development, an act that Ries calls a pivot. We may find out that they really want something much simpler than our original idea, or might even find that there is a different product that they would prefer.

As well as a scientific experiment based style of product development, the book also has some things to say about more agile development processes, in particular the 5 whys. When a problem happens, it is all too easy to jump at a solution by only answering the first "why". The 5 whys technique strives to make the implementer find the true root causes, rather than the first thing that can be fixed. Why did it happen? Why did that happen? Why did that happen? Why did that happen? Why did that happen and what can we do about it?

This technique can take us back to a far less specific cause of a problem. In the book, the example is that a bug on a web site can lead us into realising that we need to offer more training to our developers. This looks like something that will cost a lot of time to fix, but Ries argues that this should be solved by using a small amount of effort to start towards the solution. If the same root cause is found again, then we'll spend a little more time on the solution... but if it doesn't happen again, then this automatically avoids us wasting time solving a problem that isn't. This is his scientific, measure and react philosophy applied to the development process itself.

The book is fairly short and thought provoking - I can see how the technique works well with a start up or a new product. Some of the scientific method is taken a little too far with terms like statistically significant being thrown around a lot without saying how we determine this. The real life examples from the author's experience at IMVU and his experience consulting at other companies make the described techniques much more plausible.

I know that we'll be attempting to apply these techniques at work. Maybe you should too.

1. [http://www.amazon.co.uk/Lean-Startup-Innovation-Successful-Businesses/dp/0670921602/ref=sr\\_1\\_1?ie=UTF8&qid=1326547522&sr=8-1](http://www.amazon.co.uk/Lean-Startup-Innovation-Successful-Businesses/dp/0670921602/ref=sr_1_1?ie=UTF8&qid=1326547522&sr=8-1)

### **6.1.7 To the future and beyond! (2012-01-19 17:44)**

[1]9 Algorithms that changed the Future by John MacCormick

I thought this book was really good. It is isn't very technical but explains several interesting algorithms using simplified examples in a way that makes the algorithm very clear. In some cases, various technicalities are missed out, but the general idea behind the algorithm is explained brilliantly.

What makes the book even better is the fantastic choice of algorithms that the author has picked. The chosen algorithms really are things that we have our computers do every day. Indexing web pages so that they can be efficiently searched and page ranking are the two algorithms that are covered first. Public key cryptography, used to secure web connections, and digital signatures, used for ensuring that a program we download really came from the expected publisher, are both covered really well. Transferring data across the internet relies on data compression and some use of error correcting codes. Most of the data stored around the world is stored in databases, so its important that the consistency of transactions can be maintained.

There are two other algorithms that are covered that don't really fit into this pattern. There is a discussion of algorithms for pattern matching, in particular using neural networks to learn from examples in a scenario where there is no real algorithm. And a very good chapter on computability, covering the proof that there are things that a computer cannot do.

The author emphasises the clever ideas behind the algorithms, and really communicates a love for computer science without the book becoming too technical at any stage.

1. [http://www.amazon.co.uk/Nine-Algorithms-That-Changed-Future/dp/0691147140/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1326549118&sr=1-1](http://www.amazon.co.uk/Nine-Algorithms-That-Changed-Future/dp/0691147140/ref=sr_1_1?s=books&ie=UTF8&qid=1326549118&sr=1-1)
- 

### **6.1.8 You've got loads of things in that container (2012-01-22 12:48)**

[1]Dependency injection in .NET by Mark Seeman

First, let me say that this is a really brilliant book. Lots of technical discussion backed up with lots of practical examples, though perhaps with a few too many metaphors associated with cooking.

Software typically runs in some kind of container. Our programs run inside a virtualised environment where we interact with the outside world via an abstraction – the container takes some of the detail away from us, presenting us with a simple abstraction that we can use. Dependency injection is the concept of getting a container to handle this wiring and object lifetime management for us. We use a dependency injection container by registering with it the appropriate types, and then it allows us to pull out instantiated instances of the various object types with all of the connections made for us.

The introductory chapter of the book makes this setting very clear with a “Hello world” example of dependency injection which illustrates the three dimensions of dependency injection – object composition, object lifetime managements and interception. To be honest, this chapter

provides the clearest introduction to dependency injection that I have ever read – I've not seen dependency injection studied via these three views before. The book advertises the many benefits of dependency injection in terms of better application architecture and much better application testability.

The author looks at these three pillars and describes how they are implemented across the various types of container. For example composition can be done using auto-wiring, XML configuration or via fluent code, and the author spends some time discussing the benefits of each approach, with examples of each from a couple of sample containers.

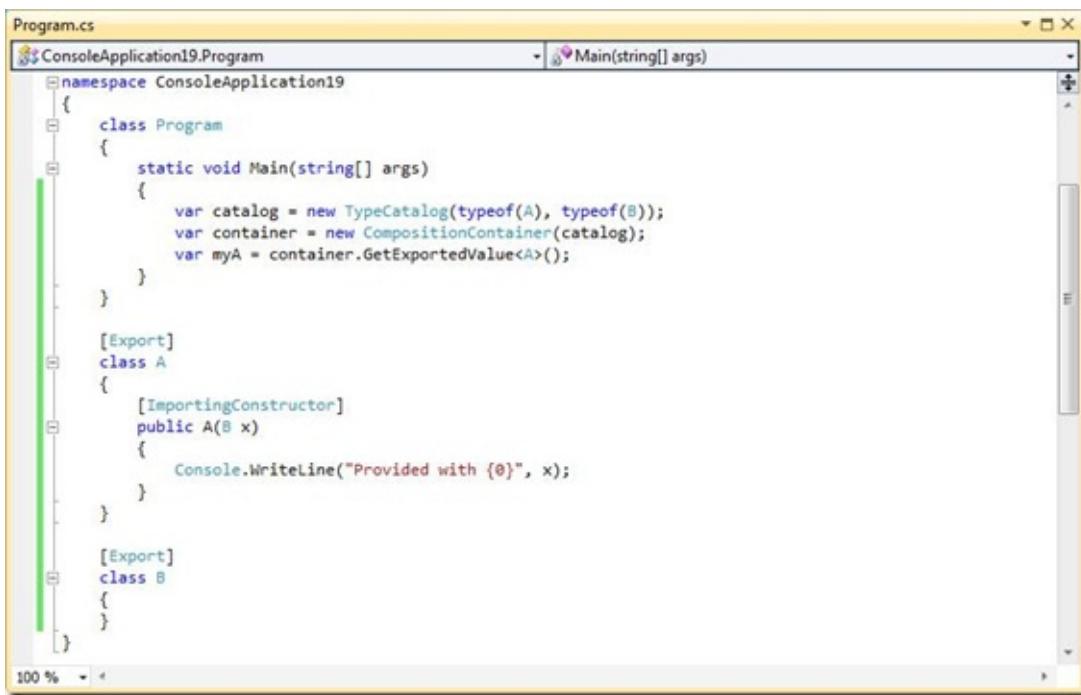
In order to get people using dependency injection, the author demonstrates how to find a composition root, a place where the container is instantiated and configured, in a number of scenarios – console applications, WPF and WCF applications and web applications too. He also explains a number of common anti-patterns and how to avoid them. There's also a chapter on some useful refactoring to make it possible to use dependency injection, typically by using an abstract factory to decouple the concrete object from the dependency.

Using the three comparison areas, the author spends the second half of the book looking at 6 different containers ranging from Castle Windsor to MEF (which only implements one of the three pillars of dependency injection), by way of StructureMap, Spring.NET, Autofac and Unity, explaining the differences between the containers and their designs.

Along the way we see lots of practical examples and lots of sample code together with a lot of relevant and interesting discussion.

Since there is a version of MEF built into .NET 4, I know that I need to spend more time looking at it... indeed extensibility of Visual Studio's editor is largely by using MEF exports to add our code to extensibility points. There is a good overview of it [2]here. The screenshot below shows a simple example of using MEF by registering two types with a catalog and then into the container. When an instance of the class A is constructed, MEF can pass in a new instance of the type B without any user intervention.

[3]



The screenshot shows the Visual Studio code editor with the file 'Program.cs' open. The code defines a 'Program' class with a static 'Main' method. Inside 'Main', a 'TypeCatalog' is created for types A and B, and a 'CompositionContainer' is initialized with this catalog. An instance of class A is then retrieved from the container. Below this, classes A and B are defined. Class A has an '[Export]' attribute and an importing constructor that takes an instance of B. Class B also has an '[Export]' attribute. The code editor interface is visible, including the title bar, tabs, and status bar at the bottom.

```
Program.cs
ConsoleApplication19.Program
namespace ConsoleApplication19
{
    class Program
    {
        static void Main(string[] args)
        {
            var catalog = new TypeCatalog(typeof(A), typeof(B));
            var container = new CompositionContainer(catalog);
            var myA = container.GetExportedValue<A>();
        }
    }

    [Export]
    class A
    {
        [ImportingConstructor]
        public A(B x)
        {
            Console.WriteLine("Provided with {0}", x);
        }
    }

    [Export]
    class B
    {
    }
}
```

MEF falls short of being a proper dependency injection container, but does offer extra facilities for extensibility scenarios. It will be interesting to see where Microsoft take it in the future.

1. [http://www.amazon.co.uk/Dependency-Injection-NET-Mark-Seemann/dp/1935182501/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1326618088&sr=1-1](http://www.amazon.co.uk/Dependency-Injection-NET-Mark-Seemann/dp/1935182501/ref=sr_1_1?s=books&ie=UTF8&qid=1326618088&sr=1-1)
  2. <http://msdn.microsoft.com/en-us/library/dd460648.aspx>
  3. <http://clivetong.files.wordpress.com/2012/01/mefexample.jpg>
- 

### **6.1.9 I'm still standing (2012-01-29 11:40)**

Ted Neward has been trying to stir again with a recent post on the subject [1]Is Programming Less Exciting Today? Also these lines, there have been several recent times when I've heard people say that .NET is dead - [2].NET Rocks and [3]various developers.

To me, nothing is further from the truth. .NET has made it possible for a mass of developers to use higher levels of abstraction for writing applications by way of the BCL and the various managed wrappers for the [4]Win32 API, and things are only going to get better in the future with the [5]WinRT platform offering an object based view of the underlying operating system. People have been talking about doing this for years - remember [6]the Longhorn days when we were promised .NET implementations of the file system and more exposure of the OS via managed code. Ok, so things aren't going to be fully managed - Microsoft seem to have dropped back to using COM and its reference counting in preference to going the fully managed route, but they have taken care of automatically generating managed wrappers from the winmd files so the OS will be equally accessible from managed and unmanaged worlds.

Mobile computing is really shaking up the field and we are at a stage where lots of old ideas, which were too expensive in performance terms to implement in their day, are now feasible. It may well be a curse, but we live in interesting times, and I for one can't wait to see what's around the corner.

1. <http://blogs.tedneward.com/2012/01/25/Is+Programming+Less+Exciting+Today.aspx>
  2. <http://www.dotnetrocks.com/default.aspx?showNum=730>
  3. <http://ilker.de/not-leaving-.net>
  4. <http://en.wikipedia.org/wiki/Win32>
  5. <http://en.wikipedia.org/wiki/WinRT>
  6. <http://www.winsupersite.com/article/faqtip/windows-longhorn-faq>
- 

### **6.1.10 It's not a game you know (2012-01-30 05:00)**

I've been promising to get my son's game running on a mobile phone for some time. When he was younger we spent some time writing a version of PacMan that we could run on his Nintendo

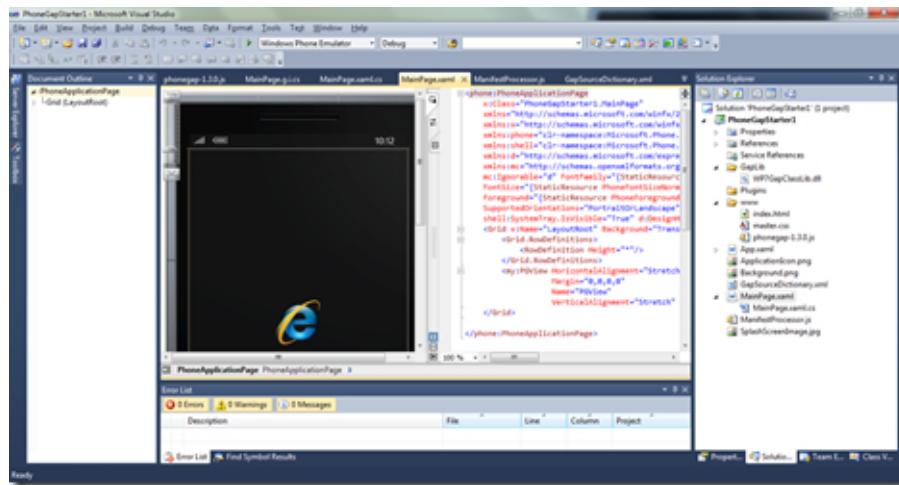
DS, though this had to be [1]written in C and we needed to use an [2]R4 chip to allow us to boot the game on the hardware. He's older now, with much grander ideas, so I spent some time looking at writing a game for Windows Phone 7 using Silverlight. The XAML skills I would have picked up doing this would have been useful, but the lack of portability to other platforms has been holding me back.

[3]HTML5 could be the answer. The idea of writing one version of the game, and then being able to run it on several mobile phone variants is certainly attractive, and [4]PhoneGap is a recent technology that has promised this ability, to take a HTML5/Css/Javascript application and move it easily on to another platform.

PhoneGap is great in that it offers a standard library which runs on the different variants of mobile phones, simulating library functions when they aren't available natively. It still leaves the nagging question of how PhoneGap is actually implemented, so this weekend I pulled out [5]Reflector and did some investigation.

The PhoneGap project that comes as a Visual Studio template is fairly straightforward. As a user of the the template, you simply place your assets (html, css and javascript) into a directory (www). At build time these assets are added as resources into the resulting application, and the PhoneGap runtime takes care of unpacking and starting the code inside the browser instance that it manages.

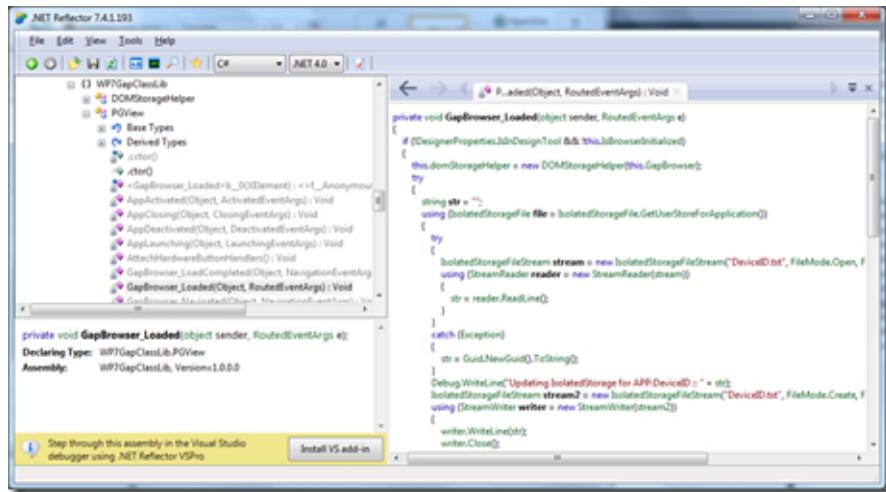
[6]



The finished project runs inside the web browser instance, which is managed by the C # code in WP7GapClassLib.dll. PGView is the user control which wraps the web browser.

[7]

614



In order to understand exactly what it was doing, I wanted to write my own version, so I took a standard Silverlight project for Windows Phone, added a web browser control to it, and changed the property of the browser to make it scriptable.

[8]

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <phone:WebBrowser Height="40" HorizontalAlignment="Left" Margin="65,29,0,0" Name="webView1"
        VerticalAlignment="Top" Width="291" IsScriptEnabled="True" />
    <Button Content="Press to start" Height="79" HorizontalAlignment="Left" Margin="119,156,0,0" Name="button1"
        VerticalAlignment="Top" Width="214" Click="button1_Click" />
    <TextBlock Height="44" HorizontalAlignment="Left" Margin="65,90,0,0" Name="textBlock1" Text=" "
        VerticalAlignment="Top" Width="280" />
</Grid>
```

The trick that PhoneGap uses, is to write the data from the resources into isolated storage, so that they are then accessible to the browser. We do the same in the following code, effectively simulating the PhoneGap start sequence.

[9]

```
 MainPage.xaml.cs
using System;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Navigation;
using Microsoft.Phone.Controls;
using Microsoft.Phone.Shell;

public partial class MainPage : PhoneApplicationPage
{
    // Constructor
    public MainPage()
    {
        InitializeComponent();
        using (IsolatedStorageFile file2 = IsolatedStorageFile.GetUserStoreForApplication())
        {
            using (IsolatedStorageFileStream stream2 = file2.OpenFile("foo.htm", FileMode.Create))
            {
                var data = Encoding.Unicode.GetBytes(@"
<head>
<script type='text/javascript'>
    function TestFunction()
    {
        setTimeout('Callback()', 1000);
    }
    var count = 0;
    function Callback()
    {
        count++;
        outputID.innerHTML = count.toString();
        window.external.notify(count.toString());
        setTimeout('Callback()', count + 1000);
    }
</script>
</head>
<body>
<div id='outputID' style='color:Red; font-size:10px'>
    Hello from HTML document with script!
</div>
</body>
</html>");;
                stream2.Write(data, 0, data.Length);
            }
        }
        webView1.Navigate(new Uri(@"foo.htm", UriKind.Relative));
        webView1.ScriptNotify += new EventHandler<EventArgs>(webView1_ScriptNotify);
    }
}
```

Then all we need are a couple of event handlers, one that calls into the javascript to start it running, and another that handles the callback from the javascript into the managed code.

[10]

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    webBrowser1.InvokeScript("TestFunction");
}

void webBrowser1_ScriptNotify(object sender, NotifyEventArgs e)
{
    textBlock1.Text = e.Value;
}
```

With that in place, we have an application that displays a count in javascript and displays the same value in a TextBlock in the managed world.



[11]

Unfortunately we only have the ability to pass strings across the language boundary, so we may need to do some marshalling to interoperate with rich data types. Having the ability to jump out into the multi-threaded managed world might be useful for some applications, if the single threaded Javascript world doesn't have the necessary power for the application code. This means we have the option at a later stage of forgetting about portability in order to get better performance.

This does sound like a very viable way to write an application though and I'm keen to start writing – I'll let you know the result.

1. [http://double.co.nz/nintendo\\_ds/](http://double.co.nz/nintendo_ds/)
2. [http://en.wikipedia.org/wiki/R4DS#YushenDS\\_Card.2C\\_R4DS.2C\\_M3\\_DS\\_Simply.2C\\_and\\_their\\_clones](http://en.wikipedia.org/wiki/R4DS#YushenDS_Card.2C_R4DS.2C_M3_DS_Simply.2C_and_their_clones)
3. <http://www.html5rocks.com/en/>
4. <http://phonegap.com/>
5. <http://www.reflector.net/>
6. <http://clivetong.files.wordpress.com/2012/01/phonegapproject.png>
7. <http://clivetong.files.wordpress.com/2012/01/reflectoronpgobegap.png>
8. <http://clivetong.files.wordpress.com/2012/01/appxaml.png>
9. <http://clivetong.files.wordpress.com/2012/01/constructor.png>
10. <http://clivetong.files.wordpress.com/2012/01/eventhandlers.png>
11. <http://clivetong.files.wordpress.com/2012/01/finaldisplay.png>

## 6.2 February

### 6.2.1 Watch out for the ceiling! (2012-02-14 20:43)

The consumer preview of Windows 8 is due for the end of the month, and since my wife recently bought me a new laptop for my birthday, which is powerful enough to run a VM, I thought it was time to get out the //BUILD release of Windows 8 and have more of a play. More importantly, with the recent [1]C++ Renaissance at Microsoft it's a good time to brush up those C++ skills too, particularly when writing COM code.

First we create a C++ WinRT component.

```
using namespace Windows::Foundation;  
  
namespace WinRTComponentDll1  
{  
  
    public ref class WinRTComponent sealed  
    {  
        public:  
            WinRTComponent();  
            ~WinRTComponent();  
  
            int Method(int i);  
    };  
}
```

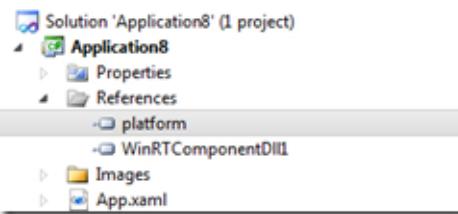
[2]

With the implementation

```
int WinRTComponent::Method(int i)  
{  
    return i*10;  
}
```

[3]

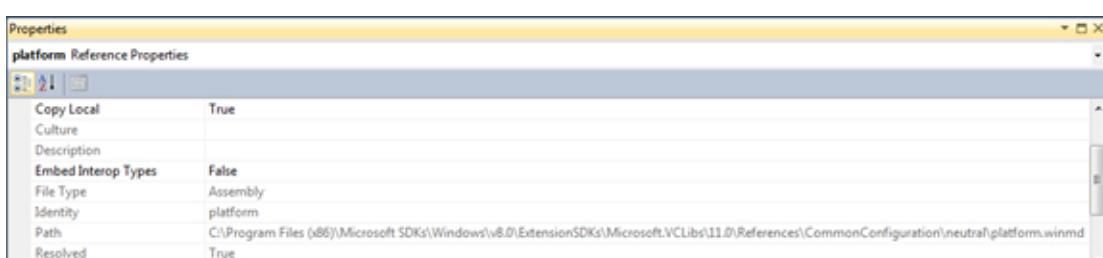
It's important that you build this as x86. Having done the system will have created a winmd file next to the assembly's dll. We can set this winmd file as a reference from a C# project and then use the component from the other language. This C++ assembly references some other interfaces in the platform winmd file, so you also need to set a reference to that - I think this is a bug that will be fixed in the next release.



[4]

The platform winmd file isn't always easy to find as it is part of the SDK.

[5]



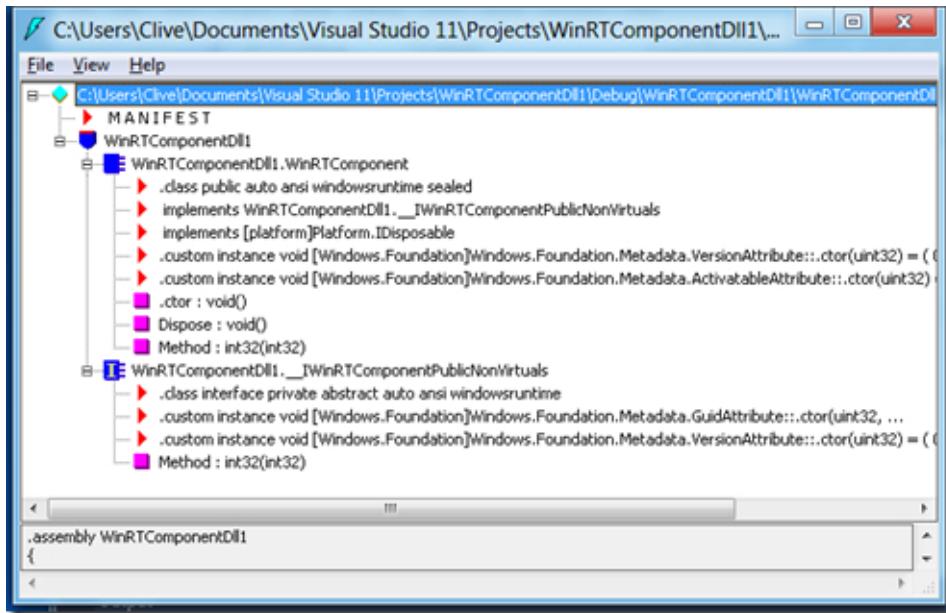
With that done, it's easy to make instances of the C++ class inside the C# application.

[6]

```
var x = new WinRTComponentDll1.WinRTComponent();
TextBox1.Text = x.Method(3).ToString();
```

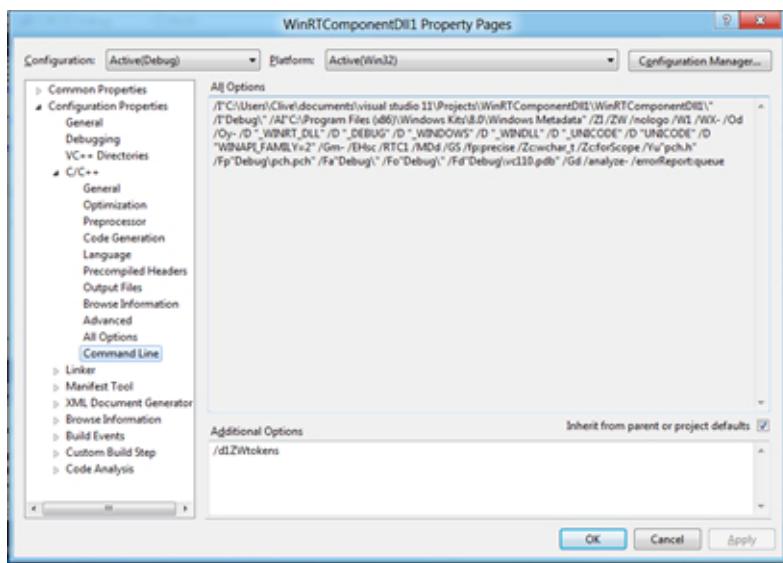
Winmd files are really metadata files in the format used for CLR metadata, and hence tools like ILDASM and Reflector work well with it.

[7]



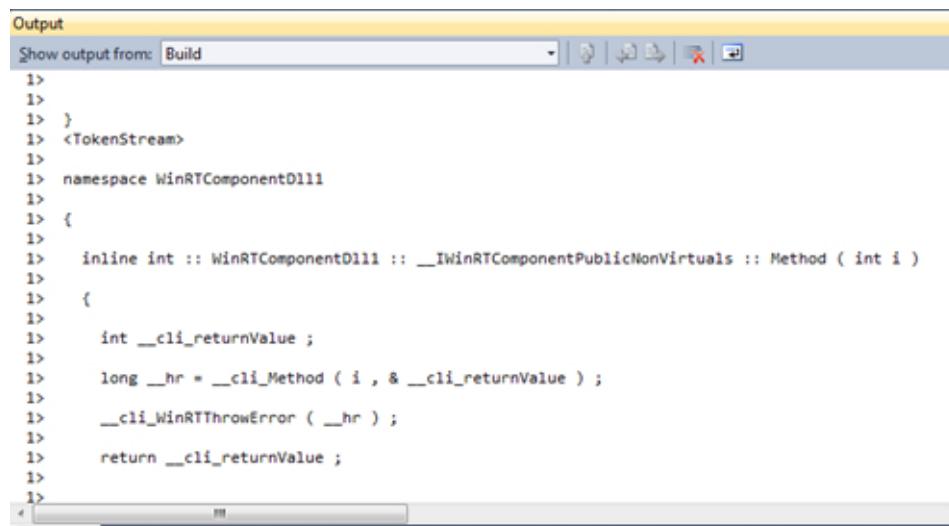
Whereas setting a reference to a winmd file in the managed world causes the CLR to generate the necessary wrappers, the Visual Studio C++ compiler has an option that allows you to get it to dump the token stream for the wrappers it creates (/d1Zwtokens).

[8]



You can then view the wrapper code in the output window of the build. Here's the way that the method named Method that we defined above is going to get called. Notice that the wrapper translates exceptions into a good old HRESULT of COM.

[9]

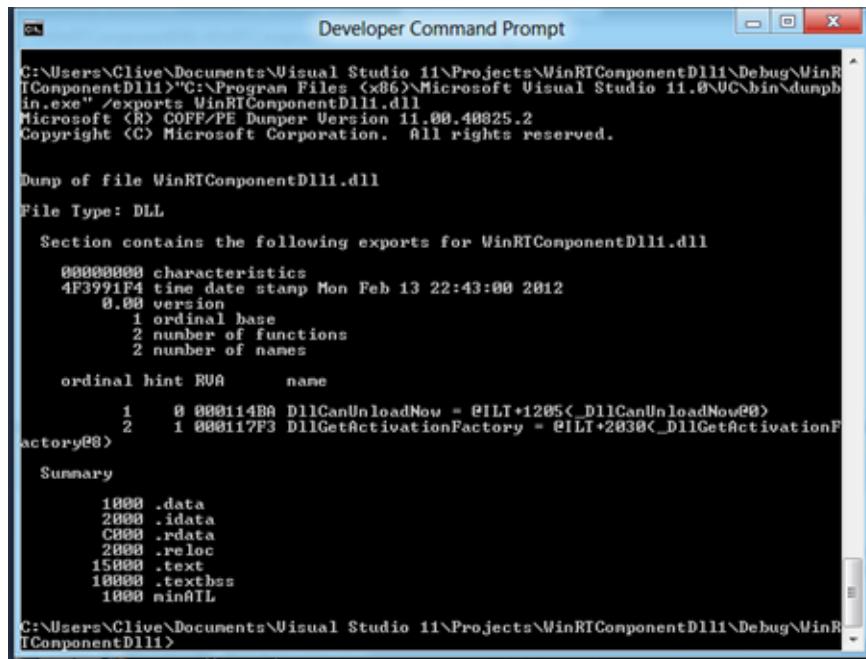


The screenshot shows the 'Output' window from Visual Studio. The title bar says 'Output'. The dropdown menu says 'Show output from: Build'. The main area contains the following C++ code:

```
1>
1>
1> }
1> <TokenStream>
1>
1> namespace WinRTComponentDll1
1> {
1> {
1>     inline int :: WinRTComponentDll1 :: __IWinRTComponentPublicNonVirtuals :: Method ( int i )
1>     {
1>         int __cli_returnValue ;
1>         long __hr = __cli_Method ( i , & __cli_returnValue ) ;
1>         __cli_WinRTThrowError ( __hr ) ;
1>         return __cli_returnValue ;
1>     }
1>
```

To see how much like a standard COM component this is, we can dump the exports and see the usual suspects, DllCanUnloadNow and DllGetActivationFactory.

[10]



The screenshot shows a 'Developer Command Prompt' window. The title bar says 'Developer Command Prompt'. The command line shows:

```
C:\Users\Clive\Documents\Visual Studio 11\Projects\WinRTComponentDll1\Debug\WinR
TComponentDll1>"C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\bin\dumpb
in.exe" /exports WinRTComponentDll1.dll
Microsoft (R) COFF/PE Dumper Version 11.00.40825.2
Copyright (C) Microsoft Corporation. All rights reserved.
```

Then it lists the exports for 'WinRTComponentDll1.dll':

```
Dump of file WinRTComponentDll1.dll
File Type: DLL

Section contains the following exports for WinRTComponentDll1.dll

00000000 characteristics
4F3991F4 time date stamp Mon Feb 13 22:43:00 2012
    0.00 version
        1 ordinal base
        2 number of functions
        2 number of names

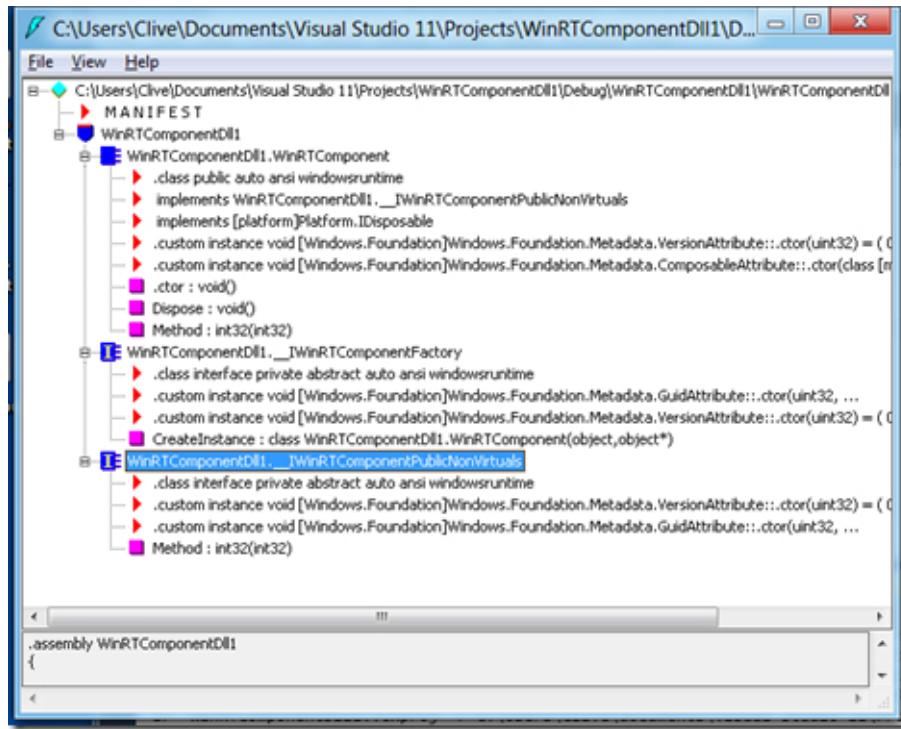
ordinal hint RVA           name
    1      0 0000114BA DllCanUnloadNow = @ILT+1205<_DllCanUnloadNow@0>
    2      1 0000117F3 DllGetActivationFactory = @ILT+2030<_DllGetActivationF
actory@0>

Summary

1000 .data
2000 .idata
C000 .rdata
2000 .reloc
15000 .text
10000 .textbss
1000 minAIL
```

I was interested to see why the class defined above is defaulted to being sealed. If you remove the sealed attribute, the winmd has an extra factory method created.

[11]



Once we do this, we start getting a MissingMethod exception when we try to consume the component via the CLR.

[12]

```

try
{
    var x = new WinRTComponentDll1.WinRTComponent();
    TextBox1.Text = x.Method(3).ToString();
}
catch (System.MissingMemberException ex)
{
    string output = ex.Message;
}

```

But not when we consume the component via C++, even though both consumers are using COM.

[13]

```

auto foo = gcnew WinRTComponentDll1::WinRTComponent();
int foo2 = foo->Method(10);

```

I assume this is a bug in the implementation, and look forward to trying this code out in the forthcoming beta of Windows 8.

It is very nice being able to consume components that were written in another language, but of course the interesting question is whether the types that you can pass around in the interfaces are rich enough to both work across lots of languages and also express enough to make the interface powerful. This is something I am going to be looking at in the coming weeks.

1. <http://channel9.msdn.com/Events/GoingNative/GoingNative-2012>
  2. <http://clivetong.files.wordpress.com/2012/02/cppcomponent1.png>
  3. <http://clivetong.files.wordpress.com/2012/02/cppcomponent2.png>
  4. <http://clivetong.files.wordpress.com/2012/02/reference1.png>
  5. <http://clivetong.files.wordpress.com/2012/02/reference2.png>
  6. <http://clivetong.files.wordpress.com/2012/02/calling.png>
  7. <http://clivetong.files.wordpress.com/2012/02/metadata1.png>
  8. <http://clivetong.files.wordpress.com/2012/02/option1.png>
  9. <http://clivetong.files.wordpress.com/2012/02/tokens1.png>
  10. <http://clivetong.files.wordpress.com/2012/02/comexports.png>
  11. <http://clivetong.files.wordpress.com/2012/02/nonsealed.png>
  12. <http://clivetong.files.wordpress.com/2012/02/missing.png>
  13. <http://clivetong.files.wordpress.com/2012/02/cppworks.png>
- 

### **6.2.2 To the future, today! (2012-02-19 08:33)**

[1]HTML5and CSS: Develop with Tomorrow's Standards Today by Brian P Hogan

This book covers the new HTML and CSS features being introduced for HTML5, but it does a lot more than that. For each feature, the author describes the feature in depth with examples, but more importantly the author lists the browsers that currently support the feature, and for those that don't he also covers how to simulate the feature, often by suggesting and demonstrating a freely available Javascript library. More importantly, the demo applications are written as progressive enhancement, starting with the basic HTML and adding Javascript in a way that would make the page usable even if the browser has Javascript turned off.

The example are all useful and practical, the explanations are really clear and the emphasis on keeping a good user experience even if Javascript is not available, all make the book a really good read.

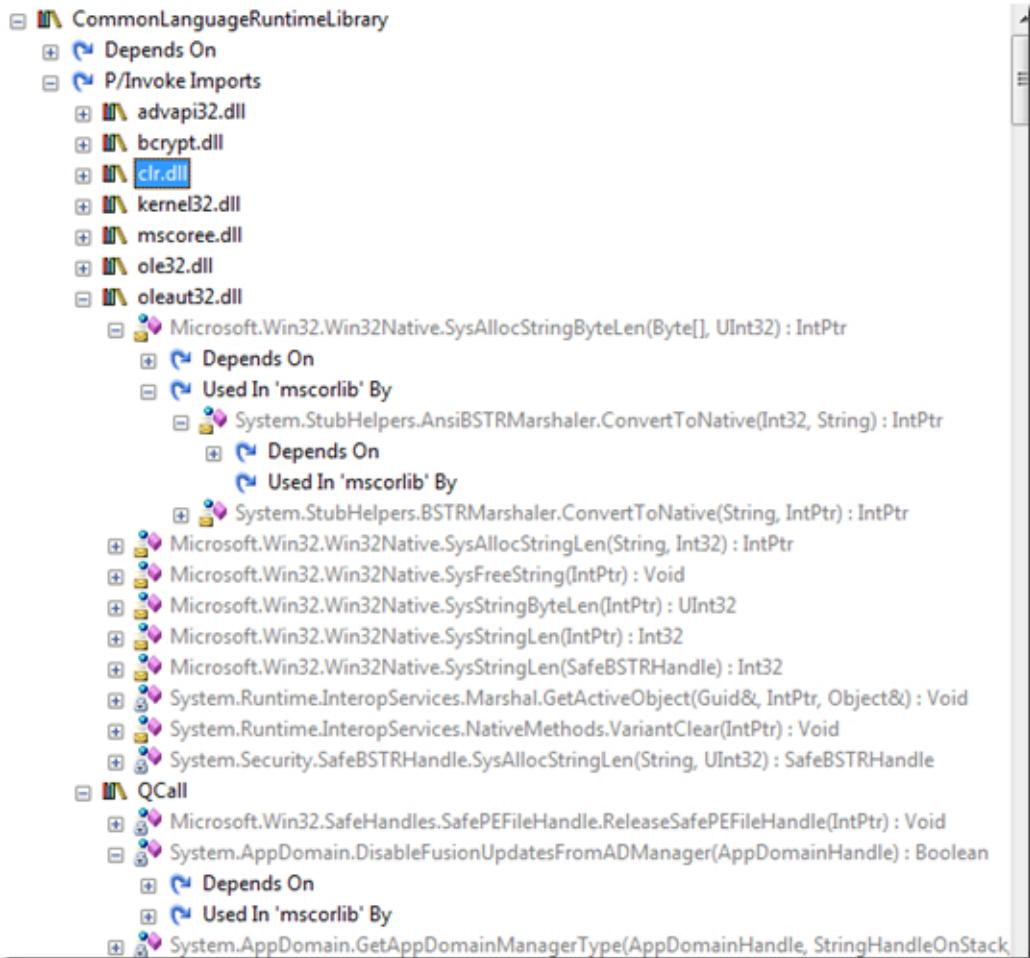
1. [http://www.amazon.co.uk/HTML5-CSS3-Tomorrows-Standards-Programmers/dp/1934356689/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1329588552&sr=1-1](http://www.amazon.co.uk/HTML5-CSS3-Tomorrows-Standards-Programmers/dp/1934356689/ref=sr_1_1?s=books&ie=UTF8&qid=1329588552&sr=1-1)
- 

### **6.2.3 Keep on searching to get your reward (2012-02-20 06:34)**

I work on [1]Reflector but whenever I go through the code I always some other useful feature that I've not used before. On Thursday I was playing around with the Analyze functionality, available on the right context menu in the assembly tree. I'd never noticed before that when you were on a module, the Analyze offered the ability to find all P/Invoked methods on the module, and then the ability to search for where these methods were used. Using this on

mscorlib is interesting, and there appear to be a lot of interesting differences between the .NET 2 and .NET 4 versions.

[2]



Whereas in .NET 2, there was a dependency on mscorwks and mscoree, in .NET 4 these have been replaced with references to clr.dll and a new QCall.dll. Following back from some of these methods, it appears that a lot more of the COM marshalling code is now exposed via managed interfaces inside System.StubHelpers. The Microsoft.Win32.Win32Native class also exposes a lot of useful functionality.

1. <http://www.reflector.net/>
2. <http://clivetong.files.wordpress.com/2012/02/reflector-analyse.png>

#### 6.2.4 It's all Greek to me (2012-02-20 08:36)

[1] Famous Problems of Geometry and How To Solve Them by Benjamin Bold

When I studied for my mathematics degree, we were in the period analytic geometry and so we didn't do a lot of study of the constructions using compass and straight edge for which the

Greeks were famous. This book fills in some of the gaps in my knowledge, covering the ways the Greeks (and Gauss for some of the later material) had for constructing lengths using a compass and a straight edge.

Chapter one covers the construction of  $ab$ ,  $a/b$ ,  $a^2$ ,  $\sqrt{a}$  and the roots of the quadratic equation  $x^2 - ax + b = 0$ , the first three using similar triangles, the  $\sqrt{a}$  using a semi-circle and [2]Thales' theorem and the roots using a circle whose diameter goes from  $(0,1)$  to  $(a,b)$ .

Chapter two covers the analytic criteria for constructability, in particular the theorem: If a cubic equation with rational coefficients has no rational root, then none of its roots is constructible. This is used in later chapters on the Delian problem and the trisection of an angle, after a chapter on the complex numbers. There is then a chapter on squaring the circle.

This is followed by a chapter on the construction of regular polygons, covering the Greeks and their construction of regular polygons of  $n$  sides where  $n = 2^m 3^{r_1} 5^{r_2}$  where  $m$  is any non-negative integer and  $r_1$  and  $r_2$  are 0 or 1. This then goes on to cover Gauss' work in this area.

The book has great explanations, and a set of exercises that take you through the proof of the constructions, and which also offer extra useful information. It gets a little dated in the end where it talks about the potential proofs of Fermat's Last Theorem, but that's the only bad thing I can say about it.

1. [http://www.amazon.co.uk/Famous-Problems-Geometry-explaining-science/dp/0486242978/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1329588938&sr=1-1](http://www.amazon.co.uk/Famous-Problems-Geometry-explaining-science/dp/0486242978/ref=sr_1_1?s=books&ie=UTF8&qid=1329588938&sr=1-1)
  2. [http://en.wikipedia.org/wiki/Thales%27\\_theorem](http://en.wikipedia.org/wiki/Thales%27_theorem)
- 

### 6.2.5 Is there an MD in the house? (2012-02-22 09:20)

We're still waiting for the beta of Windows 8, and I'm keen to get to grips with a more final version of WinRT. In the meantime, I've been doing a lot of reading of various posts that are based on the //BUILD developer preview. From my old days working with Common Lisp I know that working with a non-Microsoft platform means that you often need to dig deep to understand how to interface with the Windows platform. The Delphi guys are having just these opportunities in getting Delphi to work against WinRT and this has produced a number of very interesting blog posts, among them [1]this one which goes into some details about the items that you can find in a winmd file.

1. <http://www.thomgerdes.com/2011/12/winrt-internals-winmd-files.html>
-

## **6.2.6 The more the merrier! (2012-02-24 09:18)**

A year or two so ago I spent a lot of time looking at Clojure. It was exciting to see a modern version of Lisp making its way onto the Java platform – though this is a Lisp which places an emphasis on non-mutability and platform interoperation. Since those days, people have been working hard to get Clojure running on the CLR, and more recently there has been an effort to compile Clojure to Javascript. This dialect of Clojure, ClojureScript, is finding good real world uses.

There were two talks at recent [1]Clojure Conj, the first on the [2]ClojureScript compiler and a second on how to get ClojureScript code to interact with existing [3]Javascript libraries.

1. <http://clojure-conj.org/>
  2. <http://blip.tv/clojure/chris-houser-clojurescript-5919739>
  3. <http://blip.tv/clojure/kevin-lynagh-extending-javascript-libraries-from-clojurescript-5919758>
- 

## **6.3 March**

### **6.3.1 Never the twain (2012-03-01 20:17)**

So the Consumer Preview of Windows 8 has arrived, and it's starting to answer some of the questions that were left hanging by the //Build developer preview of the platform. In the earlier release, we seemed to be in a position where you either wrote a XAML application (think C #) or a DirectX application (think C++). In this latest release, some [1]DirectX objects can be embedded into the XAML code. Admittedly, most of the samples assume that you'll be working with the DirectX code via a component written in C++, as the DirectX library calls are not exposed as WinRT types, but at least the managed world will have the ability to mix DirectX into its traditional GUI technology.

1. <http://msdn.microsoft.com/library/windows/apps/hh825871.aspx>
- 

### **6.3.2 It's all in the details (2012-03-04 10:52)**

[1]Hacker's Delight by Henry S Warren

There are loads of books out there on the subject of algorithms, but there aren't many that look in detail at algorithms and tricks which use the magic of 2's complement arithmetic and the various shift and logic operations that have been supported by processors since the early days.

The arithmetic that is supported by a modern processor is a funny beast. Due to the choice of the most significant bit as a sign bit (which is one when the value is negative), and the choice of all zeros representing zeros, this leaves more representable numbers that are strictly negative than those that are strictly positive. This leads to the interesting effect of the negation of the most negative number (-2147483648 on a 32 bit machine) being itself. This allows all kinds of interesting tricks in the book.

Chapter one starts with tricks revolving around the representation of a number – finding the rightmost 1 bit, turning off the rightmost 1 bit and identifying the trailing zeros in the bit pattern. There's also the interesting theorem: “A function mapping words to words can be implemented with word-parallel add, subtract, and, or and not if and only if each bit of the result depends only on bits at and to the right of each input operand.” This chapter also considers the relationship between signed and unsigned comparisons, overflow detection (in the absence of a flags register set by the arithmetic operations), shifting operations and the implementation of multi-word arithmetic. There's also material that looks at which comparisons can be implemented without needing the use of branch instructions which can cause pipeline stalls on modern architectures, and also considers code sequences to be better when they exhibit instruction level parallelism.

There are then chapters on powers of 2 boundaries (rounding up and down to them), and arithmetic bounds and how they propagate through add, subtract and logical operations. Chapter 5 covers ways of counting bits, and shows off some of tricks to divide and conquer, carrying out calculations in segments of the word and later merging the results. The example below counts the number of 1 bits in the word.

```
[sourcecode language="csharp"]
uint x = 0x8007;
x = (x & 0x55555555) + ((x >> 1) & 0x55555555);
x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
x = (x & 0x0f0f0f0f) + ((x >> 4) & 0x0f0f0f0f);
x = (x & 0x00ff00ff) + ((x >> 8) & 0x00ff00ff);
x = (x & 0x0000ffff) + ((x >> 16) & 0x0000ffff);
[/sourcecode]
```

The next chapters covers tricks for searching words, for example finding a null byte string terminator in a word of ASCII characters and then algorithms for rearranging the bits of a word (for example reversing it). This is followed by chapters on multiplication and division, which covers in detail the tricks of converting a division by a constant into a multiplication (calculating the high word of the result) followed by a shift and possible adjustment by a constant. The author proves the validity of the transformation, which is something I've not seen before.

This would be enough material, but there are further chapters on implementing the elementary functions (such as integer square and cube root), floating point, unusual number bases, Hilbert's curve and formulas for the primes.

This book is a really good read. It's full of interesting and illuminating tricks, and just shows how interesting the field of compiler optimisation can be.

1. [http://www.amazon.co.uk/Hackers-Delight-Henry-Warren-Jr/dp/0201914654/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1330791151&sr=1-1](http://www.amazon.co.uk/Hackers-Delight-Henry-Warren-Jr/dp/0201914654/ref=sr_1_1?s=books&ie=UTF8&qid=1330791151&sr=1-1)

### 6.3.3 Get to the heart of the matter (2012-03-04 15:09)

There's a time in every programmer's life when he wonders if he can write his own kernel. (I'm keen on understanding how the page management facilities of an operating system can be used to implement a write barrier for garbage collection - the kind of thing that Azul does in its [1]Zing architecture) After spending some time investigating, the answer comes along - there's a lot of documentation on getting it working on Linux, but less on how to get things going on Windows. I spent the last week or so going around various tutorials and taking bits from each in order to get to a stage where I can write some C code, and then boot it under [2]Bochs (an x86 emulator) and [3]VMWare. I thought it would be good to note down how to do this in case anyone out there is struggling to get this going.

The first thing to get a handle on is the boot loader. The wiki at [4]OsDev.org contains some good material on this, in particular this [5]article on the boot sequence which links to [6]here and [7]here. Of course to really see what's going on, it's good to have some working code, and after some searching I came across [8]MikeOS, a 16 bit operating system for x86. This is written entirely in x86 assembly, so you get to see the boot code itself (in sources\bootload\bootload.asm). This code uses calls into the BIOS to search the disk for the kernel executable, which it then loads and executes. The kernel then offers a basic command line interface, allowing you to run a number of modules that come with the system. These modules include a BASIC interpreter and other utilities. All the code is there, and is well commented so it is really easy to follow.

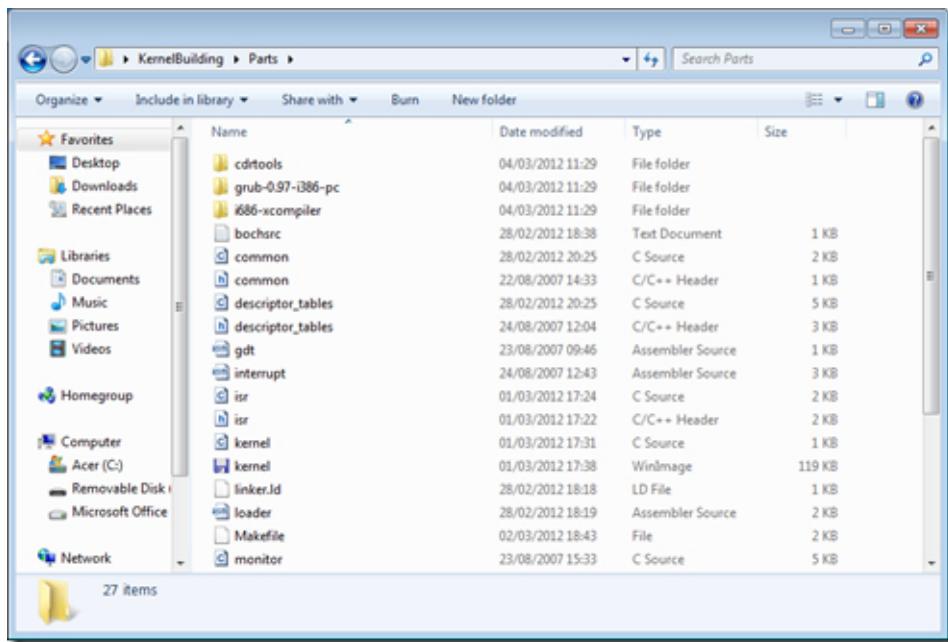
Building MikeOS is a little tricky - the lack of a loopback device on Windows requires the use of ImDisk (a virtual disk driver), but by following the instructions you can get something going which boots under VmWare fairly quickly.

I was more interested in running a 32 bit kernel and so started working through the tutorials by [9]James Molloy. These are brilliant. They use GRUB2 (the Grand Unified Boot Loader) to get the kernel loaded, and use a combination of a very small amount of assembler and C to get a kernel going on Windows. This tutorial kernel offers memory protection and multi-tasking though, unfortunately, the tutorial stops before getting to how we interact with external devices (such as disc drives) so I'm still investigating such material. It is a complete joy to be writing some C again - the feeling of being close to the machine level while still using a high level language is really liberating.

The tutorial is aimed at Linux and very probably works there, but I had a load of trouble getting it to run under Windows. The tools expect a compiler tool chain that generates ELF format object files, but the default Cygwin compilers on Windows generate PE format. I had to go out and find a set of precompiled cross compiler tools to build things.

I then had trouble actually writing an image in the correct format - the Linux examples all use a file mounted via a loopback device, and though there are such device drivers on windows I couldn't find one that successfully runs on 64bit machines. In the end I had to find another tutorial which uses GRUB, and modify that into the build scripts to get things to work.

I put a zip [10]here, KernelBuilding.zip, that contains a working examples from midway through the JamesM tutorial, at a point where we have just added the PIT to the kernel. The zip contains the necessary cross compilers, CD writing tools and the parts of GRUB that are required. You'll need to download [11]Cygwin, which simulates a Unix environment on Windows, [12]NASM, an assembler, and [13]Bochs, an x86 emulator which allows much quicker development.



We can then start a command shell, set up the path to Cygwin, and build the kernel.img.

[15]

```
C:\Windows\System32\cmd.exe
C:\Users\clive\Desktop\KernelBuilding\Parts>
C:\Users\clive\Desktop\KernelBuilding\Parts>set PATH=c:\cygwin\bin;%PATH%
C:\Users\clive\Desktop\KernelBuilding\Parts>make kernel.img
i686-xcompiler/bin/i686-elf-gcc -Werror -c -o kernel.o kernel.c
i686-xcompiler/bin/i686-elf-gcc -Werror -c -o descriptor_tables.o descriptor_
tables.c
i686-xcompiler/bin/i686-elf-gcc -Werror -c -o isr.o isr.c
i686-xcompiler/bin/i686-elf-gcc -Werror -c -o monitor.o monitor.c
nm -f elf -o gdt.o gdt.s
nm -f elf -o interrupt.o interrupt.s
i686-xcompiler/bin/i686-elf-gcc -Werror -c -o common.o common.c
i686-xcompiler/bin/i686-elf-ld -T linker.ld -o kernel.bin loader.o descriptor_ta_
bles.o kernel.o isr.o monitor.o gdt.o interrupt.o common.o timer.o
cat grub-0.97-i386-pc/boot/grub/stage1 grub-0.97-i386-pc/boot/grub/stage2 pad ke_
rnel.bin > kernel.img
C:\Users\clive\Desktop\KernelBuilding\Parts>
```

In order to run the GRUB boot loader, we are going to need to tell the loader the size of the kernel in 512 byte blocks (rounded up).

[16]

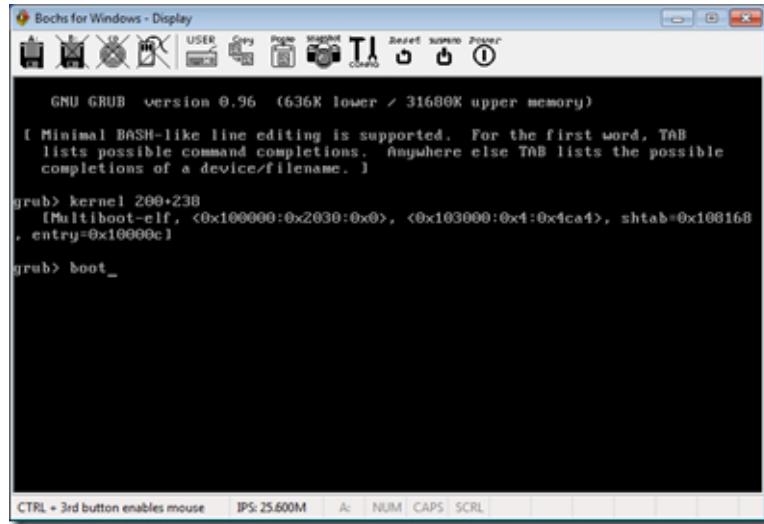
```
C:\Windows\System32\cmd.exe
C:\Users\clive\Desktop\KernelBuilding\Parts>ls -l kernel.img
-rwx-----+ 1 clive None 121672 Mar 4 11:32 kernel.img
C:\Users\clive\Desktop\KernelBuilding\Parts>set /a 121672/512
237
C:\Users\clive\Desktop\KernelBuilding\Parts>
```

Now we can run the emulation under Bochs using the command which I added to the Makefile:

make run

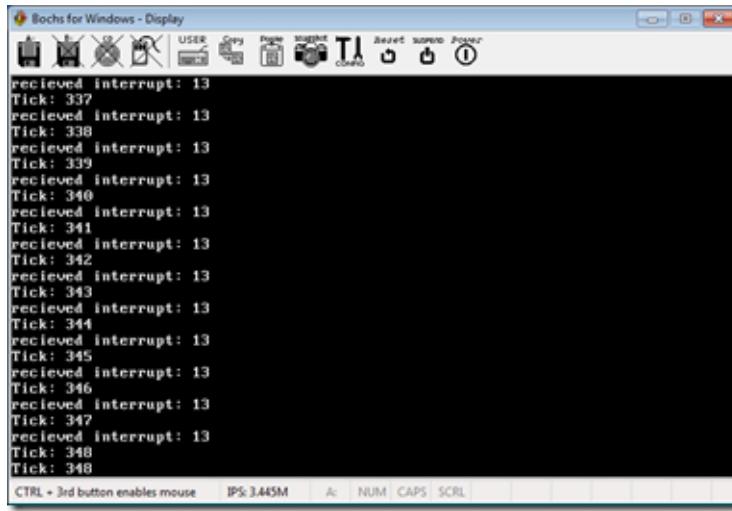
which starts the boot inside the Bochs x86 emulator.

[17]



The 237 value on the kernel line is the size of the kernel in blocks.

[18]



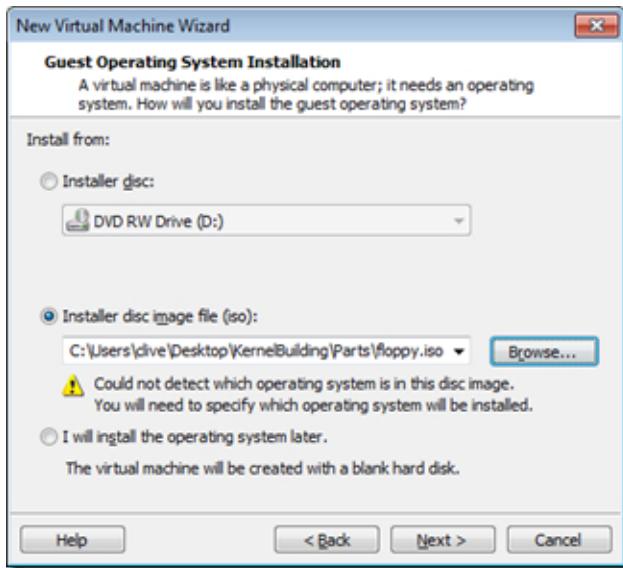
Running under an emulator is all very well, but I'd prefer to run under a real machine (or rather VMWare). To this we need to make an iso file, and for this reason I included the cdrtools in the zip file.

First we have to pad our kernel.img to the size of a standard floppy. I did this using the fsutil command in Windows (which you need to be admin to run). Using this we can generate a pad file and generate a floppy.iso file of the correct size.

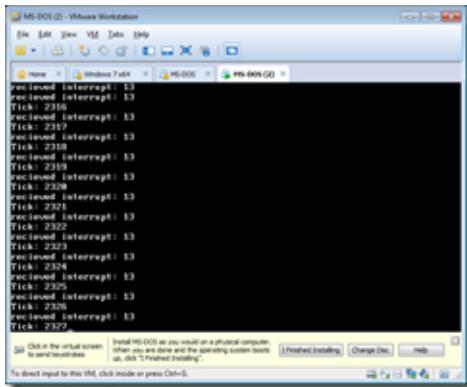
```
set /a (1474560-121672)
fsutil file createnew pad3 1352888
copy /b kernel.img+pad3 floppy.img
cdrtools\mkisofs.exe -o floppy.iso -b floppy.img floppy.img
```

This produces an iso we can run under VMWARE.

[19]



Running it gets us to the GRUB loader where we can then boot into our code.



[20]

Having got the setup out of the way, I now hope to spend more time working through the tutorials in order to get a better kernel running. It's pretty impressive that you can freely download all of the material that you need to do this, though the amount of different utilities means that it is some work to assemble a working set. Like all things, understanding how the kernel is working at the higher level helps to explain many of the higher level abstractions that you see, and is therefore a worthwhile learning exercise.

1. <http://www.azulsystems.com/products/zing/whatisit>
2. <http://bochs.sourceforge.net/getcurrent.html>
3. <http://www.vmware.com/>
4. [http://wiki.osdev.org/Main\\_Page](http://wiki.osdev.org/Main_Page)
5. [http://wiki.osdev.org/Boot\\_Sequence](http://wiki.osdev.org/Boot_Sequence)
6. <http://duartes.org/gustavo/blog/post/how-computers-boot-up>
7. <http://duartes.org/gustavo/blog/post/kernel-boot-process>
8. <http://mikeos.berlios.de/>
9. [http://www.jamesmolloy.co.uk/tutorial\\_html/1.-Environment%20setup.html](http://www.jamesmolloy.co.uk/tutorial_html/1.-Environment%20setup.html)
10. <https://skydrive.live.com/?cid=3F21DF299C355E7F&id=3F21DF299C355E7F%21129>
11. <http://www.cygwin.com/>
12. <http://www.nasm.us/pub/nasm/releasebuilds/>
13. <http://bochs.sourceforge.net/getcurrent.html>
14. <http://clivetong.files.wordpress.com/2012/03/distro.png>

15. <http://clivetong.files.wordpress.com/2012/03/compile.png>
  16. <http://clivetong.files.wordpress.com/2012/03/calculatesize.png>
  17. <http://clivetong.files.wordpress.com/2012/03/bochs.png>
  18. <http://clivetong.files.wordpress.com/2012/03/runningunderbochs.png>
  19. <http://clivetong.files.wordpress.com/2012/03/vmwareiso.png>
  20. <http://clivetong.files.wordpress.com/2012/03/vmware.png>
- 

### 6.3.4 Sorry, I gave you the wrong result last time (2012-03-25 15:24)

[1]Continuations almost seem magical when you meet them for the first time. We can take a standard program and transform it so that all of the function calls in it are implemented as tail calls. Magically the transformed program takes up very little stack space with all of the program state ending up stored in closures on the heap. Admittedly, this does make it harder to debug code as we lose the partial history that you get from a standard call stack view of an exception, but it also gets rid of all of the limitations that stack based implementations experience.

To me, the most mind blowing aspect of continuations is that they let you implement something like Scheme's call/cc, a construct that lets you implement a feature such as exception throwing as well as providing a means to implement cooperative multi-tasking within the language itself. Call/cc really is magical in that you can return from the dynamic function call multiple times. This can get really confusing and has to be used with care if your language has mutable state - the whole continuation idea has extensions such as delimited continuations which try to take away some of the power of the more general mechanism in order to take it more efficient to implement and to simplify the semantics.

The other day I came across a paper by [2]John Reynolds that describes the history of the discovery of continuations. I first came across continuations when studying functional languages, so it is a little ironic that they were invented as means of giving a semantics to gotos and labels.

Implementing such a construct on a standard architecture requires a number of tricks to get things working efficiently. Standard CPUs have a great deal of support for stack allocation and have instructions that expect stack frames of a particular format. We don't want to simply allocate everything on the heap and so need to choose carefully at which points we take snapshots of the stack that can be used if a captured continuation returns multiple times. A good number of possibilities are explored for ease of implementation and performance trade-offs in the following [3]paper by William Clinger

Equally interesting is that continuations are also connected with [4]Peirce's law in first order logic via the Curry-Howard isomorphism.

1. <http://en.wikipedia.org/wiki/Continuations>
2. <http://www.brics.dk/~hosc/local/LaSC-6-34-pp233-248.pdf>
3. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.70.9076>
4. [http://en.wikipedia.org/wiki/Peirce%27s\\_law](http://en.wikipedia.org/wiki/Peirce%27s_law)

---

### **6.3.5 I didn't mean to cause any disruption (2012-03-26 07:27)**

I've been following the [1]Mechanical Sympathy blog for some time. It hits the sweet spot of one of my interests in computer science – the use of virtual machines and their interaction with the hardware when running with multiple concurrent operations over shared resources. Locks are the traditional way to control concurrency, serializing the access to resources to make it easy to calculate the effects of multiple completing readers and writers, but their use may require entry to the kernel of the operating system with potential context switches and dirtying of the cache. These disadvantages can lead to a large loss in performance when compared against lower level but harder to use implementation mechanisms such as memory barriers and CAS (check and set) instructions that are built into most processor architectures.

The blog's author has spent a lot of time trying to get producers and consumers interacting at very high message rates as part of the implementation of a trading system. [2]Disruptor, his component programming framework, is the result of this work. There's a very good technical paper [3]here together with a [4]whole set of links to other blog posts and articles including this one [5]on the badness of locks and a [6]quick overview of memory barriers.

The real trick is to use a circular buffer for the buffering – this avoids the jitter of allocation of more dynamic schemes. Producers can use a CAS operation to reserve a slot within the buffer which they can fill at their leisure (within certain bounds), and they can then publish this change using another CAS. The consumers can check for the availability of new data using only a memory barrier rather than requiring the use of a potentially expensive lock.

Over the course of the work, the author and his collaborators have found various inefficiencies in the existing JVM implementations, and have managed to get fixes into particular high performance systems such as [7]Azul's Zing architecture.

The blog is good because the author goes out of his way to write micro-benchmarks that demonstrate the problems. For example, he has examples that show how adding padding can eliminate false sharing in the cache lines of one benchmark. The implementation is full of clever ideas and tricks and is well worth a read.

1. <http://mechanical-sympathy.blogspot.co.uk/>
  2. <http://code.google.com/p/disruptor/>
  3. <http://disruptor.googlecode.com/files/Disruptor-1.0.pdf>
  4. <http://code.google.com/p/disruptor/wiki/BlogsAndArticles>
  5. <http://mechanitis.blogspot.co.uk/2011/07/dissecting-disruptor-why-its-so-fast.html>
  6. <http://mechanitis.blogspot.co.uk/2011/08/dissecting-disruptor-why-its-so-fast.html>
  7. <http://www.azulsystems.com/>
- 

### **6.3.6 Money for nothing and your JITs for free (2012-03-27 07:32)**

Usually you write your VM, debug it, then write your JIT, and spend a lot of time debugging that. It's often really complicated to get the interpreted and jitted versions of the code to behave in

the same way. Laurence Tratt, who's been writing a VMs for a language he has developed, has managed to avoid having to do the second part of this workflow. In this [1]post, he documents a rather novel approach for writing a JIT for the VM that he has written – amazingly, he got the JIT without doing any work (or rather without writing code that actually writes any machine code instructions).

All he had to do was write the VM in a restricted subset of Python, called RPython, and then use the tool chain developed by the PyPy [2]project. I remember coming across this project many years ago – it is a project to write a version of Python inside Python. Writing a virtual machine in a high level language is good for all of the usual reasons that we use a high level language – productivity via good debuggability and conciseness of expression. Writing the VM in the language itself has been a technique for a long time. The original Smalltalk-80 was written in Smalltalk and the [3]Squeak Smalltalk implementation is bootstrapped by translating this VM into C which can then be compiled into a standalone virtual machine, while allowing the VM itself to be debugged using the standard Smalltalk tools. This idea has now been extended by the PyPy people to include [4]the generation of the tracing JIT. This meta-tracer traces through both the code of the interpreter and the code that the interpreter is running – this means that the interpreter needs to [5]pass extra information to the tracer so that it can determine when we are tracing the same code again, and it also means that the tracer doesn't actually trace the C code but through a byte code representation of what the C code was generated from. The process is, however, fairly automatic.

Tratt uses this technique to get a VM with good performance very quickly, and it's an impressive piece of work.

As a side note, tracing JITs seem to have recently fallen out of favour in some spheres. They rely on repeated paths of instructions and require that these paths don't get too long before a loop is detected. The Mozilla people have [6]gone back to a [7]more standard method based JIT.

1. [http://tratt.net/laurie/tech\\_articles/articles/fast\\_enough\\_vms\\_in\\_fast\\_enough\\_time](http://tratt.net/laurie/tech_articles/articles/fast_enough_vms_in_fast_enough_time)
  2. <http://pypy.org/>
  3. <http://squeak.org/>
  4. <http://codespeak.net/svn/pypy/extradoc/talk/icooolps2009/bolz-tracing-jit-final.pdf>
  5. <http://morepypy.blogspot.co.uk/2011/03/controlling-tracing-of-interpreter-with.html>
  6. <http://blog.mozilla.com/nethercote/2011/05/31/you-lose-more-when-slow-than-you-gain-when-fast/>
  7. <http://blog.mozilla.com/nethercote/2011/11/23/memshrink-progress-report-week-23/>
- 

## 6.4 April

### 6.4.1 Let's do this scientifically (2012-04-01 19:21)

For a long while, I've been meaning to spend some time investigating how much optimisation the .NET jit does to your program, and in particular how it trades off the debugging experience against the performance of the running application. I was recently given some time at work

to look into improving the debugging experience when people use the pdb files generated by [1]Reflector Pro.

There are various blog posts by Microsoft people spread across the internet that give snippets of information, but it was really nice to get a couple of days to run some experiments on a .NET application which was built in retail mode to see what you need to do to turn off enough optimisations to see the values of local variables, get a faithful stacktrace and be allowed to set the next statement as you can in a debug build with a debugger attached. What I found out I wrote up [2]here.

1. <http://www.reflector.net/vspro/>
  2. <http://www.reflector.net/2012/03/debugging-debugging-experience/>
- 

#### **6.4.2 You're so static (2012-04-02 19:22)**

I've spent a long time programming, for money, in static languages like C # and ML, and also a lot of time programming in dynamic languages like Common Lisp. Many times I've been involved in discussions about the trade offs between the two approaches. Dynamic languages make some things really easy - typically there's a REPL so you can quickly try out ideas and explore your way towards a solution to the problem you are working on. Static languages have their moments too - when I programmed ML, usually the program would work first time, though it often took a lot of debugging to get the code to type check in the first place. Type checking is basically a primitive form of theorem proving, and this goes to show that static typing is really about proving properties about your program. Some type systems like those of C # are too weak to ensure you won't get a runtime error, whereas the type system of ML and its emphasis on algebraic datatypes is often strong enough to avoid runtime errors.

[1]This blog post gives a really good contrast between the two systems, pointing out several of the common misconceptions about static and dynamic languages.

1. <http://cdsmith.wordpress.com/2011/01/09/an-old-article-i-wrote/>
- 

#### **6.4.3 Not your prototypical podcast (2012-04-03 19:23)**

I've recently come across this rather good podcast on JavaScript, [1]JavaScript Jabber. They have only done 9 episodes so far, but the [2]fifth episode on JavaScript Objects, and the [3]eighth with two members of the V8 (and Dart) team were every good. From the former I learned the [4]JavaScript with keyword and the latter contained a great discussion of Dart and why it exists. There's also some interesting biographical information on Lars Bak and Kasper Lund.

1. <http://javascriptjabber.com/>
  2. <http://javascriptjabber.com/005-jsj-javascript-objects/>
  3. <http://javascriptjabber.com/008-jsj-v8-and-dart-with-lars-bak-and-kaspar-lund/>
  4. <http://www.aptana.com/reference/html/api/JSKeywords.with%20statement.html>
- 

#### 6.4.4 A little Reflection can solve most problems (2012-04-04 19:25)

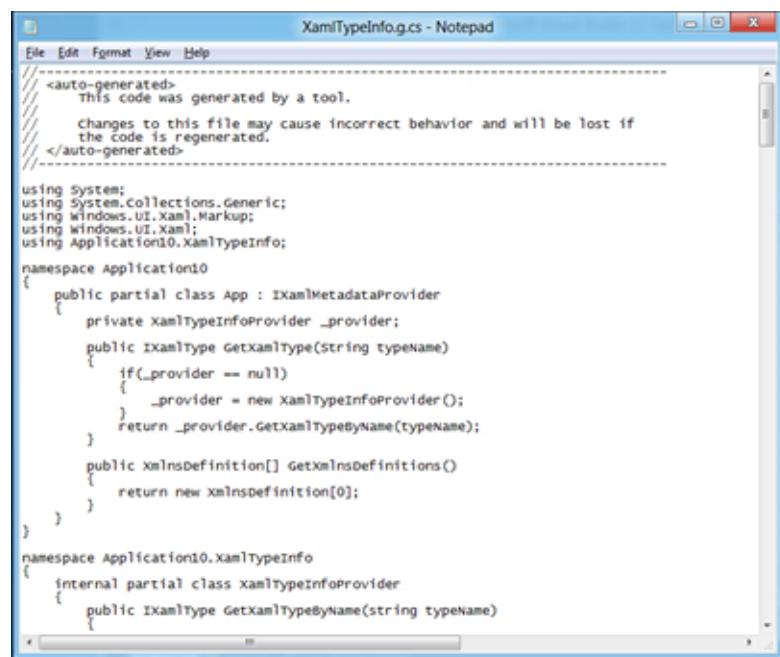
At the recent [1]DevWeek 2012 I attended quite a few of the sessions by [2]Dave Wheeler on WinRT. The resurrection of COM is really interesting, and Dave Wheeler is very probably right that the .NET framework libraries will now be entering maintenance mode and all future extensions will happen in the WinRT platform whose types can then be automatically projected into the .NET world.

XAML, which used to be a declarative system for writing a .NET object graph, and which was originally interpreted by managed code, is now interpreted by unmanaged code which produces a set of COM objects that implement the various elements of the GUI and application model.

The question that struck me while I was in one of the talks, was the observation that the old framework presumably used Reflection to find and instantiate the objects of the object graph, and to find out information about their properties for use in binding. COM objects, even with the extended form of `IUnknown`, don't offer this kind of Reflection. So how does this hang together?

If you take a new C# Windows Metro Style Application, and then build it, you'll notice that there is a new generated C# file called `XamlTypeInfo.g.cs` that contains extensions to some of your classes.

[3]



The screenshot shows a Windows Notepad window titled "XamlTypeInfo.g.cs - Notepad". The content of the file is as follows:

```
File Edit Format View Help
<auto-generated>
    This code was generated by a tool.
    Changes to this file may cause incorrect behavior and will be lost if
    the code is regenerated.
</auto-generated>

using System;
using System.Collections.Generic;
using Windows.UI.Xaml.Markup;
using Windows.UI.Xaml;
using Application10.XamlTypeInfo;

namespace Application10
{
    public partial class App : IXamlMetadataProvider
    {
        private XamlTypeInfoProvider _provider;
        public IXamlType GetXamlType(string typeName)
        {
            if(_provider == null)
            {
                _provider = new XamlTypeInfoProvider();
            }
            return _provider.GetXamlTypeByName(typeName);
        }

        public XmlNamespaceDefinition[] GetXmlNamespaceDefinitions()
        {
            return new XmlNamespaceDefinition[0];
        }
    }
}

namespace Application10.XamlTypeInfo
{
    internal partial class XamlTypeInfoProvider
    {
        public IXamlType GetXamlTypeByName(string typeName)
        {
```

It is this IXamlMetadataProvider interface, which is automatically implemented for your class, which allows WinRT to do the equivalent of Reflection. After Googling, I came across a really [4]good blog post which explains this more.

1. <http://www.devweek.com/>
  2. <http://coloringguy.com/>
  3. <http://clivetong.files.wordpress.com/2012/03/xamltypeinfo.png>
  4. <http://jaylee.org/post/2012/03/07/Xaml-integration-with-WinRT-and-the-IXamlMetadataProvider-interface.aspx>
- 

#### 6.4.5 In the place through which we wander (2012-04-23 07:00)

It's been hard to find any time for blogging lately. At work, I've been fully occupied getting [1]Reflector to handle the [2]new async feature in C # and VB, as well as doing loads of other Reflector enhancements. At home, I've been reading up on some mathematics, and preparing a lightening talk on Erlang for our bi-weekly lightning talk sessions. Any free time I've had, I've spent catching up with the [3]channel9 videos of the recent [4]Lang.NEXT and [5]Going Native conferences which Microsoft hosted this year.

The Lang.NEXT conference had plenty of material if you're interested in languages and virtual machines.

The first keynote by Martin Odersky called [6]Reflection and Compilers covers recent work in Scala on exposing the compiler's abstract syntax tree information via Reflection and links this with the addition of macros to the Scala language. The implementation of macros is rather clever, using a reify macro to move back from functions to their syntax trees, without having to add any special operators to the language. John Rose covered a nice example in his write up of the conference [7]here. Here's hoping that some of the ideas make it into C #.

Various Microsoft people covered the up and coming areas of Windows: [8]async, language support for asynchronous programming, the [9]Windows Runtime and [10]Roslyn, the managed compiler infrastructure that allows users access to semantic analysis features of the compiler. These are all interesting and are going to affect your day job if you work on the Window platform.

Two other interesting talks were one on [11]how they have successfully used Haskell at [12]Galois for the last ten years, and one on [13]the implementation of IKVM.NET, a .NET application which allow you to execute Java JAR files on the .NET platform and which also offers various libraries such as a better Reflection.Emit.

Channel9 recorded some interviews that happened alongside the main conference. One was [14]a discussion between Eric Meijer and Carl Hewitt on the [15]Actor model, in which Hewitt describes his [16]Actor model in great detail. There are lots of concurrency implementations in languages which are "based on the Actor model", and it was great to hear a description from one of the inventors which made it clear that none of these implementations are truly faithful. Hewitt raised the idea of [17]unbounded nondeterminism, an interesting idea that caused me to go off on a Google hunt where I found a good paper on the [18]history of process algebra as well as an [19]interesting wikipedia article.

[20]Going Native also covered some interesting material, with Stephan Lavavej doing a [21]presentation on the most recent STL and a talk on [22]extensions to templates in C++ 11. The panel discussions were also interesting, particularly the one about [23]the importance of being native.

1. <http://www.reflector.net/2012/04/net-framework-4-5-support-net-reflector/>
  2. <http://blogs.msdn.com/b/pfxteam/archive/2012/04/12/10293335.aspx>
  3. <http://channel9.msdn.com/>
  4. <http://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012>
  5. <http://channel9.msdn.com/Events/GoingNative/GoingNative-2012>
  6. <http://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012/Reflection-and-Compilers>
  7. [https://blogs.oracle.com/jrose/entry/monday\\_at\\_microsoft\\_lang\\_next](https://blogs.oracle.com/jrose/entry/monday_at_microsoft_lang_next)
  8. <http://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012/Language-Support-for-Asynchronous-Programming>
  9. <http://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012/The-Windows-Runtime>
  10. <http://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012/Roslyn>
  11. <http://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012/A-Means-to-Many-Ends-10-Years-of-Haskell-at-Galois>
  12. <http://corp.galois.com/>
  13. <http://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012/IKVM-NET-Building-a-Java-VM-on-the-NET-Framework>
  14. [http://channel9.msdn.com/\(A\(DIZWlv8LzQEkAAAAOTQONWI2ZTUtM2ZlYS00Yjg1LTg4NzMtNzJhZjA1MmUwZmMxAmqfHykWJRBKmZh75HL0--PjXeY1\)\)/Shows/Going+Deep/Hewitt-Meijer-and-Sz](http://channel9.msdn.com/(A(DIZWlv8LzQEkAAAAOTQONWI2ZTUtM2ZlYS00Yjg1LTg4NzMtNzJhZjA1MmUwZmMxAmqfHykWJRBKmZh75HL0--PjXeY1))/Shows/Going+Deep/Hewitt-Meijer-and-Sz)
  15. [http://en.wikipedia.org/wiki/Actor\\_model](http://en.wikipedia.org/wiki/Actor_model)
  16. [http://en.m.wikipedia.org/wiki/Actor\\_model\\_and\\_process\\_calculi\\_history](http://en.m.wikipedia.org/wiki/Actor_model_and_process_calculi_history)
  17. [http://en.wikipedia.org/wiki/Unbounded\\_nondeterminism](http://en.wikipedia.org/wiki/Unbounded_nondeterminism)
  18. <http://www.win.tue.nl/fm/0402history.pdf>
  19. [http://en.m.wikipedia.org/wiki/Actor\\_model\\_and\\_process\\_calculi\\_history](http://en.m.wikipedia.org/wiki/Actor_model_and_process_calculi_history)
  20. <http://channel9.msdn.com/Events/GoingNative/GoingNative-2012>
  21. <http://channel9.msdn.com/Events/GoingNative/GoingNative-2012/STL11-Magic-Secrets>
  22. <http://channel9.msdn.com/Events/GoingNative/GoingNative-2012/Variadic-Templates-are-Funadic>
  23. <http://channel9.msdn.com/Events/GoingNative/GoingNative-2012/Interactive-Panel-The-Importance-of-Being-Native>
- 

#### 6.4.6 There's nothing to see here (2012-04-25 07:00)

The release of Windows 8 is getting closer and closer, and there's a lot of new material to understand around this new platform. Here are a few articles that have recently caught my eye.

[1]An msdn article on App contracts and extensions – Windows 8 Metro style apps are fairly isolated from each other, and to get a unified experience the runtime manages the communication between applications via contracts.

Metro is a very grid based layout, and so HTML5 style applications are going to need to use CSS3 grid layouts to get their interfaces to fit in to the Metro style. [2]This article is a good introduction to the new layout.

So just how secure is the code of an application which is written for the WinRT platform. [3]This article covers reverse engineering of such applications.

There seems to be an association in the WinRT world between DirectX and native applications, but of course you can use libraries like SharpDx to do DirectX from managed code. [4]This article goes in to some of the details.

1. <http://msdn.microsoft.com/en-us/library/windows/apps/hh464906.aspx>
  2. <http://stephenwalther.com/blog/archive/2012/03/19/metro-introduction-to-css-3-grid-layout.aspx>
  3. <http://justinangel.net/ReverseEngineerWin8Apps>
  4. <http://advertboy.wordpress.com/2012/04/04/directx-in-your-winrt-xamlc-apps-sharpdx/>
- 

#### **6.4.7 Let's get it going (2012-04-27 07:00)**

I recently gave a lightning talk at work on the subject of Erlang. In five minutes of [1]slides and [2]code examples I tried to give a feel for several interesting features that the language and its runtime offers – lightweight processes using an Actor like model for communication, linking processes together to achieve fault tolerance and dynamic code updating.

1. <https://skydrive.live.com/#!/view.aspx?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%21478>
  2. [https://skydrive.live.com/embed?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%21477&authkey=AP34oyFIwQ\\_U5Ms](https://skydrive.live.com/embed?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%21477&authkey=AP34oyFIwQ_U5Ms)
- 

## **6.5 May**

### **6.5.1 Entomology for beginners (2012-05-04 10:42)**

[1]A Bug Hunter's Diary by Tobias Klein

This is a book that takes 7 examples of the author inspecting an application, finding an exploit and then exploiting it to take control of the machine. The examples cover a number of different machine architectures and operating systems. The appendices cover various technical items such as how to set up a windows kernel for debugging, notes on type conversions in C and an explanation of the ELF format global offset table.

Each exploit is covered in its own chapter. The author typically traces input from the user, and uses C reverse generated from x86 assembler to show how the target application doesn't correctly use this input data. By crafting a suitable set of data, the author shows how the instruction pointer (EIP) can be manipulated, and then explains how this could be used to take control of the machine. On Windows, one of the examples is the exploitation of a virus checker which installs a device driver, accessible by everyone, which has an IOCTL call which copies

data from a source location in user space to a target location that can be in either user or kernel space. By using a suitable crafted input request packet, this piece of code can be coerced into writing over its own code so that later execution of the virus checker will be running in kernel mode but will jump to an address that the writer has set up. Hence the user has elevated their privilege and has effectively got control of the machine.

The author makes the exploits look really easy- he explains that there are numerous fuzzing tools in existence that automatically alter the input to some piece of code as a means of finding out if the code can be coerced into altering something it shouldn't.

Moreover, every chapter has a fairly extensive set of references to interesting papers and web pages that explain some of the tricks in more detail. For example, there are references to a paper [2]explaining a Windows or Linux hack to enable a command shell and a very interesting article on [3]how to construct a Rootkit and how this uses DKOM to hide the fact that it is on the machine.

The appendices cover some of the more modern defences against the techniques that are outlined – address space randomisation, security cookies (to prevent stack buffer overflows, and using the CPU no-execute bit. A very good read!

1. [http://www.amazon.co.uk/Bug-Hunter%2527s-Diary-Software-Security/dp/1593273851/ref=sr\\_1\\_1?ie=UTF8&qid=1333222787&sr=8-1](http://www.amazon.co.uk/Bug-Hunter%2527s-Diary-Software-Security/dp/1593273851/ref=sr_1_1?ie=UTF8&qid=1333222787&sr=8-1)
  2. <http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-litchfield-paper.pdf>
  3. <http://c0decstuff.blogspot.co.uk/2011/01/ring-0f-fire-rootkits-and-dkom.html?m=1>
- 

### 6.5.2 Surely performant isn't a word (2012-05-08 06:47)

[1]High Performance JavaScript by Nicholas C Zakas

My loan of this from the library was almost finished before I'd had time to finish it – I had the choice of reading it in one day or not reading it at all. Thankfully I chose to read it in a couple of sittings, and I'm really glad I did.

The book consists of ten chapters on different aspects of writing high performance JavaScript enabled web sites and covers a range of issues around this topic. I'd heard bits and pieces of the advice from various sources such as blogs in the past, but it was nice to find the information all in one place.

Chapter one offers advice for getting the page to render quickly, predominantly by moving the script elements to the end of the page near the closing of the body element. Other techniques are also covered: deferred scripts in IE, and delayed loading of script using XHR or other methods and then loading the new code using a <script> element which is added to the DOM.

Chapter two drills into the implementation of JavaScript, in particular scope chains for the activation records and prototype inheritance for the objects, and uses this to explain why some coding patterns are slower than others. The book is really good in measuring the performance improvements across a number of browsers and displaying the speed difference using graphs to give the reader a clear feel for the impact of the suggestions.

Chapter three covers the implementation of the DOM, emphasising that the DOM and the JavaScript engine often live in different DLLs with a fairly slow communication path between the two parts. The chapter looks at the actions that cause repaints and reflows by the browser, and DOM queries that are blocked while this happens. This leads to a discussion of techniques for batching activities to avoid the engine having to wait for the DOM layout.

Chapters 4 and 5 look at slow language constructs including loop performance, if-then versus switch and lookup tables. There's a discussion of stack overflow and the stack sizes in the various browsers, and some techniques to avoid it happening. Writing efficient regular expressions is covered in some detail.

Chapter 6 looks at how the single-threadedness of a browser affects the perceived performance. The UI rendering and the execution share a single thread in most browser implementations. If the JavaScript blocks for more than 100 milliseconds, then this is going to cause the user to feel that the page is unresponsive. The chapter looks at how we can use `setTimeout` to schedule a long activity as a set of tasks that can be executed part by part whilst allowing the browser to catch up with rendering between the tasks. The behaviours of the different browsers is covered – information that I've not seen before. The chapter also contains details of the originally HTML5 web workers proposal which allows long running activities to happen on their own thread with a mechanism to communicate results between the main thread and the worker thread.

Chapter 7 looks at Ajax and the different in performance when using the data formats XML, JSON, JSON-P, HTML compared to custom data transmission formats. Response caching using HTTP headers is also covered.

Chapter 8 covers practices to get good performance including lazy loading and using native methods for good performance.

Chapters 9 and 10 round off the book by talking about the tooling for building and deploying the sites, and for analysing the performance of both the JavaScript (using a profiler) and the page download (by looking at the order of resource download and the techniques for compressing and merging the code files).

The book is fairly short but covers the material with a good amount of detail, making it both a quick and an informative read.

1. [http://www.amazon.co.uk/Performance-JavaScript-Faster-Application-Interfaces/dp/059680279X/ref=sr\\_1\\_1?ie=UTF8&qid=1336250207&sr=8-1](http://www.amazon.co.uk/Performance-JavaScript-Faster-Application-Interfaces/dp/059680279X/ref=sr_1_1?ie=UTF8&qid=1336250207&sr=8-1)

---

### 6.5.3 The sooner the better (2012-05-13 11:50)

[1] It's nice to see that Microsoft have made some additions to .NET 4.5 to try to use information recorded during one run of the application to improve the performance in subsequent runs. Multicore JIT is going to allow the JIT to be compiling methods on a background thread before the main thread of execution reaches them, hopefully improving the start up time. All you need to do is provide a directory where the CLR can store the information it accumulates, and name the profile when you tell the system to start profiling.

[2]

```
ProfileOptimization.SetProfileRoot("c:/temp/profile");
ProfileOptimization.StartProfile("profile1");
```

If the named file already exists, because of a previous recorded run, then the existing information is used to start background jitting of the recorded methods, and details from the latest run are used to overwrite the file.

There's also a way to [3]run your application through scenarios while collecting profile data which can be used by the NGEN compiler to optimise the final assembly - profile guided optimisation. This isn't as dynamic as the above approach, but the NGEN compiler is able to spend a lot more time optimising than the JIT compiler and hence use more traditional compiler optimisations.

It's going to interesting to try these features out on our .NET applications to see what performance improvements we can get.

Oh, and on a performance related theme, there's a good talk on the benefits of lock free programming [4]here on InfoQ, which has a pointer to the following paper on [5]memory ordering on Intel processors and details of the cache hit/miss information and cycles per instruction data that Intel processors make available by (unfortunately) Ring 0 instructions.

1. <http://msdn.microsoft.com/en-us/magazine/hh882452.aspx>
2. <http://clivetong.files.wordpress.com/2012/05/profile.png>
3. <http://blogs.msdn.com/b/dotnet/archive/2012/03/20/improving-launch-performance-for-your-desktop-applications.aspx>
4. <http://www.infoq.com/presentations/Lock-free-Algorithms>
5. <http://www.multicoreinfo.com/research/papers/2008/damp08-intel164.pdf>

---

#### 6.5.4 That's magic, that is (2012-05-13 16:53)

[1]Magical Mathematics: The Mathematical Ideas That Animate Great Magic Tricks by Persi Diaconis and Ron Graham

A great read which takes some principles from combinatorics and shows how they can be used to produce some fantastic magic tricks. This includes the [2]Gilbreath Principle which can be used to maintain some order in a deck when it is shuffled.

The book contains lots of interesting tricks, mathematical ideas, some juggling and lots of interesting stories behind some of the magicians of the day. I hadn't realised before that there are people who can do a perfect shuffle on demand - take a pack of cards and split it in half, and then riffle shuffle the two parts back together so that the final order has alternate cards from the left and right hands.

1. [http://www.amazon.co.uk/Magical-Mathematics-Mathematical-Animate-Tricks/dp/0691151644/ref=sr\\_1\\_1?ie=UTF8&qid=1336846634&sr=8-1](http://www.amazon.co.uk/Magical-Mathematics-Mathematical-Animate-Tricks/dp/0691151644/ref=sr_1_1?ie=UTF8&qid=1336846634&sr=8-1)
2. <http://headinside.blogspot.co.uk/2009/01/gilbreath-principle.html>

## 6.5.5 You're doing it upside down (2012-05-14 06:24)

[1]Programming Concurrency on the JVM by Venkat Subramamiam

I've always been fascinated by the connection between the hardware and the software, which manifests itself in a programming language when we start having to think about memory barriers and atomic check-and-set instructions which are often hidden behind the abstraction of locks, semaphores and monitors. It is an interesting area but I've always found this level of programming to be really hard and error prone.

I remember the horror of a colleague a few years back when he showed me some code that had one thread that made a call and then set a static field and another thread that waited for the thread to be set. Despite the first thread finishing, the second thread didn't notice the change. I pointed out that he needed to use the volatile keyword to stop the compiler allowing the second thread to cache the value that was read from the field. That such a simple program requires the field to be marked specially is a shock to many people. Locks have the same feel to them, and there's a [2]good paper by Edward Lee which argues that the non-determinism that the thread abstraction offers is completely non-intuitive, leaving us in a position where we are forced to hack away the non-determinism by using locks to get us to a position where we can start reasoning about the program again. Somehow this feels wrong – the usual mantra of development is “get it working, get it right, get it fast”, and we would often prefer to be in a position where we can reason about the program being right and then add extra code to get it fast rather than having to obscure the algorithm using tons of locking to get it right in the first place.

This book looks at concurrency from three perspectives: shared mutability (mutable fields and locks), isolated mutability (using actors) and pure immutability (using [3]STM). It looks at implementations of these three techniques running on the JVM, using the languages [4]Scala mainly for the [5]Akka framework which offers Actors and [6]STM, [7]Clojure which offers [8]STM and a version of Actors called Agents, and Java for the thread pool and Java 7 fork-join APIs.

Part one is a great introduction to concurrency. It first looks at a couple of the perils of concurrency and continues by looking at how we can introduce concurrency to speed up an I/O bound and a computationally intensive application. In these two cases, we need to break the work into independent tasks which then need to synchronise their results with each other. This breakdown introduces the ideas of state and scalability, allowing a discussion around ensuring visibility via the use of memory barriers and the preservation of invariants via atomicity using locks and synchronised.

Part two deals with [9]software transactional memory, first from the perspective of Clojure, a language in which all of the datatypes are immutably persistent and then from the point of view of Scala and the Akka library. In this second case, the programmer has to be a lot more careful to make sure that the objects managed by the transaction are immutable to avoid side-effects leaking through the transaction. STM looks like a great solution for the cases where there are many reads and fairly infrequent writes – too many writes and we may spend a lot of time retrying transactions, only to have them aborted again after they have burned lots of CPU cycles. STM is clever in that it leads to code that can be composed, without the programmer having to introduce any extra synchronisation.

Every time I see STM, I forget that we're not really talking about making it appear to the participants that they live in a world where all results can be serialized. By default the two implementations covered in the text implement [10]snapshot isolation, though both offer a means to work around problems that this might introduce by allowing reads to be included in the transaction log in such a way that the transaction will fail to commit if a value it read is modified

by a previously committed transaction – Clojure has the ensure function to do logging reads and Akka allows the transaction to be configured so that all reads are logged. You can see the Clojure implementation [11]here. It's not clear to me how many programs will be broken if programmers forget to upgrade some reads into effective writes.

Part three looks at [12]Actors. These are one technique for isolating mutability – changes to some state can only be made from a single thread which also serializes access to the state by demanding the requests to access the state are pushed into a mailbox which is processed by the controlling thread. The book looks at untyped Actors, in which the processing happens in an untyped receive method that the Actor implements, and typed Actors where the messages are strongly typed to correspond to methods defined on a particular interface. In a true Actor model, the messages need to be immutable, and in both Java and Scala it is the responsibility of the user to ensure that this is the case.

Actors help us to isolate the places that change the state, giving us thread safety without the need to introduce synchronisation. As the author points out, you can still get deadlocks happening, particularly in the Akka model which offers not only the traditional send-and-forget message send but also the send-and-wait-for-reply message, but at least Actors provide a nice boundary across which you access controlled state.

In order to get actors to cooperate, Akka has a notion of transactors – objects responsible for getting a set of actors to do a series of actions inside a transaction.

Part four summarizes what we've learned, that the less mutable state, the easier it is to manage the concurrency of an application. I think the book is a really good overview of concurrency and how there may be better ways for handling it than via the use of locks and atomic instructions. As a side effect this book made me keen to look more closely at Scala and the Akka framework.

1. [http://www.amazon.co.uk/Programming-Concurrency-JVM-Mastering-Synchronization/dp/193435676X/ref=sr\\_1\\_1?ie=UTF8&qid=1336820200&sr=8-1](http://www.amazon.co.uk/Programming-Concurrency-JVM-Mastering-Synchronization/dp/193435676X/ref=sr_1_1?ie=UTF8&qid=1336820200&sr=8-1)
  2. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.pdf>
  3. [http://en.wikipedia.org/wiki/Software\\_transactional\\_memory](http://en.wikipedia.org/wiki/Software_transactional_memory)
  4. <http://www.scala-lang.org/>
  5. <http://akka.io/>
  6. [http://en.wikipedia.org/wiki/Software\\_transactional\\_memory](http://en.wikipedia.org/wiki/Software_transactional_memory)
  7. <http://www.clojure.org/>
  8. [http://en.wikipedia.org/wiki/Software\\_transactional\\_memory](http://en.wikipedia.org/wiki/Software_transactional_memory)
  9. [http://en.wikipedia.org/wiki/Software\\_transactional\\_memory](http://en.wikipedia.org/wiki/Software_transactional_memory)
  10. [http://en.wikipedia.org/wiki/Snapshot\\_isolation](http://en.wikipedia.org/wiki/Snapshot_isolation)
  11. <https://github.com/clojure/clojure/blob/master/src/jvm/clojure/lang/LockingTransaction.java>
  12. [http://en.wikipedia.org/wiki/Actor\\_model](http://en.wikipedia.org/wiki/Actor_model)
- 

### 6.5.6 Thanks for the memory (2012-05-30 06:09)

At work I gave the following [1]lightning talk on software transactional memory. It's a lovely technology that is beautifully explained in the Simon Peyton-Jones' [2]paper. There's a [3]good retrospective from Joe Duffy on Microsoft's experiments in this area with an additional follow

up [4]here.

1. <https://skydrive.live.com/redir?resid=3F21DF299C355E7F!488>
2. <http://research.microsoft.com/en-us/um/people/simonpj/papers/stm/stm.pdf>
3. <http://www.bluebytesoftware.com/blog/2010/01/03/ABriefRetrospectiveOnTransactionalMemory.aspx>
4. <http://www.bluebytesoftware.com/blog/2010/05/17/MoreThoughtsOnTransactionalMemory.aspx>

---

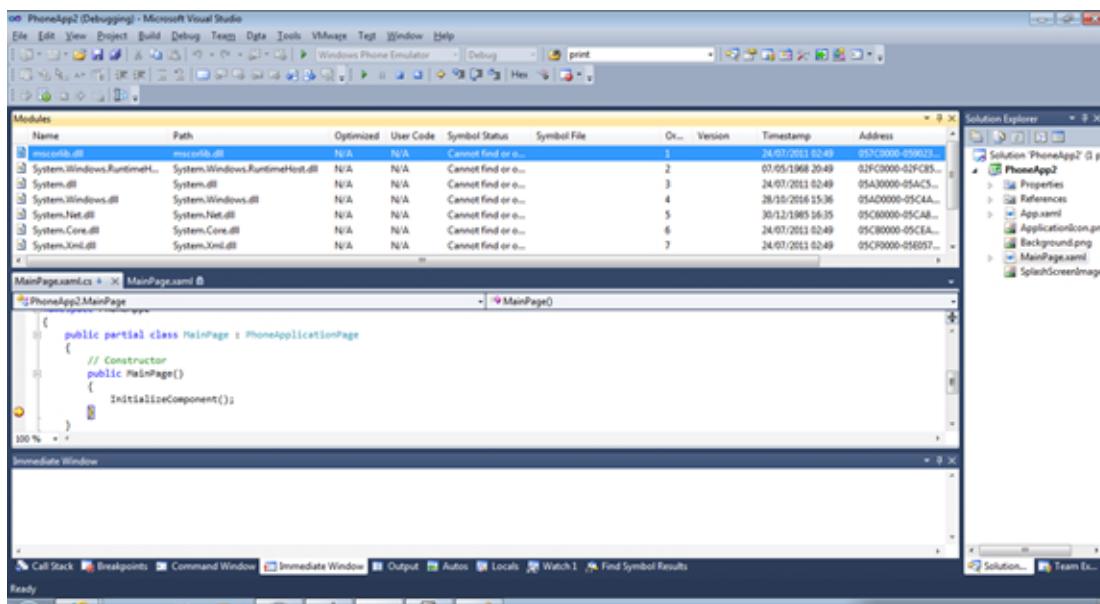
## 6.6 June

### 6.6.1 Phone me! (2012-06-14 05:00)

I have to admit that it's been bugging me for a while. I'd like to have a look at the code of the assemblies that are used by Windows Phone in order to get a feel for how the managed code interacts with the phone hardware. The trouble is that the true assemblies are only held within the emulator that ships with Visual Studio. As part of the install on the desktop, all you get is reference assemblies which contain the metadata but not the IL. I couldn't find a way to get access to the file system of the emulator to get hold of the assemblies.

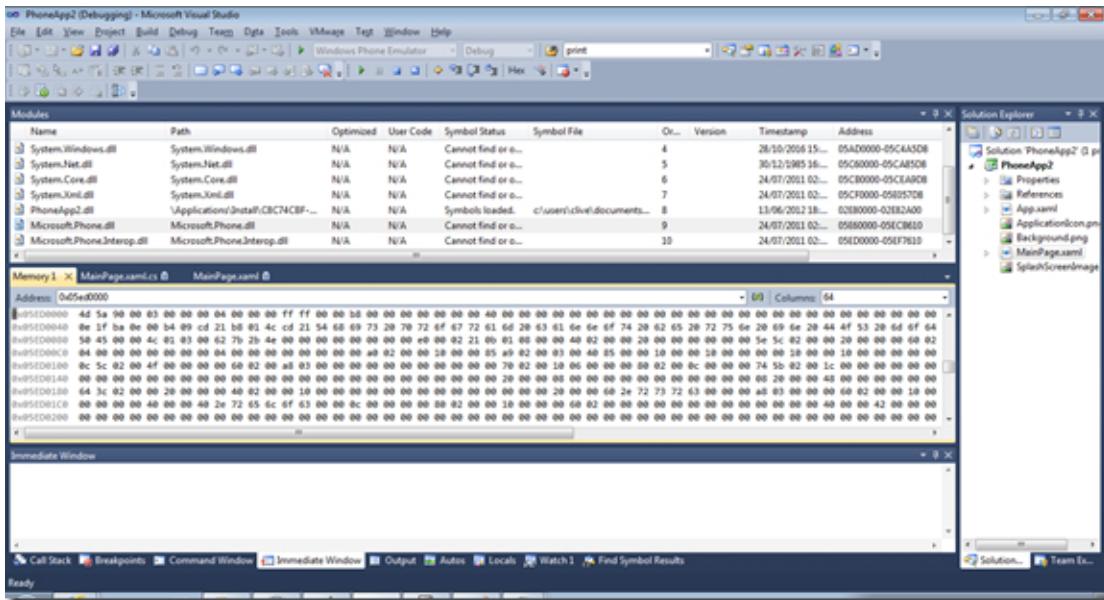
When you write a Phone application and debug it within the emulator, you can see that a number of assemblies are loaded, but I couldn't think of an easy way to get to them.

[1]



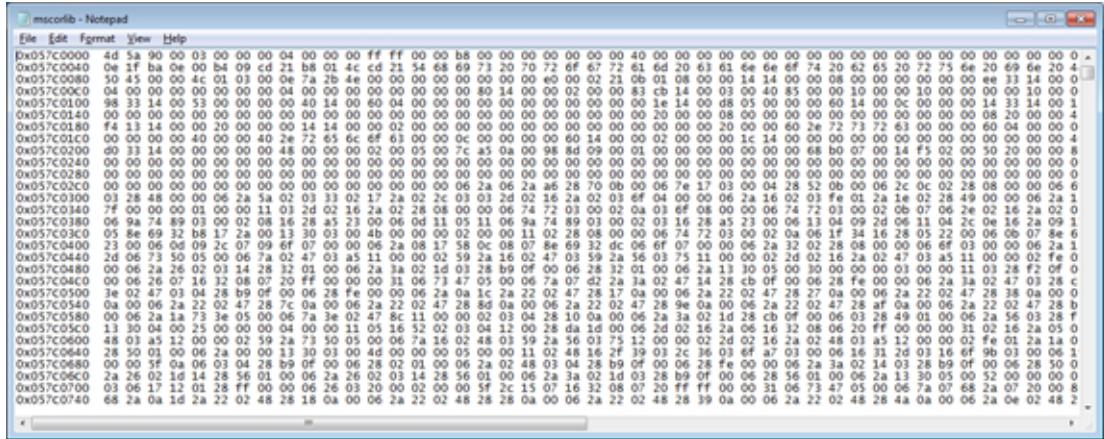
The other day I remembered two things. First, the CLR memory maps assemblies into memory as a means of quickly accessing their bytes, and secondly, the Modules window in the debugger tells you where an assembly is loaded, giving both the start and end address.

[2]



Most importantly, we can get to the bytes in a process using the Memory window from inside Visual Studio. Hence we can fairly easily cut and paste the textual representation of a section of memory into a text file – in fact you could easily write a macro to do this. For example, using the address of mscorlib.dll from the Modules window, we can generate a text file containing all of the bytes that were mapped into the emulator process for mscorlib.

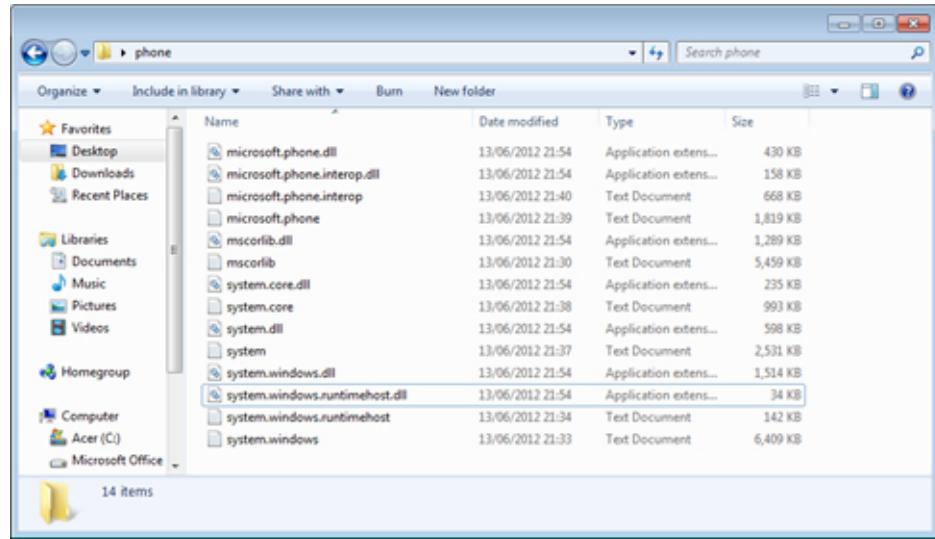
[3]



It's then easy to write 15 lines or so of C# which read this text file and convert it back into a dll file.

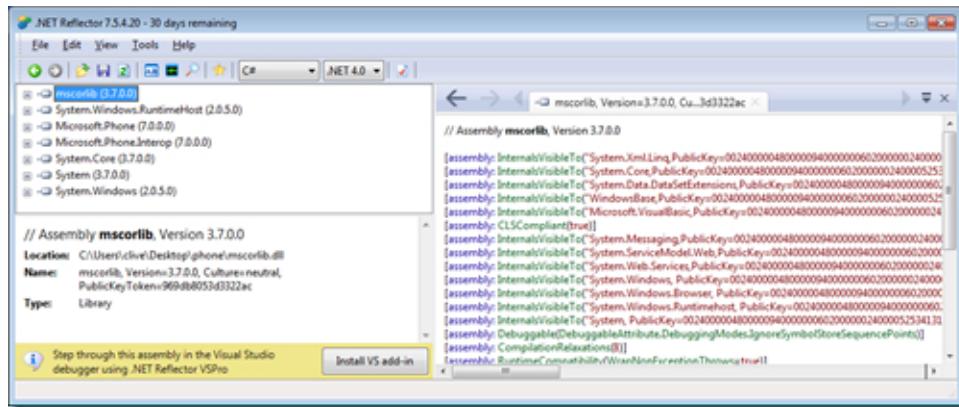
[4]

644



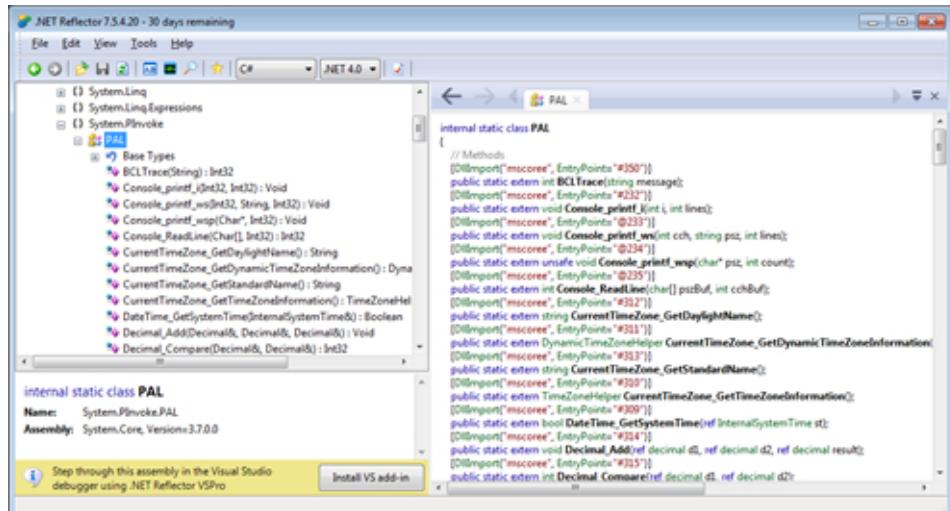
Having done this, we can load the assembly into a tool such as Reflector, and start looking at the decompiled code.

[5]



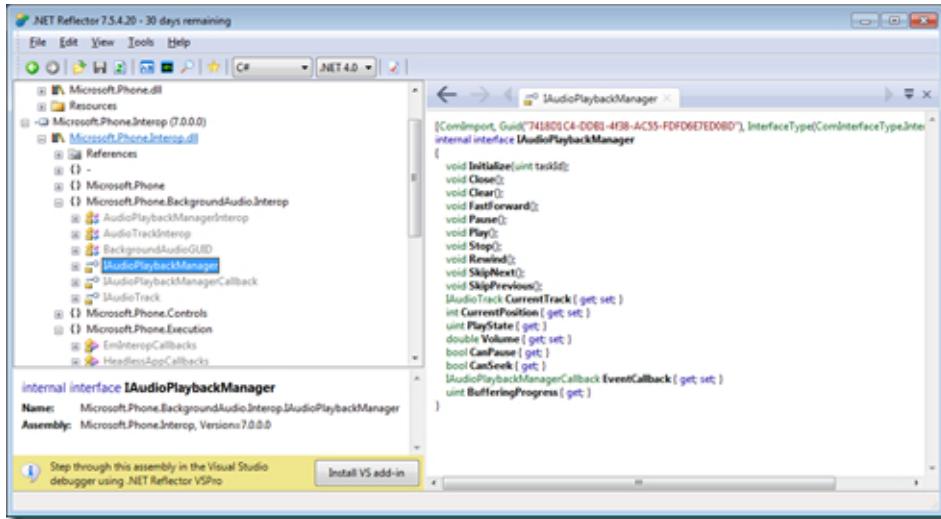
For example, we can see where the managed code calls into unmanaged functions that are exposed from mscoree.dll

[6]



Or take a look into the Interop dll to see the managed calls via COM into the unmanaged routines that handle audio playback.

[7]



Using a tool like [8]Reflector Pro, one can even decompile and generate a pdb from these assemblies, and hence set breakpoints in the managed code that is running in the emulator. All of which I think is rather cool.

1. <http://clivetong.files.wordpress.com/2012/06/vs1.png>
2. <http://clivetong.files.wordpress.com/2012/06/vs2.png>
3. <http://clivetong.files.wordpress.com/2012/06/mscorlib.png>
4. <http://clivetong.files.wordpress.com/2012/06/folder.png>
5. <http://clivetong.files.wordpress.com/2012/06/reflector1.png>
6. <http://clivetong.files.wordpress.com/2012/06/pal.png>
7. <http://clivetong.files.wordpress.com/2012/06/interop.png>
8. <http://www.reflector.net/>

## 6.6.2 Two for the price of one (2012-06-15 05:48)

I read two theory related blog posts at the weekend which I thought were very interesting.

The first was by Laurence Tratt on the [1]subject of parsing, and suggests that we are getting to the stage where we'd like to embed DSLs into standard programming languages, raising the question of how we compose the two grammars. The post raises a lot of interesting points, and also links to a load of [2]interesting papers on parsing PEGs and [3]boolean grammars. PEGs look really interesting to me and demonstrate the advantages of functional thinking – if we remove the ambiguity of a grammar by ordering the rules for a non-terminal symbol, then we can regard the parser as a pure function with inputs the current character positions and the non-terminal that we expect to start at this position and which outputs a parse tree representation. We can therefore apply memoisation, a standard functional programming trick, to these parsing functions to get a parser that works in linear time.

The second was by Chris Smith on the subject of [4]“Why Do Monads Matter?”. This essay compares and contrasts the views of mathematicians and computer sciences in the way that they look at monads. He looks at the problems represented by the four horsemen of the cataclysm and considers how these problems relate to categories. In so doing, he gives a very good motivation for the inclusion of monad-based datatypes in languages such as Haskell.

1. [http://tratt.net/laurie/tech\\_articles/articles/parsing\\_the\\_solved\\_problem\\_that\\_isnt](http://tratt.net/laurie/tech_articles/articles/parsing_the_solved_problem_that_isnt)
  2. <http://bford.info/packrat/>
  3. <http://users.utu.fi/aleokh/boolean/>
  4. <http://cdsmith.wordpress.com/2012/04/18/why-do-monads-matter/>
- 

### **6.6.3 All together now (2012-06-16 05:46)**

[1]Stephen Toub was on a fairly [2]recent episode of .NET Rocks where he makes some good observations on asynchrony, concurrency and parallelism. He's also got some really good posts on the [3]pfxteam blog, in particular [4]a recent post on [5]synchronization and [6]execution contexts which provide a means for taking thread local information from one thread to another.

1. <http://blogs.msdn.com/b/toub/>
  2. <http://dotnetrocks.com/default.aspx?showNum=775>
  3. <http://blogs.msdn.com/b/pfxteam/>
  4. <http://blogs.msdn.com/b/pfxteam/archive/2012/06/15/executioncontext-vs-synchronizationcontext.aspx>
  5. <http://msdn.microsoft.com/en-us/library/system.threading.synchronizationcontext.aspx>
  6. <http://msdn.microsoft.com/en-us/library/system.threading.executioncontext.aspx>
- 

### **6.6.4 Please be discrete (2012-06-17 05:04)**

[1]Invitation to Discrete Mathematics by Jiri Matousek and Jaroslav Nesetril

A lovely book that I borrowed from the library and spent a few weeks reading. Discrete mathematics covers a whole mass of interesting areas for someone who spends a lot of time with computers. There are chapters on functions and relations, orderings, combinatorial counting, graphs and trees including drawing graphs in the plane, double-counting (the inclusion-exclusion principle) and the count of spanning trees. After this there are chapters on finite projective planes, probability and probabilistic proofs, Ramsey's theorem and generating functions.

The material is explained clearly with lots of great examples and plenty of problem sets that help with understanding the material, ranging in difficulty from straightforward to fairly difficult. There were several interesting proofs that I hadn't seen before. These included

the Erdos-Szekeres lemma, “an arbitrary sequence ( $x_1, x_2, \dots, x_{n^2+1}$ ) of real numbers contains a monotone subsequence of length  $n+1$ ”, which was proved using a result on finite orderings, and Sperner’s lemma on the labelling of the vertices of a triangle which has been dissected into smaller triangles which was proved using the handshaking lemma. The chapter on generating functions was also very good, and the whole book was a really good read.

1. [http://www.amazon.co.uk/Invitation-Discrete-Mathematics-Jir%C3%AD-Matousek/dp/0198570422/ref=sr\\_1\\_sc\\_1?ie=UTF8&qid=1339786172&sr=8-1-spell](http://www.amazon.co.uk/Invitation-Discrete-Mathematics-Jir%C3%AD-Matousek/dp/0198570422/ref=sr_1_sc_1?ie=UTF8&qid=1339786172&sr=8-1-spell)

---

### 6.6.5 Sweet seventeen (2012-06-18 05:08)

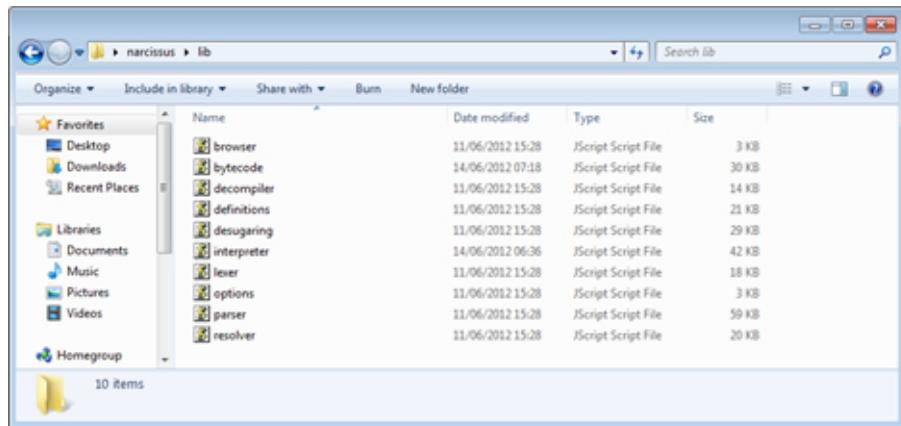
I really enjoyed the [1]recent talk by Brendan Eich at the fluent conference. Javascript is 17 years old, and Eich mentions some of the recent developments, including [2]Emscripten which has been used to [3]port a number of [4]C++ games to Javascript/HTML5, and [5]LLjs, an implementation of Javascript that lets you work at the byte level.

For me though, the highlight was the mention of [6]Narcissus, a meta-circular Javascript interpreter. Meta-circular interpreters started with the definition of Lisp in the incarnation of Lisp 1.5, where the Lisp interpreter (and hence the whole system) was defined in Lisp, the idea being that if you had a vague understanding of Lisp, you could use this knowledge to further your understanding by looking at the implementation. That’s exactly the position I feel I’m in with Javascript.

You can download Narcissus [7]from here though you’ll need the [8]SpiderMonkey shell to run it. I initially started with NodeJs but the interpreter uses some of the new JavaScript meta-programming APIs (Proxy) which the release version of NodeJs doesn’t support.

The Narcissus code is fairly small, but contains a lexer/parser, bytecode compiler and decompiler, and interpreter so it a great workbench for understanding the semantics of Javascript.

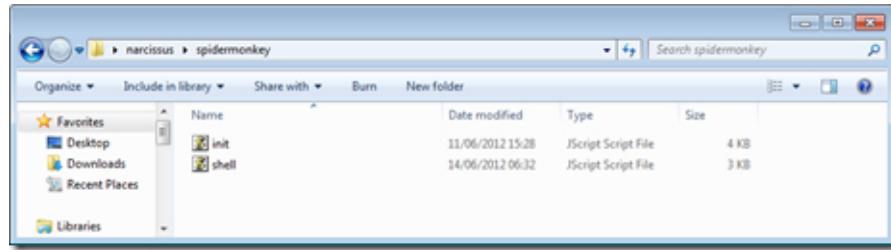
[9]



The download contains a SpiderMonkey boot script

[10]

648



It is really easy to start up and use the parser or start a REPL for the Narcissus Javascript engine.

[11]

```
C:\Windows\System32\cmd.exe -_js.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\clive\Desktop\narcissus\spidermonkey..\js.exe
js> load("shell.js");
js>
js> Narcissus.parser.parse("function foo() { return 10; }");
<type:SCRIPT, lineno:1, filename:"", children:[{type:FUNCTION, value:"function", lineno:1, start:2, end:29, filename:"", children:[], params:[], paramComments:[], blockComments:[], name:"foo", body:{type:SCRIPT, value:""}, lineno:1, start:15, end:23, filename:""}, {type:RETURN, value:{type:NUMBER, value:10, lineno:1, start:24, end:26, filename:"", children:[], generatingSource:true}, lineno:1, start:17, end:23, filename:"", children:[], generatingSource:true}, {type:FUNCTION, value:"function", lineno:1, start:24, end:26, filename:"", children:[], params:[], paramComments:[], blockComments:[], name:"foo", body:{type:SCRIPT, value:""}, lineno:1, start:25, end:27, filename:"", children:[], generatingSource:true}, {type:RETURN, value:{type:NUMBER, value:10, lineno:1, start:28, end:30, filename:"", children:[], generatingSource:true}, lineno:1, start:29, end:31, filename:"", children:[], generatingSource:true}], modDecl:[], funDecl:[], varDecl:[], modDefn:[], modLoad:[], impDecl:[], expDecl:[], exports:[], hasEmptyReturn:false, hasReturnValue:true, hasYield:false, generatingSource:true}, functionForm:0, generatingSource:true}, funDecl:[{type:FUNCTION, value:"function", lineno:1, start:2, end:29, filename:"", children:[], params:[], paramComments:[], blockComments:[], name:"foo", body:{type:SCRIPT, value:""}, lineno:1, start:15, end:23, filename:""}, {type:RETURN, value:{type:NUMBER, value:10, lineno:1, start:24, end:26, filename:"", children:[], generatingSource:true}, lineno:1, start:17, end:23, filename:"", children:[], generatingSource:true}, {type:FUNCTION, value:"function", lineno:1, start:24, end:26, filename:"", children:[], params:[], paramComments:[], blockComments:[], name:"foo", body:{type:SCRIPT, value:""}, lineno:1, start:25, end:27, filename:"", children:[], generatingSource:true}, {type:RETURN, value:{type:NUMBER, value:10, lineno:1, start:28, end:30, filename:"", children:[], generatingSource:true}, lineno:1, start:29, end:31, filename:"", children:[], generatingSource:true}], modDecl:[], funDecl:[], varDecl:[], modDefn:[], modLoad:[], impDecl:[], expDecl:[], exports:[], hasEmptyReturn:false, hasReturnValue:true, hasYield:false, generatingSource:true}, hasEmptyReturn:false, hasReturnValue:true, hasYield:false, generatingSource:true}
js> Narcissus.interpreter.repl();
njs>
njs> function foo() { return 20; }
njs> foo();
20
njs>
```

It's easy to call functions in the Narcissus environment

[12]

```
C:\Windows\System32\cmd.exe -_js.exe
C:\Users\clive\Desktop\narcissus\spidermonkey>
C:\Users\clive\Desktop\narcissus\spidermonkey..\js.exe
js> load("shell.js");
js> Narcissus.interpreter.repl();
njs> function bar(x) { return x * 8; }
njs> .exit
njs> Narcissus.interpreter.evaluate("bar (9);");
72
njs>
```

And experiment with the bytecode that the compiler produces

[13]

```

C:\Windows\System32\cmd.exe - ..\js.exe
njs> print(dis <function foo(x) < if (x == 2) return 40; return 20; >>>
Wrapping print
  0: GETGLOBAL name: "x"
  1: CONST value: 2
  2: BINOP operator: 13 name: "=="
  3: JUMPF label: 7
  4: CONST value: 40
  5: MODE unwind: [] goal: -1
  6: UNWIND
  7: NOP
  8: CONST value: 20
  9: MODE unwind: [] goal: -1
 10: UNWIND
 11: CONST value: <void 0>
 12: MODE unwind: [] goal: -1
 13: UNWIND
njs> 

```

The actual implementation of Narcissus is clever. It uses [14]Proxies to wrap existing objects to allow the Narcissus interpreter to intercept access to them. This is used to allow you to access functions from the normal environment in which Narcissus is running ie the “wrapping print” text above is generated by a print statement that I added into the code of Narcissus which wraps the underlying function when it is exposed to the Narcissus code.

[15]

```

function wrapNative(name, val) {
  print("wrapping " + name);
  if (!definitions.isNativeCode(val))
    return val;
  return Proxy.createFunction(
    definitions.makePassthruHandler(val),
    function() { return val.apply(hostGlobal, arguments); },
    function() { return applyNew(val, arguments); });
}

```

Narcissus is a good platform for experimenting with extensions to Javascript - you have a full parser so you can add new constructs to the grammar, add support to the compiler and associated bytecode and then test it from the interpreter. I was more interested in using it to get a better understanding of the semantics of Javascript and it works really well for that purpose.

On a slightly related note, there is also a discussion about the Chakra engine in the post [16]Advances in Javascript performance in IE10.

1. <http://www.youtube.com/watch?v=Rj49rmc01Hs>
2. <http://hacks.mozilla.org/2012/04/porting-me-my-shadow-to-the-web-c-to-javascriptcanvas-via-emscripten/>
3. <http://www.webpronews.com/easily-port-c-to-html5javascript-with-emscripten-2012-04>
4. <http://hacks.mozilla.org/2012/04/porting-me-my-shadow-to-the-web-c-to-javascriptcanvas-via-emscripten/>
5. <http://lljs.org/>
6. <https://github.com.mozilla/narcissus/>
7. <https://github.com.mozilla/narcissus/>
8. [https://developer.mozilla.org/En/SpiderMonkey/Introduction\\_to\\_the\\_JavaScript\\_shell](https://developer.mozilla.org/En/SpiderMonkey/Introduction_to_the_JavaScript_shell)
9. <http://clivetong.files.wordpress.com/2012/06/lib.png>
10. <http://clivetong.files.wordpress.com/2012/06/spidermonkey.png>
11. <http://clivetong.files.wordpress.com/2012/06/simpleuse.png>
12. <http://clivetong.files.wordpress.com/2012/06/evaluate.png>
13. <http://clivetong.files.wordpress.com/2012/06/dis.png>
14. <http://wiki.ecmascript.org/doku.php?id=harmony:proxies>
15. <http://clivetong.files.wordpress.com/2012/06/wrap.png>
16. <http://blogs.msdn.com/b/ie/archive/2012/06/13/advances-in-javascript-performance-in-ie10-and-windows-8.aspx>

---

### **6.6.6 There can only be one winner (2012-06-20 05:09)**

[1]The Triumph of Types by Robert Constable

This is a really interesting paper giving an overview of the history of types in mathematics and logic, and how this led to constructive type theory. It's a quick but informative read after which I went and read two other papers - [2]Naive Computational Type Theory and [3]Programming in Martin-Löfs type theory.

1. <http://www.cs.uoregon.edu/Research/summerschool/summer11/lectures/Triumph-of-Types-Extended.pdf>
  2. <http://www.nuprl.org/documents/constable/naive.pdf>
  3. <http://www.cse.chalmers.se/research/group/logic/book/book.pdf>
- 

### **6.6.7 Back you go! (2012-06-22 05:11)**

[1]Reversing: Secrets of Reverse Engineering by Eldad Eilam

We've all done a bit of reversing in the past, whether in the debugger when we look at the x86 machine code and try to understand what we're seeing as a function, or when we take a DLL and use dumpbin to look at its imports and exports in order to understand the APIs that it is using to do its work. I felt it was time to have a look at a book that explained the general techniques while at the same time offering a discussion of the legality of what you are doing.

This is a really good book that covers all of this and more. It's very practical containing lots of x86 machine code, so it would be useful to have experience with this before reading it though the book has a fairly good but concise introduction to the x86 architecture so it is still self-contained.

The first part has an overview of reverse engineering, explaining how it is legal for interoperability reasons, and covers Windows fundamentals such as threads and synchronisation objects. There's also an overview of useful reversing tools including debuggers such as IDA Pro, patching tools such as Hex Workshop (though I'd use EMACS for this) and standard Windows tools like DUMPBIN and PEVIEW.

Part two gets into the worked examples that cover various aspects of reversing, all of which have the intention of getting C/C++ code out of x86 assembler. The first example looks at generating code for an undocumented Windows API, the generic table implemented in NTDLL.DL. Along the way we get to see some of the optimisations that the C compiler uses and how this complicates the x86. The next example is on how to decode a file format for a cryptography application that the author has written, and this is followed by a chapter on auditing program binaries for vulnerabilities like stack and heap overflows and double decoding problems in web servers. The last example takes a real piece of malware and looks at how you reverse it in order to understand what it is doing.

Part three, Cracking, has chapters on the general methods for copy protection, ideas for preventing reversing of your application which includes methods for confusing disassemblers and anti-debugger techniques (such as using SEH to determine if a debugger is attached), and an example of breaking the protections on an example application.

Part four looks at reversing .NET and decompilation in general. It looks at some .NET obfuscators and how they do their work.

The book also has a number of appendices that cover x86 calling conventions and the compilation of the various structured programming constructs, how arithmetic is compiled to x86 plus lots of other miscellaneous stuff.

If you like low level programming this book has lots to offer and though a few years out of date, it was published in 2005, it contains tons of relevant material and is a good read.

1. [http://www.amazon.co.uk/Reversing-Secrets-Engineering-Eldad-Eilam/dp/0764574817/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1339786840&sr=1-1](http://www.amazon.co.uk/Reversing-Secrets-Engineering-Eldad-Eilam/dp/0764574817/ref=sr_1_1?s=books&ie=UTF8&qid=1339786840&sr=1-1)
- 

## 6.7 July

### 6.7.1 Are you still there? (2012-07-30 05:24)

It's been a busy time at work lately, what with working on the next version of Reflector, and the requirement of getting our summer intern up to speed. Also it's holiday season, and there's also the question of the [1]latest IBM Ponder which I spent a lot of time thinking about - the answer for the latter is contained in [2]this MathOverflow post. This post is a summary of the useful articles and posts that I have recently found to be useful.

I've read previous editions of Windows Internals, and the [3]recently released sixth edition keeps the standard very high. It covers the material of older editions, but has now been split into two books, only the first of which is currently available. I'd wondered recently about the implementation of virtualisation using a Hypervisor and how 32 bit applications run successfully on a 64 bit machine - this is covered in some detail in the book and makes a very interesting read (and also set the scene for understanding the [4]recent post about a bug in this area). I read the book from cover to cover, which is a good way to get an idea of the topics that you'll want to come back to at a later stage.

During a holiday, I think it is good to read a couple of technical books (as well as a couple of non-technical books), so this year I took a couple of books and papers on my iPad.

[5]Programming Windows Sixth Edition which is a version of the classic text aimed at the XAML stack that runs on WinRT - as usual it's a clear explanation of the windows side of applications for this platform. Petzold takes his time explaining the mapping between the XAML and underlying object instantiations in the underlying platform. It includes some good examples and some good explanations.

[6]A Tutorial on Co-induction and Function Programming by Andrew Gordon is a great explanation of the foundation of co-induction and explains how co-induction can make it easy to

prove properties of lazy functional programs. Gordon explains a formulation of bisimulation as a means of proving properties of functional programs starting from an operational semantics. It's a fairly concise paper with some maths that takes a while to get to grips with.

[7]Isolated Actors for Race Free Concurrent Programming is a Phd thesis which covers the design of the Scala actors library. It's interesting that Scala offers enough language hooks to make it possible to implement Actors as a library; the library offers both the threading and reactive version of the Actors. In the former, the Actor potentially consumes a thread pool thread while it is blocking whereas in the latter, the continuation is stored into a task which can be started by the library when the continuation can continue. The only real weakness of a library approach is that an exception is needed to exit the local computation and return to the main processing loop – this exception could be seen (and potentially caught) by user code, breaking the invariants of the library implementation.

The thesis covers the threading and eventing models of distributed computation and explores their trade offs. It also covers the implementation of a Join calculus library on top of the Actor model. This highlights the hooks that Scala has for implementing such libraries – in particular the way that function application is translated into a method call of an apply method on a function object which is extended to partial functions (which support the isDefinedAt which can be called before the apply is launched), and the extension of pattern matching to use an unapply method to convert what looks like a simple pattern match into active code execution.

The last chapter of the thesis covers a new type based approach for ensuring Actor isolation. I've also spent some time watching the following videos.

[8]Bart De Smet covers Rx 2.0 in a Channel9 Deep Dive. Rx is a fascinating technology and seems to be getting more and more hardened for industrial use as the releases go by. 2.0 has a number of performance improvements and bug fixes. This is the first video that I've seen which goes into detail about the abstraction over time which underlies the library.

[9].NET 4.5 in Practice: Bing is a Channel9 interview with a couple of members of the .NET team and a developer on the Bing platform. It covers some of the improvements that have been made to the .NET 4.5 platform, and explains how some of the optimisations were used to improve the performance of the Bing web site.

[10]Deep dive into the kernel of the .NET framework, a video from last year's build, covers the profiler guided optimisation and background thread jitting that was included in the latest version of the .NET framework. [11]There's also a recent Channel9 Live on the subject of Roslyn.

[12]Building HTML and Javascript apps with KnockoutJS and MVVM was also good viewing, giving a good overview of KnockoutJS and showing how it enables one to write a view model and bind it to the GUI with simple declarative syntax.

There's the [13]best explanation of WinRT and how it fits in with .NET on a recent HanselMinutes. The interview covers projections and the adapters that implement them, and makes it clear that the WinRT API surface is designed to surface abstractions that can be implemented by many languages and their associated runtimes.

Finally, Joe Duffy advertises a dream job [14]here.

1. [http://domino.research.ibm.com/Comm/www\\_ponder.nsf/Challenges/July2012.html](http://domino.research.ibm.com/Comm/www_ponder.nsf/Challenges/July2012.html)

2. <http://mathoverflow.net/questions/41939/a-balls-and-colours-problem>

3. [http://www.amazon.co.uk/Windows-Internals-Part-6th-Edition/dp/0735648735/ref=sr\\_1\\_1?ie=UTF8&qid=134356922](http://www.amazon.co.uk/Windows-Internals-Part-6th-Edition/dp/0735648735/ref=sr_1_1?ie=UTF8&qid=134356922)

8&sr=8-1

4. <http://www.os2museum.com/wp/?p=960>
5. <http://shop.oreilly.com/product/0790145369079.do>
6. <http://research.microsoft.com/apps/pubs/default.aspx?id=68298>
7. [http://biblion.epfl.ch/EPFL/theses/2010/4874/EPFL\\_TH4874.pdf](http://biblion.epfl.ch/EPFL/theses/2010/4874/EPFL_TH4874.pdf)
8. <http://channel9.msdn.com/Shows/Going+Deep/Bart-De-Smet-Rx-20-RC-Time-Error-Handling-SafeSubscribe-and-More>
9. <http://channel9.msdn.com/Blogs/Charles/.NET-45-in-Practice-Bing>
10. <http://channel9.msdn.com/Events/BUILD/BUILD2011/TOOL-813T>
11. <http://channel9.msdn.com/Events/Ch9Live/Channel-9-Live-at-Tech-Ed-Europe-2012/Dustin-Campbell-Roslyn>
12. <http://channel9.msdn.com/Events/TechEd/NorthAmerica/2012/DEV361>
13. <http://hanselminutes.com/329/understanding-winrt-and-windows-8-for-net-programmers>
14. <http://www.bluebytesoftware.com/blog/2012/07/18/WorkFun.aspx>

---

## 6.8 August

### 6.8.1 Did that get a reaction? (2012-08-08 09:18)

It did a lightning talk yesterday on Rx – the slides (a text document) can be found [1]here and the play project I used can be found [2]here.

1. [https://skydrive.live.com/download?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%21490&authkey=ALaN\\_pzsSqRaACa](https://skydrive.live.com/download?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%21490&authkey=ALaN_pzsSqRaACa)
  2. <https://skydrive.live.com/embed?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%21491&authkey=AP1vS3LFkcW-UUI>
- 

### 6.8.2 You've got that backwards (2012-08-25 06:16)

Every time a new version of the framework comes out, there are always new classes and members to learn. Of course, .NET 4.5 is backwards compatible with .NET 4.0, so I went ahead and upgraded my development machine to the new version. After running some tests on the upgraded machine, some of the unit tests started failing, and it took me a while to realise why. The sorting implementation in the base class libraries has changed – elements that are equal in the sort order, might no longer appear in the same relative positions using the new version of the framework.

For example, try running the following code in .NET 4 and .NET 4.5, and you'll see very different results. Such sorting predicates are an easy way to accidentally rely on unspecified behaviour.

[1]

654

```
static void Main(string[] args)
{
    var data = new List<KeyValuePair<int, int>>();
    for (int i = 0; i < 1000; i++)
    {
        data.Add(new KeyValuePair<int, int>(i % 10, i));
    }
    data.Sort((x, y) => x.Key.CompareTo(y.Key));
}
```

The fix I implemented was to make the sort predicate use more of the data to differentiate what were previously equal objects.

1. <http://clivetong.files.wordpress.com/2012/08/code.png>

---

### 6.8.3 Put down the tools (2012-08-28 05:10)

Last week was “down tools week” at [1]Red Gate. This happens several times a year and consists of a week of time during which developers are allowed to work on any project they like. Some people spend the time learning new tools, some prototype a new product and some just spend the time working on something they find interesting.

I chose to work with [2]Laurie on a project to do zip backup of an SQL database. This was to be based on a current Red Gate product, [3]HyperBac, which uses a device driver (a file system filter) and a user level service to intercept read and write calls on chosen files by chosen processes, replacing the data that the reading/writing process sees with a modification of the data that comes from the disc. The idea of the project was to gain some experience of [4]Clojure, a language that I’ve blogged a fair amount about in the past, particularly in the early 1.1 and 1.2 versions, though haven’t used it for some time.

The device driver communicates with the user process code using messages to and from a HyperBac device. This meant that we needed to use low level Win32 calls and deal with unmanaged memory, so we used the [5]JNA library to do this. Some of the project therefore involved transcribing datatypes implemented as C structs into the JNA equivalent. JNA made this fairly straight forward.

Clojure seems to have moved forward quite a lot since I last used it. It’s now really easy to get started, and if you are going to use [6]Leiningen you simply need to have the Java runtime installed on your machine and then you can this [7]script which will use [8]wget to download the various Clojure libraries when you run it with the self-install command line argument.

We did all of editing inside emacs, using the [9]swank-clojure code to allow us to compile and load editor buffers into the running Clojure session. This mode offers a clojure-jack-in command which searches up the directory structure from the location of the current buffer, finds a Leiningen project, starts Clojure and loads the project into the session, and then links

the emacs session into it. This makes it really easy to start a REPL, update it with modifications and other code changes, and then interactively test out the added functionality.

After a couple of blue screens, we started running inside a virtual machine with windbg attached across a virtual serial port as a kernel debugger. This made it fairly easy to debug the JNA calls that we were using, as we could see in the REPL (read-eval-print-loop - the interactive prompt) the data that we were trying to write to the device driver and then we could set breakpoints in the driver code and look at the data structures that were passed to see if they contained the right data. The REPL made the whole experience highly efficient – rather than writing a program to validate a series of messages to and from the driver, we could issue a request to the driver, check the result and then get that working before sending the next message.

[10]Leiningen projects have a straightforward structure. In order to use the swank integration, you just need to add the lein-swank plugin to the project definition. The project file defines the entry point of the code for when you want to generate a standalone application from the project. This works really well – you develop inside the REPL, dynamically adding code as you go along, and then produce a standalone jar that contains the final application and starts it at the defined entry point.

[11]

```
(defproject hb-clj "0.1.0-SNAPSHOT"
  :description "HyperBac service written in Clojure"
  :url "https://github.com/red-gate/hyperclj"
  :plugins [[lein-swank "1.4.4"]]
  :dependencies [[org.clojure/clojure "1.4.0"]
    [net.java.dev.jna/jna "3.4.0"]
    [net.java.dev.jna/platform "3.4.0"]]
  :warn-on-reflection true
  :java-source-paths ["src"]
  :compile-path "target/classes"
  :aot :all
  :main hyper-clj.core)
```

The warn-on-reflection causes Clojure to warn in cases where it needs to use reflection, allowing you to add type annotations to get better performance. These annotations allow the system to add a simple type check to code, after which the code can be specialised to expect the defined type.

The JNA definitions that come with some with the latest version of the library were missing some Win32 functions, so we defined them using the following JNA definition. The JNA library uses reflection to find the methods in the interface, and generates the appropriate interop thunks.

[12]

```

package interop;

import com.sun.jna.win32.StdCallLibrary;
import com.sun.jna.Native;
import com.sun.jna.Platform;
import com.sun.jna.Pointer;

import com.sun.jna.platform.win32.WinNT.HANDLE;
import com.sun.jna.ptr.LongByReference;
import com.sun.jna.platform.win32.WinDef.DWORD;
import com.sun.jna.platform.win32.WinNT.LARGE_INTEGER;

// Based on https://github.com/twall/jna/blob/master/www/GettingStarted.md

public interface Win32Extras extends StdCallLibrary {
    Win32Extras INSTANCE = (Win32Extras) Native.loadLibrary("kernel32", Win32Extras.class);

    // http://msdn.microsoft.com/en-us/library/windows/desktop/aa364957(v=vs.85).aspx
    boolean GetFileSizeEx(HANDLE handle, LARGE_INTEGER.ByReference size);

    // http://msdn.microsoft.com/en-us/library/windows/desktop/aa364439(v=vs.85).aspx
    boolean FlushFileBuffers(HANDLE handle);

    // http://msdn.microsoft.com/en-us/library/windows/desktop/aa365539(v=vs.85).aspx
    boolean SetFileInformationByHandle(HANDLE handle,
                                       int FileInformationClass,
                                       Pointer lpFileInformation,
                                       int dwBufferSize);
}

```

We could use these definitions inside Clojure using code like the following.

(.SetFileInformationByHandle Win32Extras/INSTANCE fh FileEndOfFileInfo size 8)

We used a Java zip library which expects you to provide an OutputStream. Clojure provides a way to implement Java interfaces, allowing you to name the function that is called to construct the new instance, and the implementations of the various methods from the interface.

[13]

```

(ns hyper-clj.HbOutputStream
  (:gen-class
   :extends java.io.OutputStream
   :state state
   :init init
   :constructors {[com.sun.jna.platform.win32.WinNT$HANDLE] []}
   :main false)
  (:require [hyper-clj.win32 :as win32])
  (:import [java.io ByteArrayOutputStream]))

```

After the above definition, we defined Clojure methods with the following signatures to provide the implementation.

```

(defn -init [fh] ...)
(defn -close [this] ...)
(defn -write-int [this b] ...)

```

The main complication was that the service needed to handle the file being written as a set of non-contiguous blocks which we needed to buffer and then pass through as contiguous blocks to the zip library code. There was also the complication of the device driver only accepting blocks with size a multiple of 512 bytes (apart from the final block). In order to write a smaller segment of memory, the existing driver needs to issue a read to get the full block into memory and then write back this data after the new values have been merged in. Our use of a zip stream allowed us to buffer and hence avoid this complication.

After getting the code working, and testing the entry point from the REPL, the Leiningen uberjar command produces a jar file that contains the entire application plus all of the libraries

on which it depends. This jar is easy to copy around and run on another machine.

The whole experience reinforced to me how useful a REPL is for exploring while you develop code. [When programming C # inside Visual Studio I sometimes do some experimentation from inside the immediate window, but this is far from being a proper REPL windows.] It's easy to check your understanding by making instances of various objects, calling methods and checking the results. [14]All the benefits of Lisp... while still being able to run and deploy like any old Java application.

1. <http://www.red-gate.com/>
  2. [http://www.linkedin.com/profile/view?id=92276&goback=%2Enpe\\_\\*1\\_\\*1\\_\\*1\\_\\*1\\_\\*1\\_\\*1%2Enpc\\_40260932\\_\\*1\\_\\*1\\_\\*1](http://www.linkedin.com/profile/view?id=92276&goback=%2Enpe_*1_*1_*1_*1_*1_*1_*1%2Enpc_40260932_*1_*1_*1)
  3. <http://www.red-gate.com/products/dba/sql-hyperbac/>
  4. <http://www.clojure.com/>
  5. <http://jna.java.net/javadoc/overview-summary.html>
  6. <http://leinigen.org/>
  7. <https://github.com/technomancy/lein>
  8. <http://gnuwin32.sourceforge.net/packages/wget.htm>
  9. <https://github.com/technomancy/swank-clojure>
  10. <http://en.wikipedia.org/wiki/Lein>
  11. <http://clivetong.files.wordpress.com/2012/08/project.png>
  12. <http://clivetong.files.wordpress.com/2012/08/jnaextras.png>
  13. <http://clivetong.files.wordpress.com/2012/08/outputstream.png>
  14. [http://winestockwebdesign.com/Essays/Lisp\\_Curse.html](http://winestockwebdesign.com/Essays/Lisp_Curse.html)
- 

## 6.9 September

### 6.9.1 I can see CLRly now the reign has gone (2012-09-03 05:32)

I've been reading a few interesting CLR related articles over the past few weeks.

The [1]Windows 8 FAQ is a plausible set of answers to a set of questions around the history of Silverlight/Wpf and the forthcoming Windows RT, mainly concerned with the internal Microsoft power struggle between the two Microsoft divisions responsible for the implementation. One of the points of view (which I don't agree with) is covered in a post on the code4k blog in the post [2]Future of Winrt which argues about the inefficiency of managed code and how the CLR should be a lot closer to unmanaged code (for example, allowing replaceable memory management libraries to be included). It will be interesting to see where the CLR moves in the next few years.

The CLR implementation is still moving forwards though. There's a good blog post, [3]Evolving the Reflection API, that covers the re-architecting of the Reflection APIs, breaking type information into the Type and TypeInfo classes, which will allow a program to reflect on an assembly and its types without actually loading it. The trick is to have different representations of type

references and type declarations. As in the Reflector API, the former don't require the type itself to be loaded.

I also recently came across some Channel9 videos covering [4]PerfView, a tool which assists with taking an ETW trace and then helps with decoding the contents. More and more of the operating system and other layered applications (like the CLR) can now provide ETW information which can be logged (in a high performance manner) and used later for analysis of performance or errors.

1. [http://www.riagenic.com/archives/960?utm\\_source=feedburner&utm\\_medium=feed&utm\\_campaign=Feed%3A+MsMossyblog%28MS+MossyBlog%29](http://www.riagenic.com/archives/960?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+MsMossyblog%28MS+MossyBlog%29)
  2. <http://code4k.blogspot.co.uk/#!/2012/08/going-native-20-future-of-winrt.html>
  3. <http://blogs.msdn.com/b/dotnet/archive/2012/08/28/evolving-the-reflection-api.aspx>
  4. <http://blogs.msdn.com/b/vancem/archive/2012/07/17/perfview-videos-on-the-web-finally.aspx>
- 

### 6.9.2 Does it compute? (2012-09-11 05:18)

[1]Turing's Cathedral: The origins of the Digital Universe by George Dyson

This book wasn't really what I was expecting. Its main subject is John Von Neumann with an emphasis on the time he spent affiliated with the Institute of Advanced Study in Princeton, and the kinds of projects that happened there around the Second World War. Von Neumann was a truly brilliant man, who did some fantastic work in the pure mathematics before he became interested in more applied mathematics such as the work on the first atomic bomb. During the course of this work on the bomb, huge teams worked as human computers in order to carry out the calculations required to simulate the mathematical model of the bomb, and this was obviously a motivating factor in deciding to get a team together that could build an electronic machine that could automate the process.

The book gives some of the history of the IAS, and then tells the story of the development of the first computer at the IAS and the projects that it was used for. In the course of the story we learn about a number of the people involved and their contributions – for example Stan Ulam and his Monte Carlo methods for solving hard mathematical problems. We don't learn any specifics about the computer that they designed – it would have been nice to have seen information on the instruction set, or example programs from the early days, but it's still an interesting read and I enjoyed it.

1. [http://www.amazon.co.uk/Turings-Cathedral-Origins-Digital-Universe/dp/0713997508/ref=sr\\_1\\_1?ie=UTF8&qid=1346789196&sr=8-1](http://www.amazon.co.uk/Turings-Cathedral-Origins-Digital-Universe/dp/0713997508/ref=sr_1_1?ie=UTF8&qid=1346789196&sr=8-1)
-

### **6.9.3 I'll get back to you sometime in the future (2012-09-13 05:20)**

I've spent the last week or two reading [1]this paper, which contrasts continuations and exceptions, by giving a mathematical proof that they can be separated by various expressions (where separating means giving two closed terms which are equal in one scenario and differ in another scenario). The paper also shows that you cannot implement state in either of these mechanisms. The paper is interesting as it shows how you can prove these things from the operational semantics of the language.

I got interested in continuations again after reading this [2]argument against call/cc and the [3]associated continuations page, and mean to spent a lot more time looking at them in the future.

1. <http://www.cs.bham.ac.uk/~hxt/research/exncontjournal.pdf>
  2. <http://okmij.org/ftp/continuations/against-callcc.html>
  3. <http://okmij.org/ftp/continuations/index.html>
- 

## **6.10 October**

### **6.10.1 It's certainly a type of script (2012-10-02 17:09)**

Microsoft certainly seem to be pushing TypeScript, yet another language that (trans)compiles to JavaScript. There is an introductory video [1]here, more detail in the video [2]here, the language specification [3]here and a [4]play area where you can see the TypeScript and its transcompiled form.

It's certainly worth a look, though it's not clear where this is all going in the long term. Certainly some of comments suggest that people don't really want yet another JavaScript++.

1. <http://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript>
  2. <http://channel9.msdn.com/posts/Anders-Hejlsberg-Steve-Lucco-and-Luke-Hoban-Inside-TypeScript>
  3. <http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf>
  4. <http://www.typescriptlang.org/Playground/>
- 

### **6.10.2 It's more about async than you might think (2012-10-03 06:10)**

I recently got to do a [1]presentation on C # 5. This included using Reflector to walk through some of the implementation of async. I've uploaded the [2]slides here.

1. <https://msevents.microsoft.com/CUI/EventDetail.aspx?culture=en-US&EventID=1032527696&CountryCode=US>
  2. <https://r.office.microsoft.com/r/rlidPowerPointEmbed?p1=1&p2=1&p3=SD3F21DF299C355E7F!492&p4=&ak=!APPkVRnTnuENjX0&kip=1>
- 

### 6.10.3 Another one bites the dust (2012-10-15 06:01)

Way back in 1990 I started a Phd at [1]Cambridge University Computer Laboratory in the area of automated theorem proving. My interest at the time was around the synthesis of functional programs from mathematical proofs of their correctness, but I remember that one of the groups at the computer laboratory was interested in implementing the real numbers in one of the programming logics that they used ([2]HOL). Since those days, things have moved on a lot, and in 2008 [3]Georges Gonthier published his [4]computer checked proof of the four colour theorem. More impressively, he and a team have recently finished [5]a proof of the Feit-Thompson Theorem, a complicated result from group theory. Of course, another long term computer proof attempt is the [6]Flyspeck project, the formalisation of Thomas Hales' proof of the [7]Kepler conjecture which is also nearing completion.

There's an interesting page [8]here which records 100 mathematical theorems that have been proved in a range of theorem provers and it will be interesting to see how far the formalisation of mathematics will go in the coming years, and how many holes they find in the existing proofs in the process!

1. [http://en.wikipedia.org/wiki/University\\_of\\_Cambridge\\_Computer\\_Laboratory](http://en.wikipedia.org/wiki/University_of_Cambridge_Computer_Laboratory)
  2. [http://en.wikipedia.org/wiki/HOL\\_\(proof\\_assistant\)](http://en.wikipedia.org/wiki/HOL_(proof_assistant))
  3. <http://research.microsoft.com/en-us/people/gonthier/>
  4. <http://www.ams.org/notices/200811/tx081101382p.pdf>
  5. <http://research.microsoft.com/en-us/news/features/gonthierproof-101112.aspx>
  6. <http://code.google.com/p/flyspeck/>
  7. [http://en.wikipedia.org/wiki/Kepler\\_conjecture](http://en.wikipedia.org/wiki/Kepler_conjecture)
  8. <http://www.cs.ru.nl/~freek/100/>
- 

### 6.10.4 Abort, abort ... no way! (2012-10-16 06:12)

Multi-core and its exploitation has brought a number of challenges to the programming world. It's clear that we need to exploit the many cores that are now available to us, but it can be hard work to manage and coordinate the activities that are running on many threads. Fortunately, from the .NET 4 framework onwards, the Task Parallel Library (TPL) has given us excellent abstractions for working in this world.

Tasks abstract the notion of a unit of work, and the library offers ways to coordinate tasks and add dependencies between them, allowing tasks to notify each other of completion and

hence get ordering guarantees. The TPL has a great cancellation model based on cooperative cancellation via the `CancellationToken` type and its methods which allow user code to regularly poll the token and throw an exception if a cancellation is desired. The TPL has a special well known exception that is used to communicate the co-operative cancellation of a task.

[1]

```
var tokenSource = new CancellationTokenSource();
var token = tokenSource.Token;

var myTask = new Task(() =>
{
    int i = 0;
    while (true)
    {
        Console.WriteLine(i++);
        token.ThrowIfCancellationRequested();
    }
});

myTask.Start();
tokenSource.Cancel();
```

With all of this, it was strange to [2]see a post discussing problems with framework classes when `Thread.Abort` is used. In code you see every day, very few classes would handle `Thread.Abort` raising an exception at some arbitrary place in the code. You can try to put a `finally` or `catch` around the state changing code, but of course, a second `Thread.Abort` could happen during the running of the fixup code. To harden the code sufficiently to handle this would be both impractical and a real waste of programmer time, and until we get truly transactional objects injecting an exception onto a thread is simply something you should not do. Joe Duffy covers this brilliantly in this [3]post.

1. <http://clivetong.files.wordpress.com/2012/10/cancellation.png>
2. <http://codebetter.com/patricksmacchia/>
3. <http://www.bluebytesoftware.com/blog/2009/03/13/ManagedCodeAndAsynchronousExceptionHardening.aspx>

---

## 6.10.5 Use the source Luke (2012-10-17 06:34)

There is an existing mechanism for handling source level debugging in languages that are translated to Javascript. The [1]Google Closure compiler can generate [2]source maps which can be used by a number of browsers, and despite some discussion to the contrary, the TypeScript compiler seems to have [3]the ability to do this as well. It's really nice being able to see [4]the full source for the TypeScript compiler on codeplex and also interesting to see [5]TypeScript interface definitions already available for a number of libraries. Now to find the excuse to use TypeScript in a project at work!

1. <https://developers.google.com/closure/>
2. <http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/>

3. <http://www.aaron-powell.com/web/typescript-source-maps>
  4. <http://typescript.codeplex.com/SourceControl/changeset/view/fe3bc0bfce1f>
  5. <http://typescript.codeplex.com/SourceControl/changeset/view/fe3bc0bfce1f#typings%2fjquery.d.ts>
- 

## 6.10.6 Very professional! (2012-10-18 05:45)

[1]Pro .NET Performance by Sasha Goldshtain with Dima Zurbalev and Ido Flatow

This is a really excellent book if you want to understand what is happening at the lower levels of the CLR and the base class library. It contains lots of information that I have only previously seen spread across lots of disparate blog posts and organises it into chapters that focus on individual performance concerns. Better still, it contains lots of information that I haven't seen before – some of it comes from the author's experience of the BCL and some comes from experiments that he has carried out on the various versions of the CLR. The author isn't afraid to get down into the actual assembly code when discussing some of the issues (such as the implementation of virtual dispatch), and has chosen x86 assembly code as the means of doing this. (This is fine by me, as x86 is a language I've programmed in for a long time, but it might have been nice to discuss the x64 architecture at some point).

Chapters 1 and 2 of the book set the scenario with a discussion of performance metrics (what we are actually trying to measure) and a look at some of the tools that can be used to do this measurement. These chapters contain a great introduction to ETW and the xperf tool that can be used to capture the logging information, following this with a look at the Visual Studio performance and memory allocation profilers, the concurrency profiler and the profiling tools available from Red Gate. These chapters stress the need for accurate measurement, and spend a little time dissecting a micro-benchmark which leads to a wrong conclusion owing to the optimisations that are carried out by the JIT.

Chapter 3, Type Internals, covers the in-memory layout of the data structures that the CLR uses at runtime, including the layout of the method tables, the management of sync blocks which are the underlying objects used for locking objects, and the use of value types to avoid the allocations that happen because of boxing. There's a good discussion of the IL that is generated when you make a call on a value type via an interface, and how you can use generics to avoid the boxing that would normally happen when you do this – ie why there is boxing in the first call to Boo in the following code, but not in the second.

[2]

```

static void Main(string[] args)
{
    var x = new Foo();
    ((IIInterface<Foo>)x).Boo();
    MakeCall(ref x);
}

interface IIInterface<T>
{
    int Boo();
}

static int MakeCall<T>(ref T x) where T:IIInterface<T>
{
    return x.Boo();
}

struct Foo : IIInterface<Foo>
{
    int x;
    public int Boo()
    {
        x++;
        return 10;
    }
}

```

The IL for Main looks like the following; notice the use of the box before the first call.

[3]

```

.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    .maxstack 2
    .locals init (
        [0] valuetype ConsoleApplication1.Program/Foo foo)
    L_0000: nop
    L_0001: ldloca.s foo
    L_0003: initobj ConsoleApplication1.Program/Foo
    L_0009: ldloc.0
    L_000a: box ConsoleApplication1.Program/Foo
    L_000f: callvirt instance int32 ConsoleApplication1.Program/IMine`1<valuetype ConsoleApplication1.Program/Foo>::Boo()
    L_0014: pop
    L_0015: ldloc.0
    L_0016: ldloc.0
    L_0017: call int32 ConsoleApplication1.Program::MakeCall<valuetype ConsoleApplication1.Program/Foo>(!0, !0)
    L_001c: pop
    L_001d: ret
}

```

The important point to notice is the use of the constrained IL instruction in the MakeCall static method.

[4]

```

.method private hidebysig static int32 MakeCall<(ConsoleApplication1.Program/IMine`1<!!T>) T>(!IT x, !IT y) cil managed
{
    .maxstack 1
    .locals init (
        [0] int32 num)
    L_0000: nop
    L_0001: ldarga.s x
    L_0003: constrained. !!T
    L_0009: callvirt instance int32 ConsoleApplication1.Program/IMine`1<!!T>::Boo()
    L_000e: stloc.0
    L_000f: br.s L_0011
    L_0011: ldloc.0
    L_0012: ret
}

```

Boxing can be a killer for some applications and is worth being aware of. The author shows the more modern ways to do object equality using the IEquatable<> interface which allows a non-boxing call to the equality method and contrasts this with the standard Object.Equals which requires a boxing of the argument (as it is on type object). In a later chapter he completes the

discussion of how generic types like `List<T>` use an [5]`EqualityComparer<T>` class to general a non-boxing equality when the type supports `IEquatable<T>`.

They do this by offering a `Default` property which caches the equality comparison.

[6]

```
public static EqualityComparer<T> Default
{
    [TargetedPatchingOptOut("Performance critical to inline across NGen image boundaries"), __DynamicallyInvokable]
    get
    {
        EqualityComparer<T> defaultComparer = EqualityComparer<T>.defaultComparer;
        if (defaultComparer == null)
        {
            defaultComparer = EqualityComparer<T>.CreateComparer();
            EqualityComparer<T>.defaultComparer = defaultComparer;
        }
        return defaultComparer;
    }
}
```

This equality comparer checks to see what interfaces the type supports, and if it supports `IEquatable<T>`,

[7]

```
private static EqualityComparer<T> CreateComparer()
{
    RuntimeType c = (RuntimeType) typeof(T);
    if (c == typeof(byte))
    {
        return (EqualityComparer<T>) new ByteEqualityComparer();
    }
    if (typeof(IEquatable<T>).IsAssignableFrom(c))
    {
        return (EqualityComparer<T>) RuntimeTypeHandle.CreateInstanceForAnotherGenericParameter((RuntimeType) typeof(GenericEqualityComparer<int>), c);
    }
    if (c.IsGenericType && (c.GetGenericTypeDefinition() == typeof(Nullable<T>)))
    {
        RuntimeType type2 = (RuntimeType) c.GetGenericArguments()[0];
        if (typeof(IEquatable<T>).MakeGenericType(new Type[] { type2 }).IsAssignableFrom(type2))
        {
            return (EqualityComparer<T>) RuntimeTypeHandle.CreateInstanceForAnotherGenericParameter((RuntimeType) typeof(NullableEqualityComparer<int>), type2);
        }
    }
    if (c.IsEnum && (Enum.GetUnderlyingType(c) == typeof(int)))
    {
        return (EqualityComparer<T>) RuntimeTypeHandle.CreateInstanceForAnotherGenericParameter((RuntimeType) typeof(EnumEqualityComparer<int>), c);
    }
    return new ObjectEqualityComparer<T>();
}
```

then the trick above is used to avoid the boxing.

[8]

```
internal class GenericEqualityComparer<T> : EqualityComparer<T> where T: IEquatable<T>
{
    // Methods
    [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image boundaries")]
    public GenericEqualityComparer()
    {
        public override bool Equals(object obj)
        {
            public override bool Equals(T x, T y)
            {
                public override int GetHashCode()
                {
                    [TargetedPatchingOptOut("Performance critical to inline across NGen image boundaries")]
                    public override int GetHashCode(T obj)
                    {
                        internal override int IndexOf(T[] array, T value, int startIndex, int count);
                        internal override int LastIndexOf(T[] array, T value, int startIndex, int count);
                    }
                }
            }
        }
    }
}
```

Chapter four is a brilliant introduction to Garbage Collection and the variations of it that are implemented in the CLR. It covers the use of generations and how these are mapped

to segments in virtual memory, the need for the threads to be brought to safe points before the garbage collection can occur, the costs of pinning, and gives really good coverage of the many flavours of garbage collection that the CLR offers. There is also discussion of the large object area, the effects that allocating large objects can have on performance and how you might use unmanaged memory instead of heap memory to avoid some of the associated overheads. There is also a brief mention of object pooling and a pointer to the System.ServiceModel.Channels.BufferManager abstract class which implements the facade for such a pool infrastructure for use by WPF.

Chapter five covers generics and the collection types, looking in some detail at the performance of the various collection types and contrasting C# generics with C++ templates and Java generics. There are a couple of good pages that look at cache considerations, and which point out that iterating through an array in the wrong way can be vastly inefficient because the code ends up working against the cache.

Chapter six looks at concurrency and parallelism as a means of improving performance. It starts off by considering the use of the ThreadPool and TPL Tasks, moving on to Parallel.For and Parallel.ForEach and PLINQ. The chapter then moves on to memory models, the in-built windows synchronisation mechanisms and more discussion of caches and false sharing and its performance problems. The chapter finishes off by looking at GPU computing, a means to get dramatic performance improvements for certain kinds of algorithms.

Chapter seven is a set of paragraphs describing various ideas around asynchronous IO, IO completion ports, which are supported by the ThreadPool.BindHandle mechanism, and the .NET Thread Pool.

[9]

```
public static bool BindHandle(SafeHandle osHandle)
{
    if (osHandle == null)
    {
        throw new ArgumentNullException("osHandle");
    }
    bool flag = false;
    bool success = false;
    RuntimeHelpers.PrepareConstrainedRegions();
    try
    {
        osHandle.DangerousAddRef(ref success);
        flag = BindIOCompletionCallbackNative(osHandle.DangerousGetHandle());
    }
    finally
    {
        if (success)
        {
            osHandle.DangerousRelease();
        }
    }
    return flag;
}
```

Analyzer

- Depends On
- Used By
  - System.IO.FileStream..ctor(SafeFileHandle, FileAccess, Int32, Boolean)
  - System.IO.FileStream.Init(String, FileMode, FileAccess, Int32, Boolean, FileShare, Int32, FileOptions, SECURITY\_ATTRIBUTES, String, Boolean, Boolean, Boolean)
  - System.IO.FileSystemWatcher.StartRaisingEvents() : Void
  - System.IO.Pipes.PipeStream.InitializeHandle(SafePipeHandle, Boolean, Boolean) : Void
  - System.IO.Ports.SerialStream..ctor(String, Int32, Parity, Int32, StopBits, Int32, Int32, Handshake, Boolean, Boolean, Boolean, Byte)
  - System.Net.Sockets.Socket.BindToCompletionPort() : Void
  - System.ServiceModel.Channels.MsmqQueue.GetHandleForAsync(Boolean&) : MsmqQueueHandle
  - System.ServiceModel.Channels.PipeConnection.EnsureBoundToCompletionPort() : Void
  - System.ServiceModel.Channels.PipeConnectionListener.CreatePipe() : PipeHandle

The discussion moves on to various problems in the area of socket IO, including message chunking, chatty protocols and the costs of (de)serialization. The configuration parameters for WCF are also covered.

Chapter eight is a brilliant set of observations and comments about unsafe code and interoperability with the unmanaged world, via native code and via COM. It goes down to a very low level, discussing the costs of P/Invoke in great detail, including some material on marshaller stubs that I haven't seen before. This was a very good chapter and very interesting, giving a good feeling for the overhead of using unmanaged calls in your .NET application.

Chapter nine looks at algorithms and their costs covering a few well known algorithms.

Chapter ten covers a miscellaneous set of other optimisations around the JIT, looking at the types of optimisations the JIT carries out, when range checking is eliminated for example, though this revolves around the author's experiments as the behaviour isn't documented by Microsoft. There's then some clever material on processor specific optimisation, looking at certain instructions that are available on some processor variants (such the SIMD extensions to the x86 instruction set), with examples showing how .NET code can call into an assembled series of bytes via a delegate generated using Marshal.GetDelegateForFunctionPointer. The chapter finishes with a look at Reflection and code generation, and the TypedReference type.

Chapter eleven looks at a series of optimisations for ASP.NET web applications.

The book is really good indeed, covering all of the topics in enough detail to give the reader a good understanding, and offering pointers to more material. The coverage of tools for measuring the performance is brilliant, particularly the ETW material, and I loved the low level details about interoperating with unmanaged code. If you like low level details about the CLR, the book is a must read.

1. [http://www.amazon.co.uk/Pro-NET-Performance-Professional-Apress/dp/1430244585/ref=sr\\_1\\_1?ie=UTF8&qid=1350309370&sr=8-1](http://www.amazon.co.uk/Pro-NET-Performance-Professional-Apress/dp/1430244585/ref=sr_1_1?ie=UTF8&qid=1350309370&sr=8-1)
  2. <http://clivetong.files.wordpress.com/2012/10/code1.png>
  3. <http://clivetong.files.wordpress.com/2012/10/i11.png>
  4. <http://clivetong.files.wordpress.com/2012/10/i12.png>
  5. <http://msdn.microsoft.com/en-us/library/ms224763.aspx>
  6. <http://clivetong.files.wordpress.com/2012/10/code2.png>
  7. <http://clivetong.files.wordpress.com/2012/10/code3.png>
  8. <http://clivetong.files.wordpress.com/2012/10/code4.png>
  9. <http://clivetong.files.wordpress.com/2012/10/threadpool.png>
- 

### **6.10.7 When I go, I'm going to take it with me (2012-10-19 06:07)**

I suppose it's obvious when you think about it, but when you ask the CLR to run a method on the thread pool there's a lot of context that you'd like to be taken across to the new thread – things like CAS permissions and identity need to be captured in the current thread and added to the ThreadPool thread while the method is executed.

In the following code, for example, you can see the logical call context is copied across to the executing method's ThreadPool thread.

[1]

```

static void Main(string[] args)
{
    CallContext.LogicalSetData("Clive", 10);
    ThreadPool.QueueUserWorkItem(
        delegate
        {
            Thread.Sleep(5000);
            Console.WriteLine(CallContext.LogicalGetData("Clive"));
        });
    CallContext.FreeNamedDataSlot("Clive");
    Console.WriteLine(CallContext.LogicalGetData("Clive") ?? "null");
    Thread.Sleep(10000);
}

```

This is all implemented in terms of ExecutionContexts, one of which is captured when the method is queued.

[2]

```

public static bool QueueUserWorkItem(WaitCallback callBack)
{
    StackCrawlMark lookForMyCaller = StackCrawlMark.LookForMyCaller;
    return QueueUserWorkItemHelper(callBack, null, ref lookForMyCaller, true);
}

```

[3]

```

private static bool QueueUserWorkItemHelper(WaitCallback callBack, object state, ref StackCrawlMark stackMark, bool compressStack)
{
    bool flag = true;
    if (callBack != null)
    {
        EnsureVMInitialized();
        try
        {
            return flag;
        }
        finally
        {
            QueueUserWorkItemCallback callback = new QueueUserWorkItemCallback(callBack, state, compressStack, ref stackMark);
            ThreadPoolGlobals.workQueue.Enqueue(callback, true);
            flag = true;
        }
    }
    throw new ArgumentNullException("WaitCallback");
}

```

[4]

```

internal QueueUserWorkItemCallback(WaitCallback waitCallback, object stateObj, bool compressStack, ref StackCrawlMark stackMark)
{
    this.callback = waitCallback;
    this.state = stateObj;
    if (compressStack && !ExecutionContext.IsFlowSuppressed())
    {
        this.context = ExecutionContext.Capture(ref stackMark, ExecutionContext.CaptureOptions.OptimizeDefaultCase | ExecutionContext.CaptureOptions.IgnoreSyncCtx);
    }
}

```

Looking at the Capture method, you can see all of the work that it does.

[5]

```

Reader executionContextReader = Thread.CurrentThread.GetExecutionContextReader();
if (executionContextReader.IsFlowSuppressed)
{
    return null;
}
SecurityContext context = SecurityContext.Capture(executionContextReader, ref stackMark);
HostExecutionContext context2 = HostExecutionContextManager.CaptureHostExecutionContext();
SynchronizationContext context3 = null;
LogicalCallContext context4 = null;
if (executionContextReader.IsNull())
{
    if ((options & CaptureOptions.IgnoreSyncCb) == CaptureOptions.None)
    {
        context3 = (executionContextReader.SynchronizationContext == null) ? null : executionContextReader.SynchronizationContext.CreateCopy();
    }
    if (executionContextReader.LogicalCallContext.HasInfo)
    {
        context4 = executionContextReader.LogicalCallContext.Clone();
    }
}
if (((((options & CaptureOptions.OptimizeDefaultCase) != CaptureOptions.None) && (context == null)) && ((context2 == null) && (context3 == null))) && ((context4 == null) && (options & CaptureOptions.IgnoreSyncCb) == CaptureOptions.None))
{
    return s_dummyDefaultEc;
}
ExecutionContext context5 = new ExecutionContext {
    SecurityContext = context
};
if (context5.SecurityContext != null)
{
    context5.SecurityContext.ExecutionContext = context5;
}
context5._hostExecutionContext = context2;
context5._syncContext = context3;
context5.LogicalCallContext = context4;
context5.isNewCapture = true;
return context5;

```

And using Reflector, we can see lots of places where the ExecutionContext is captured and then flowed.

[6]



1. <http://clivetong.files.wordpress.com/2012/10/execution1.png>
2. <http://clivetong.files.wordpress.com/2012/10/execution2.png>
3. <http://clivetong.files.wordpress.com/2012/10/execution3.png>
4. <http://clivetong.files.wordpress.com/2012/10/execution4.png>
5. <http://clivetong.files.wordpress.com/2012/10/execution5.png>
6. <http://clivetong.files.wordpress.com/2012/10/execution6.png>

---

## **6.10.8 It's the future (but not as we know it) (2012-10-21 06:47)**

[1]Windows 8 Secrets by Paul Thurrott and Rafael Rivera

It's almost here. Windows 8 goes on general release this week, and we are all going to have to get used to the new "Metro" interface whether we like it or not. Anyone who listens to the regular [2]Windows Weekly podcast understands that the first author knows his stuff when it comes to Microsoft and Windows, so it was clear that this book was going to be a good read. The book is very much a hands-on guide to the new operating system, spending a lot of space describing both the old desktop and the new Metro style interface and explaining the rationale behind the various concepts like the charms and application bars. The book covers various tricks around installation, covers the new application model and the Windows Store and describes the differences between the consumer and business versions of the operating system.

When I finally get my hands on Windows 8 on my home machine sometime this week, I'm going to re-read the book and program my hands with the many keyboard shortcuts that the book covers.

1. [http://www.amazon.co.uk/Windows-8-Secrets-Paul-Thurrott/dp/1118204131/ref=sr\\_1\\_1?ie=UTF8&qid=1350734271&s\\_r=8-1](http://www.amazon.co.uk/Windows-8-Secrets-Paul-Thurrott/dp/1118204131/ref=sr_1_1?ie=UTF8&qid=1350734271&s_r=8-1)
  2. <http://twit.tv/ww>
- 

## **6.10.9 It's certainly extensible (2012-10-22 06:49)**

[1]Professional Visual Studio Extensibility by Keyvan Nayyeri

Yet another book that I bought cheaply on Amazon. Slightly out of date as it deals with Visual Studio 2008, and it has a focus on extending Visual Studio using add-ins and macros which are not supported in the Visual Studio 2012. However, it does have a chapter on extensibility using Visual Studio packages, and the material on interacting with the Visual Studio object model is also still very relevant.

At work I've written extensions that interact with Visual Studio's extension model, and this book gives a very good overview of the things that you need to understand to work effectively. From a historical point of view the book also offers the information that helps you understand why things are done in a particular way.

There are chapters on the automation model, the anatomy of an add-in (covering the COM interfaces that it needs to implement), and an introduction to solutions, projects and project items. This latter chapter helps you navigate to the various interesting documents that are contained within the project structure, and then the following chapters describe the object model around documents - with these you can navigate their structure down to the level of methods and then manipulate the body of the method as text. There are some good examples

demonstrating the types of things you can do. The next chapter covers the build process, allowing your add-in to trigger and take part in the build process for a solution.

Later chapters cover debugging, building, deploying and localising an add-in, and the book finishes with chapters on extending the debugger, and the advantages of using the more modern package instead of the older add-in (as a package gives more access to Visual Studio). The last chapters cover project templates (very interesting), MSBuild (a slightly out-dated introduction) and macros (now irrelevant).

Given how little information there appears to be out there on the internet regarding Visual Studio Extensibility this is a very good introductory book that covers the area, with good examples, though obviously given the breadth the depth is sometimes a little shallow. Well worth a read though!

1. [http://www.amazon.co.uk/Professional-Visual-Studio-Extensibility-ebook/dp/B001QFYPOA/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1350733550&sr=1-1](http://www.amazon.co.uk/Professional-Visual-Studio-Extensibility-ebook/dp/B001QFYPOA/ref=sr_1_1?s=books&ie=UTF8&qid=1350733550&sr=1-1)
- 

### **6.10.10 Safe for parallelism (2012-10-31 07:10)**

I came across the abstract for [1]this paper while looking through the list of talks at [2]SPLASH 2012. Joe Duffy wrote a [3]fantastic book on concurrent programming in Windows and on his blog he has mentioned that he is working on an incubation project at Microsoft in the area of safe concurrency. In previous Channel 9 talks, he has hinted at improvements to the type system to make concurrent programming safe, including a notion of tracking isolated object clusters.

The paper discusses an in-use extension to C # which allows references to be marked as readable and writable (as well as isolated and immutable), and also tracks references which point to isolated object clusters. Such clusters are interesting for a parallelism point of view as it is safe for individual threads to process individual clusters of isolated objects. Allowing objects to also be marked as immutable, makes it possible to maintain isolated clusters by taking an isolated cluster, converting an object inside it to mutable, modifying, and then restoring the isolated property. This is just the kind of thing that happens in parallel algorithms where isolated nodes in a tree are mutated independently of other threads.

This looks like an interesting extension to the type system which has proved to be effective in a prototype compiler.

1. <http://homes.cs.washington.edu/~csgordon/papers/oops1a12.pdf>
  2. <http://splashcon.org/2012/>
  3. <http://www.amazon.co.uk/Concurrent-Programming-Windows-Architecture-Development/dp/032143482X>
-

## 6.11 November

### 6.11.1 Things are a'changing (2012-11-01 07:11)

It's always fun to come across some new data structures. When I studied computer science at the start of the 1990s, we had a lecture course that covered the standard data structures, but since those days there has been a lot more investigation of immutable data structures and also data structures that are more friendly to modern processor architectures.

The first data structure that I've recently come across is a suffix tree. Such trees can be constructed efficiently (in time linear in the size of the string), and can be used to answer questions such as whether the string contains a particular substring, and the longest common substring between two strings. This [1]wikipedia article is very good at listing the uses and the following [2]post explains the algorithm well, including a JavaScript implementation (if you view the page source). The following linked paper explains [3]the links between the three historical algorithms for implementing such trees.

The second data structure is the [4]Skip List, a data structure based on the binary tree but which uses a probabilistic balancing to avoid the complicated balancing algorithms that you find within the standard tree data structures. This [5]post explains the details well.

1. [http://en.wikipedia.org/wiki/Suffix\\_tree](http://en.wikipedia.org/wiki/Suffix_tree)
  2. <http://www.allisons.org/ll/AlgDS/Tree/Suffix/>
  3. <http://europa.zbh.uni-hamburg.de/pubs/pdf/GieKur1997.pdf>
  4. [http://en.wikipedia.org/wiki/Skip\\_list](http://en.wikipedia.org/wiki/Skip_list)
  5. <http://cg.scs.carleton.ca/~morin/teaching/5408/refs/p90b.pdf>
- 

### 6.11.2 That's one heck of a virtual machine (2012-11-02 07:23)

I've heard talk recently about how it would be useful to have a virtual machine running inside the browser, as an alternative to trans-compiling other languages into JavaScript and running that on the browser. I hadn't seen it before, but someone pointed out this [1]article that [2]links to this code, which runs a Linux kernel on top of a PC emulator written in JavaScript. The implementation uses an interface between JavaScript and raw memory in bytes [3]specified here.

1. [http://www.theregister.co.uk/2011/05/18/javascript\\_pc\\_emulator/](http://www.theregister.co.uk/2011/05/18/javascript_pc_emulator/)
  2. <http://bellard.org/jslinux/index.html>
  3. <http://www.khronos.org/registry/typedarray/specs/latest/>
-

### **6.11.3 I'm busy! (2012-11-03 07:02)**

[1]Async in C # 5.0 by Alex Davies

This is a short book. At 100 pages it feels more like an extended essay, but given that its focus is on asynchronous programming in C # 5, it works really well. The book starts with a simple example application – a Windows Forms application which downloads a number of icons from a web site, initially synchronously. This application is improved by making it asynchronous, firstly by using the pre-C # 5 patterns and then using the new Async feature which gets rid of the awkwardness of continuations passing style and use of objects like BackgroundWorker to get things happening on the correct thread.

This book is really good. It discusses the motivation for the Async feature, showing the problems that it seeks to solve, it covers what is happening under the surface without side-stepping issues such as Execution Contexts and Synchronization Contexts, and it gives a good introduction to Tasks and the subset of the task parallel library that is needed for the examples that are used. There are also informative remarks on unit testing Async code, parallelism and actors, and a discussion of how the C # Async feature is connected to the asynchronous operation model that is offered by WinRT.

The book is a quick read, but you will learn something from reading it.

1. [http://www.amazon.co.uk/Async-C-5-0-Alex-Davies/dp/1449337163/ref=sr\\_1\\_2?ie=UTF8&qid=1351605104&sr=8-2](http://www.amazon.co.uk/Async-C-5-0-Alex-Davies/dp/1449337163/ref=sr_1_2?ie=UTF8&qid=1351605104&sr=8-2)
- 

### **6.11.4 That does require some interpretation (2012-11-05 07:35)**

I very much enjoyed [1]this post on implementing fast interpreters and [2]its follow up. An interpreter executing an integer ADD instruction is a classic example of code where the overhead to get to the operation itself is much greater than the cost of the operation that is being implemented. The classic technique to avoid handle this, and the most effective optimisation of a VM runtime like HotSpot, is inlining, which the post mentions.

The follow up post has a pointer to [3]this thesis which is an interesting read on the subject. Sometimes it is just too much work to write a compiler, and this thesis gives a number of straightforward techniques for getting good performance.

1. <http://nominolo.blogspot.co.uk/2012/07/implementing-fast-interpreters.html>
  2. [http://nominolo.blogspot.co.uk/2012/07/implementing-fast-interpreters\\_31.html](http://nominolo.blogspot.co.uk/2012/07/implementing-fast-interpreters_31.html)
  3. <https://students.ics.uci.edu/~sbruntha/cgi-bin/download.py?key=thesis>
-

### **6.11.5 I'd rather not do all of that again (2012-11-14 07:22)**

The ahead of time compilation story in .NET has always been to use NGEN. This tool pre-compiles assemblies in a way that allows them to be shared, but until very recently didn't have a great story for managing the recompilations that are required if a change is made in a base assembly on which a later assembly depends. Windows 8 phone has introduced a new technology, MDIL, machine dependent IL and there's a good Channel 9 video interview covering it [1]here. The idea is that various constants which would normally be hardwired into the code, for example the offset of a field into an instance of a class, or the offset of a method in the vtable, can now be changed without recompiling the whole assembly (using a linker style relocation). A patent application covering this technology can be found [2]here.

The Security Now podcast also had an [3]intriguing episode covering the subject of [4]homomorphic encryption, a means for carrying out operations on encrypted data without actually decrypting and re-encrypting it. The episode mentioned this [5]Phd thesis which was the first viable realisation of this idea.

[6]This Channel 9 episode on the history of Windows 8 is also quite interesting.

1. <http://channel9.msdn.com/Shows/Going+Deep/Mani-Ramaswamy-and-Peter-Sollich-Inside-Compiler-in-the-Cloud-and-MDIL>
  2. <http://www.freepatentsonline.com/20110258615.pdf>
  3. <http://www.grc.com/sn/sn-376.htm>
  4. [http://en.wikipedia.org/wiki/Homomorphic\\_encryption](http://en.wikipedia.org/wiki/Homomorphic_encryption)
  5. <http://crypto.stanford.edu/craig/craig-thesis.pdf>
  6. <http://channel9.msdn.com/Blogs/Charles/Inside-Windows-8-Martyn-Lovell-and-Elliot-H-Omiya-The-Windows-Runtime>
- 

## **6.12 December**

### **6.12.1 That's the right script for the play (2012-12-11 07:28)**

I've just finished reading [1]ClojureScript: Up and running by Stuart Sierra and Luke Vanderhart. I've been wanting to try ClojureScript for quite some time. Every language seems to be trans-compiling to JavaScript these days – languages like F #, C #, Java and TypeScript, so I was looking forward to trying a language that also allows you to interact with the browser via a live REPL.

The book is an interesting read, and is aimed at people who haven't really used the Clojure language before. It covers setup of the ClojureScript library, the basics of the Clojure(Script) language including the emphasis on functional data structures and the sequence abstraction, and demonstrates how to write ClojureScript and run the resulting output on the browser. As someone who's used Clojure in the past, I could have easily got started just be reading blog posts such as [2]the following, though the discussion around getting a REPL working the browser and having all of the information in one place made things much easier. The text of the book

discusses working with Clojure without requiring the use of Leiningen, but given past projects I've worked on I'd always use this tool to take care of the build.

It's easy to get a first Clojure project working.

```
C:\Users\clive\Desktopshlein>lein new first-cljs
```

```
C:\Users\clive\Desktopshlein\first-cljs>mkdir src-cljs
```

```
C:\Users\clive\Desktopshlein\first-cljs\src-cljs>notepad hello.cljs
```

And add the following code, which will raise an alert dialog in the browser.

```
(ns first-cljs.main)
(javascript/alert "Hello")
```

We need to edit the project file so that it references the cljs Leiningen plugin,

```
C:\Users\clive\Desktopshlein\first-cljs>notepad project.clj
and add some additional parameters which will be passed to the plugin.
:plugins [[lein-cljsbuild "0.2.7"]]
:cljsbuild {
:builds [ {
:source-path "src-cljs"
:compiler {
:output-to "web/js/main.js"
:optimizations :whitespace
:pretty-print true } }]
}
```

We can build the resulting code using the Leiningen command line

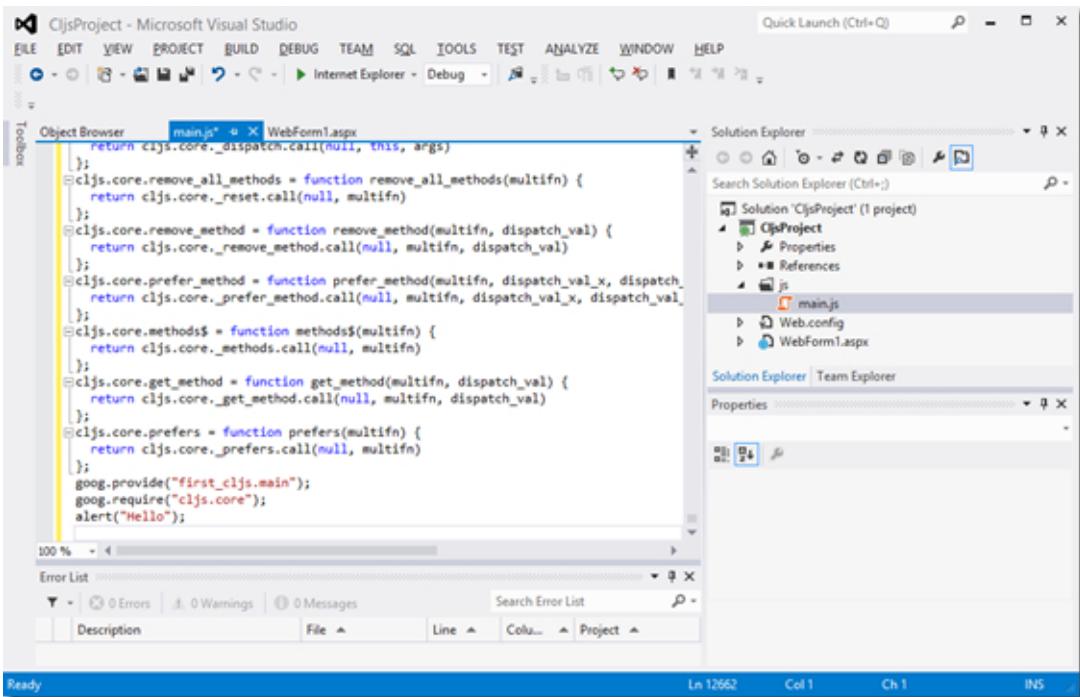
```
C:\Users\clive\Desktopshlein\first-cljs>..>lein cljsbuild once
```

```
C:\Users\clive\Desktopshlein\first-cljs\web\js>dir
Volume in drive C is Acer
Volume Serial Number is BA57-7470
```

```
09/12/2012 14:19 <DIR> .
09/12/2012 14:19 <DIR> ..
09/12/2012 14:40 447,225 main.js
```

```
1 File(s) 447,225 bytes
2 Dir(s) 572,977,061,888 bytes free
```

I copied the code into an ASP.NET web forms project. In the picture you can see the code that has been generated.



The code is rather large, and we can use the Google Closure compiler tool chain to optimise this just by changing the Optimizations argument in the project definition.  
`:optimizations :advanced`

Rebuilding with this option leads to a much smaller amount of JavaScript.

```
C:\Users\clive\Desktopshlein\first-cljs\web\js>dir
Volume in drive C is Acer
Volume Serial Number is BA57-7470
```

Directory of C:\Users\clive\Desktopshlein\first-cljs\web\js

```
09/12/2012 14:19 <DIR> .
09/12/2012 14:19 <DIR> ..
09/12/2012 14:47 69,273 main.js
1 File(s) 69,273 bytes
2 Dir(s) 572,975,824,896 bytes free
```

The Clojure compiler has applied a lot of sophisticated optimisations, though they do require that the JavaScript matches certain patterns (which the code from the Cljs compiler does). We can see that the names of the internal functions have been compressed and unused functions have been shaken away.

```

Object Browser main.js * WebForm1.aspx
  a.b = function (a) {
    L(a);
    a = M(a);
    return Pc.call(h, this, a)
  };
  return a
})();
Rc.prototype.apply = function (a, c) {
  return Pc.call(h, this, c)
};
alert("Hello");

```

Item(s) Saved

The real aim is to be able to test and debug the generated JavaScript inside a REPL that runs inside the browser.

To get the Clojure side of the REPL we need to add a dependency on [org.clojure/clojurescript "0.0-1450"] to the Clojure project definition.

To start the other end of the REPL inside the browser, we need to make the following call. (ns first-cljs.main)

```
(:require [clojure.browser.repl :as repl])
(repl/connect "[5]http://localhost:9000/repl")
```

We'll also modify the hosting web page to have a div that we can access by Id (called "TextArea")

[6]

Attribute	Value
id	TextArea
accesskey	
aria-activedescendant	
aria-atomic	False
aria-autocomplete	none
aria-busy	False

Item(s) Saved

Running the REPL inside a Clojure session, connects us to the browser.

```
C:\Users\clive\Desktop\shlein\first-cljs>..\lein repl
user=> (require 'cljs.repl)
user=> (require 'cljs.repl.browser)
user=> (cljs.repl/repl (cljs.repl.browser/repl-env))
```

We can now interact with hosted page, for example, grabbing the div and changing its contents.

```
ClojureScript:cljs.user> # _=> (def ta (.getElementById js/document "TextArea"))
ClojureScript:cljs.user> # _=> (set! (. ta -innerHTML) "Hello")
```

All considered, ClojureScript looks like a fairly good way to write and debug powerful web applications. I've always wondered about the problems of debugging trans-compiled JavaScript applications. Features like source maps allow you to get source level debugging for the instruction position, but I've not seen coercion of data types back into a representation that is meaningful in the original source language when the data is viewed via a window based debugger. In ClojureScript you debug via the REPL, and get a much more faithful textual representation of the original data value. I'd very much like to try it on a large example to find out.

1. [http://www.amazon.co.uk/ClojureScript-Up-Running-Stuart-Sierra/dp/1449327435/ref=sr\\_1\\_2?ie=UTF8&qid=1355061047&sr=8-2](http://www.amazon.co.uk/ClojureScript-Up-Running-Stuart-Sierra/dp/1449327435/ref=sr_1_2?ie=UTF8&qid=1355061047&sr=8-2)
  2. <http://onbeyondlambda.blogspot.com.au/2012/04/hello-world-in-clojurescript.html>
  3. <http://clivetong.files.wordpress.com/2012/12/project.png>
  4. <http://clivetong.files.wordpress.com/2012/12/project2.png>
  5. <http://localhost:9000/repl>)
  6. <http://clivetong.files.wordpress.com/2012/12/project3.png>
- 

### 6.12.2 Single page apps gone mad (2012-12-12 07:31)

[1]Programming Windows 8 Apps with HTML, CSS and JavaScript by Kraig Brockschmidt

Microsoft have done a lot of work to make it feasible to write full Windows 8 Store applications in JavaScript. The WinJS library contains many UI elements together with a (one-way) data binding model and the means to chain asynchronous operations using the Promise data type. This is all covered by the book, but the book goes further as it needs to cover the whole API surface. Hence as well as covering the JavaScript library in great detail, it also explains and describes the App Model of WinRT and it does this very well – by far the best explanation I have seen so far.

Some of the material is covered by these [2]talks [3]from [4]/BUILD, but having one very readable explanation is really good.

Without writing a full application, it is hard to know if JavaScript scales to large scale applications. However, the WinJS takes care of a lot of the boilerplate code for you. It is really easy to do data binding

[5]

```
<body>
    <div id="DatabindExample" data-win-bind="innerText: name"></div>
    <button id="Action">Do something</button>

    <script type="text/javascript">
        var datum = { name: "Unknown" };
        var bindable = WinJS.Binding.as(datum);

        WinJS.Binding.processAll(
            document.getElementById("DatabindExample"), bindable);

        document
            .getElementById("Action")
            .addEventListener("click",
                function () { bindable.name = "Changed"; });
    </script>
</body>
```

and use promise top defer functions until results are available.

[6]

```
<body>
    <script type="text/javascript">
        var promiseComplete;
        var promise =
            new WinJS.Promise(function (c, e) { promiseComplete = c; });
        promise
            .then(function (v) { return v + 1; })
            .then(function (v) { return 2 * v; })
            .then(function (v) { console.warn(v); });
        promiseComplete(6);
    </script>
</body>
```

There is also a model for defining Namespace and Classes, and a declarative means (data-win-control) for using controls.

[7]

```
<body>
    <script type="text/javascript">
        WinJS.Namespace.define("Clive", {
            MyLabel: WinJS.Class.define(
                function (element, options) {
                    this._setElement(element);
                    this._element.textContent = options.text;
                },
                {
                    _element: null,
                    _setElement: function (data) {
                        this._element = data;
                    }
                }
            );
    </script>
    <label
        data-win-control="Clive.MyLabel"
        data-win-options="{ text: 'hello' }">
    </label>
</body>
```

If you're going to be developing for this platform, the book is well worth a read whether you

are going to be writing in JavaScript or not.

1. [http://blogs.msdn.com/b/microsoft\\_press/archive/2012/10/29/free-ebook-programming-windows-8-apps-with-html-and-javascript.aspx](http://blogs.msdn.com/b/microsoft_press/archive/2012/10/29/free-ebook-programming-windows-8-apps-with-html-and-javascript.aspx)
  2. <http://channel9.msdn.com/Events/Build/2012/3-115>
  3. <http://channel9.msdn.com/Events/Build/2012/4-101>
  4. <http://channel9.msdn.com/Events/Build/2012/3-126>
  5. <http://clivetong.files.wordpress.com/2012/12/binding.png>
  6. <http://clivetong.files.wordpress.com/2012/12/promise.png>
  7. <http://clivetong.files.wordpress.com/2012/12/controls.png>
- 

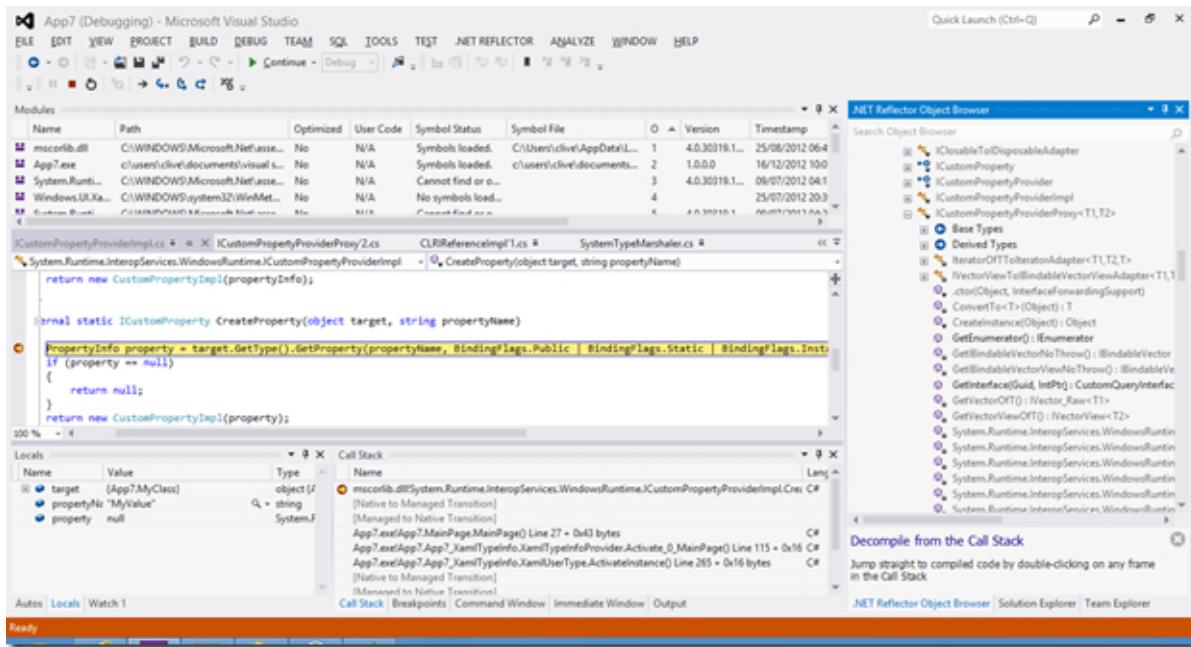
### 6.12.3 Bind it tight (2012-12-17 07:09)

Data-binding has become one of the central ideas amongst modern GUI frameworks. It has enabled the use of view models, a set of objects representing the GUI state which can be linked to the actual GUI elements in a very code free way. This means that once you have tested your view model, you can be confident that the GUI is showing the right information.

In the pre-WinRT days, the data binding was all implemented in managed code, in the same language as the code which controlled the GUI. In the WinRT world however, we need to be able to bind across languages and components (in the C++ case). There's an interesting post [1]here which looks at optimisations that the .NET WinRT wrapper provides to make this more efficient. For C++ there needs to be a way for the code implementing the binding to get hold of the metadata about the properties. This is implemented by having the compiler generate a class which can provide this information at runtime. In the C # CLR case we have Reflection, so it is possible for the compiler to avoid doing this, without all this code generation.

The original post doesn't really give a clear guide as how the external code gets hold of the type information. So I got out my copy of [2].NET Reflector, set the system up for debugging by globally setting the COMPLUS\_ZAPDISABLE environment variable, and placed an mscorelib.ini file next to mscorelib.dll to stop it being optimised too much.

We can now breakpoint the code to watch the creation of the object which supports the ICSharpInterface interface, which is exported via normal CLR COM interop back to the WinRT framework.



Using Reflector is easy to look at the WinRT support that has been included into the framework, which includes a lot of type that marshal between the WinRT types like `IVector` and `IList` and their CLR counterparts.

The screenshot shows the .NET Reflector Object Browser with three panels labeled [4], [5], and [6]. Panel [4] shows the `System.Runtime.InteropServices.WindowsRuntime` namespace with its members: `BindableIterableToEnumerableAdapter`, `BindableVectorToCollectionAdapter`, `BindableVectorToListAdapter`, `CLRIKeyValuePairImpl<K,V>`, `CLRIPropertyValueImpl`, `CLRIReferenceArrayImpl<T>`, `CLRIReferenceImpl<T>`, `ConstantSplittableMap< TKey, TValue >`, `CustomPropertyImpl`, `DefaultInterfaceAttribute`, `DesignerNamespaceResolveEventArgs`, `DictionaryKeyCollection< TKey, TValue >`, `DictionaryKeyEnumerator< TKey, TValue >`, `DictionaryToMapAdapter`, `DictionaryValueCollection< TKey, TValue >`, `DictionaryValueEnumerator< TKey, TValue >`, `EnumerableToBindableIterableAdapter`, `EnumerableToleratorAdapter`, `EnumeratorToleratorAdapter<T>`, `EventRegistrationToken`, `EventRegistrationTokenTable<T>`, `GetEnumerator_Delegate<T>`, `HSTRING_HEADER`, and `IActivationFactory`. Panel [5] shows the `IBindableIterable` interface with its members: `IBindableIterable`, `IBindableIterator`, `IBindableVector`, `IBindableVectorView`, `ICloseable`, `IClosableToDisposableAdapter`, `ICustomProperty`, `ICustomPropertyProvider`, `ICustomPropertyProviderImpl`, `ICustomPropertyProviderProxy<T1,T2>`, `IDisposableToClosableAdapter`, `IGetProxyTarget`, `Iterable<T>`, `Iterator<T>`, `KeyValuePair<K,V>`, `IManagedActivationFactory`, `IMap<K,V>`, `IMapView<K,V>`, `IMapViewToReadOnlyDictionaryAdapter`, `Indexer_Get_Delegate<T>`, `InterfaceForwardingSupport`, `InterfaceImplementedInVersionAttribute`, `IPropertyValue`, `IPropertyValueFactory`, `ReadOnlyDictionaryToMapViewAdapter`, `ReadonlyListToVectorViewAdapter`, `IReference<T>`, `IReferenceArray<T>`, and `IReferenceFactory`. Panel [6] shows the `IVectorViewToBindableVectorViewAdapter<T1,T>` interface with its members: `IVectorViewToReadOnlyListAdapter`, `ListToBindableVectorAdapter`, `ListToBindableVectorViewAdapter`, `ListToVectorAdapter`, `ManagedActivationFactory`, `MapToCollectionAdapter`, `MapToDictionaryAdapter`, `MapViewToReadOnlyCollectionAdapter`, `NamespaceResolveEventArgs`, `Point`, `.PropertyType`, `ReadOnlyArrayListAttribute`, `ReadOnlyDictionaryKeyCollection< TKey, TValue >`, `ReadOnlyDictionaryKeyEnumerator< TKey, TValue >`, `ReadOnlyDictionaryValueCollection< TKey, TValue >`, `ReadOnlyDictionaryValueEnumerator< TKey, TValue >`, `Rect`, `ReturnValueNameAttribute`, `RuntimeClass`, `Size`, `UnsafeNativeMethods`, `VectorToCollectionAdapter`, `VectorToListAdapter`, `VectorViewToReadOnlyCollectionAdapter`, `WindowsFoundationEventHandler< T >`, `WindowsRuntimeBufferHelper`, `WindowsRuntimeImportAttribute`, and `WindowsRuntimeMarshal`.

Lots of this code is quite low level and quite interesting to read, and lots of it sits on top of the existing .NET COM interop layer with its RCW and CCW wrappers.

Shawn Farkas has a nice MSDN magazine [7] article that talks about this interop and how it works.

1. <http://jaylee.org/post/2012/11/26/DataBinding-performance-in-WinRT-and-the-Bindable-attribute.aspx>
2. <http://www.reflector.net/>
3. <http://clivetong.files.wordpress.com/2012/12/property.png>
4. <http://clivetong.files.wordpress.com/2012/12/winrtsupport.png>

- 
5. <http://clivetong.files.wordpress.com/2012/12/winrtsupport2.png>
  6. <http://clivetong.files.wordpress.com/2012/12/winrtsupport3.png>
  7. <http://msdn.microsoft.com/en-us/magazine/jj651569.aspx>

---

#### 6.12.4 I came to find out about the barrier, and found RCU (2012-12-19 07:10)

I read my way through the appendix of [1]Is Parallel Programming Hard, and, if so, what can you do about it? which covered modern CPU architecture memory barriers, in my quest to understand them better. That appendix alone, Appendix C, is a very informative read, which makes things clearer than a good many other articles I've read in the past.

After reading that, I started with the rest of the book which is a great overview of writing scalable parallel algorithms - using techniques like code locking, data locking and breaking data into independent chunks with later synchronisation. Moreover, it covers a reader-writer variant called [2]RCU which the author helped to make a fundamental construct in the scalable Linux 2.6 kernel. [3]Paul McKenney's phd thesis covers the implementation in detail, as well as contrasting it with alternatives.

RCU is a technique for allowing readers access to a data structure whilst simultaneously allowing writers to mutate copies of the data structure and then update them in place to make the change visible. The key is then to avoid freeing any memory until the system can be guarantee that no readers are still using the old version of the object. RCU manages this by scoping read access to a structure between quiescent states of the CPUs, having readers notify that they are using the structure, and allowing actions to be queued on the old structure so that they execute when a suitable quiescent state is reached. It is a very clever idea, which scales very well, and which is used many times inside the Linux kernel.

- 
1. <http://www.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.2011.01.02a.pdf>
  2. <http://en.wikipedia.org/wiki/Read-copy-update>
  3. <http://www.rdrop.com/users/paulmck/RCU/RCUdissertation.2004.07.14e1.pdf>

---

#### 6.12.5 So many architectures, so little time (2012-12-21 07:11)

I've been reading a large amount of compiler related papers lately.

First, [1]a slide show on the optimisation of Haskell programs, where the author highlights the use of strictness analysis to detect the parts of the program which can be executed eagerly, and hence without the need to allocate numerous thunks. The compiler can use unpacking to avoid allocation of intermediate boxed data items on the heap, which also improves the performance.

Second a [2]blog post that discusses the LLVM compiler and how it maps an intermediate representation of an instruction into a target instruction on the target architecture. This very much

reminds me of the compiler architecture we had in my first job. The compiler targeted an instruction set known as HARP (Harlequin Abstract RISC Processor), and then HARP instructions were translated into machine instructions using a template matching scheme. HARP had an infinite set of registers, and the register colouring happened as part of this templating processing. (A paper describing one of the uses of HARP, in the Chameleon project which focussed on dynamic process migration can be found [3]here).

There's a good document [4]here on the x64 calling convention and architecture. Useful for debugging problems in x64 jitted CLR code.

Finally, in the theme of performance, GPU architecture is becoming an important way of generating high performance solutions for certain types of algorithm. A general introduction to GPUs can be found [5]here and document on using CUDA to program such architectures can be found [6]here.

1. <http://www.slideshare.net/ilyasergey/static-analyses-and-code-optimizations-in-glasgow-haskell-compiler#b tnNext>
  2. <http://eli.thegreenplace.net/2012/11/24/life-of-an-instruction-in-llvm/>
  3. <http://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=1&cad=rja&ved=0CDQQFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3>
  4. [http://software.intel.com/sites/default/files/m/d/4/1/d/8/Introduction\\_to\\_x64\\_Assembly.pdf](http://software.intel.com/sites/default/files/m/d/4/1/d/8/Introduction_to_x64_Assembly.pdf)
  5. [http://software.intel.com/sites/default/files/m/d/4/1/d/8/Introduction\\_to\\_x64\\_Assembly.pdf](http://software.intel.com/sites/default/files/m/d/4/1/d/8/Introduction_to_x64_Assembly.pdf)
  6. [http://people.maths.ox.ac.uk/~gilesm/hpc/NVIDIA/NVIDIA\\_CUDA\\_Tutorial\\_No\\_NDA\\_Apr08.pdf](http://people.maths.ox.ac.uk/~gilesm/hpc/NVIDIA/NVIDIA_CUDA_Tutorial_No_NDA_Apr08.pdf)
-



# 2013

## 7.1 January

### 7.1.1 I brought the glue for the model (2013-01-14 07:23)

[1]The Axiom of Choice by Thomas J. Lech

Ever since I did the course on Set Theory and Logic at university, I have found the [2]Axiom of Choice quite fascinating. Applications of the axiom of choice, mostly via Zorn's lemma, are found all over the place in undergraduate mathematics, where it is pointed out that the axiom is sometimes too powerful and leads to a number of undesirable results. In particular, the existence of unmeasurable sets of real numbers and the decomposition of the sphere into two spheres of the same size (using a physically unrealisable construction). The introduction to the book covers these undesirable results.

Chapter two looks at the good consequences of the axiom, and chapter three looks at what it means for an axiom system to be consistent, discussing models and relative consistency. Chapter four introduces set theory extended with atoms, items that don't have members, and shows that in this theory the axiom of choice is unprovable and is independent of the ordering principle.

Chapter five looks at generic models and forcing to demonstrate the independence of the Axiom of Choice – the technique of forcing is what allowed Paul Cohen to prove the independence of the Continuum Hypothesis in the 1960s. Chapter six looks at embedding, showing how consistency of ZF can be moved to the construction of various permutation models. There are other chapters on various independence results, mathematics without the axiom of choice and a look at some properties that contradict the axiom of choice.

The book is obviously quite technical, but has good high level explanations of how the proof is going to flow, and the chapters end with a discussion of the importance of the various results and their historical setting. If you find set theory and logic interesting, it is well worth a look.

1. [http://www.amazon.co.uk/The-Axiom-Choice-Thomas-Jech/dp/0486466248/ref=sr\\_1\\_1?ie=UTF8&qid=1357485907&sr=8-1](http://www.amazon.co.uk/The-Axiom-Choice-Thomas-Jech/dp/0486466248/ref=sr_1_1?ie=UTF8&qid=1357485907&sr=8-1)
2. [http://en.wikipedia.org/wiki/Axiom\\_of\\_choice](http://en.wikipedia.org/wiki/Axiom_of_choice)

## 7.1.2 Give us a job (2013-01-16 07:25)

[1]Cracking the coding interview by Gayle Laakmann McDowell

After a very brief introduction to techniques for being successful at interview, this book covers around 150 coding questions in a great deal of detail. These questions range from simple data structure questions on strings and arrays, stack and queues, and trees, through scalability and mathematical puzzlers to language related questions. Full answers are given to each of the questions, often with more than one answer being covered and with an emphasis on techniques for approaching the problems.

I certainly learned a few things from the solutions, though I think there were a couple of bugs in some of the solutions, in particular doing a binary search using a mid point calculated via (low + high)/2 can cause a problem if you use an integer datatype in languages like C where the addition can overflow and change sign.

Good fun, and good for keeping the brain exercised, the book seems to emphasise working for Microsoft or Google which I don't believe is everyone's dream. Some good questions.

1. [http://www.amazon.co.uk/Cracking-Coding-Interview-Programming-Questions/dp/098478280X/ref=sr\\_1\\_1?ie=UTF8&qid=1357928738&sr=8-1](http://www.amazon.co.uk/Cracking-Coding-Interview-Programming-Questions/dp/098478280X/ref=sr_1_1?ie=UTF8&qid=1357928738&sr=8-1)
- 

## 7.1.3 Something for Hercules to do later (2013-01-18 07:28)

Just before Xmas I attended [1]Liam Westley's talk at the NxtGen user group meeting on Async, where he covered the [2]Task-asynchronous pattern with a great series of examples. He demonstrated the use of various combinators such as WhenAll and WhenAny which are defined on the Task type.

During the talk I started wondering about how you might implement WhenAny. I knew that the TPL offers a ContinueWith method on a Task, and using this and subscribing to all of the tasks supplied to the method, and using some synchronisation primitive to monitor the completion events it looked like you'd be able to implement this yourself. However, I couldn't remember if Task offered a way to unsubscribe from ContinueWith, which is something you'd need in order efficiently cancel if any of the tasks throws an error.

Task is a really nice type that allows you to construct a tree of dependencies, then start one of the tasks which will start the execution of later elements in the tree. For example, in the following code we set up tasks x and y, configuring z to start when one of x or y is finished, and having t depend on z finishing.

```
var x = new Task<int>(() => { return 10; });
var y = new Task<int>(() => { return 20; });
var z = Task<int>.WhenAny<int>(x, y);
var t = z.ContinueWith(v => v.Result);
x.Start();
Thread.Sleep(5000);
```

[3]

If we set a breakpoint on the line with the Start method, if we inspect x in the watch window we can see that it contains a field m\_completionObject,

[4]

You could write code of the Invoke method on this type yourself.

```
public void Invoke(Task completingTask)
{
    if (base.TrySetResult(completingTask))
    {
        IList<Task> list = this._tasks;
        int count = list.Count;
        for (int i = 0; i < count; i++)
        {
            Task task = list[i];
            if ((task != null) && !task.IsCompleted)
            {
                task.RemoveContinuation(this);
            }
        }
        this._tasks = null;
    }
}
```

[5]

TrySetResult is the gateway that prevents the completion being set more than once, but I couldn't recall a public way of unsubscribing after doing a ContinueWith, and the code above calls RemoveContinuation so it looked like the continuation task might be released after the task had finished. I was confused when I looked in the debugger and noticed that the completion hadn't been unset. I got out [6]Reflector Pro and set a breakpoint on the RemoveContinuation method, only to notice that executes code only if the m\_continuationObject contains a List<> rather than a single subscriber.

In summary, one could (more or less) implement WhenAny using publically exposed items.

1. <http://www.nxtgenug.net/ViewEvent.aspx?EventID=548>
2. <http://download.microsoft.com/download/5/B/9/5B924336-AA5D-4903-95A0-56C6336E32C9/TAP.docx>
3. <http://clivetong.files.wordpress.com/2013/01/threadamy.png>
4. <http://clivetong.files.wordpress.com/2013/01/continuationobject.png>
5. <http://clivetong.files.wordpress.com/2013/01/invoke.png>
6. <http://www.reflector.net/>

---

#### 7.1.4 Get taller (2013-01-23 07:31)

[1]Growing Object-oriented software, guided by tests by Steve Freeman and Nat Pryce

Recommended to me by someone at work, this was a fantastic read. The introductory third of the book is a really good exposition of object-oriented design using test driven development. Rather than being a theoretical guide, this book seems to be full of practical advice and descriptions from the authors' working experience. The majority of the book is a worked example of the development of a full application using the test driven methodology, starting with the walking skeleton, an initial implementation that exercises all of the components of the system

in order to check that everything is correctly installed, and developing in very small steps and associated refactorings to get to a final application.

The authors are not fixated by unit testing a single class, but also emphasise integration tests that work between components (which seems a much more realistic goal to me). They use a “Tell don’t ask” style where the object is told to do something and takes the responsibility for the action, in turn calling on to collaborators, rather than using Java style property accesses to get an object from another object in order to tell the returned object to do some action. This was the first form of object orientation where the emphasis was on message passing between objects rather than the more typical function calling style. This leads to objects being tested by mocking their dependents, telling the object to do something and then checking that the interaction was correct. This contrasts with the more usual invasive style of setting up an object, calling a method on it and then accessing the private state to see if the right things happened. This style can be quite brittle in that you can’t change the implementation without changing the tests. With the style of this book, we can reimplement the object and still hope that the tests pass as the objects interacts as expected.

Objects require other objects which they use in one of three roles. Dependents are objects that have their methods called and are therefore the things that need to be mocked during tests so that we can test the interactions. Notifications are objects that need to be notified of changes of the object’s state, but in a fire-and-forget style so that the object doesn’t care how many listeners are present at any time. Adjustments are configuration type objects which customise the object for the system in which it is used.

The worked example is really good, showing in great detail how small steps can get us a long way, with the tests being maintained throughout. I’m hoping to get a chance to put some of this into practice in the near future.

1. [http://www.amazon.co.uk/Growing-Object-Oriented-Software-Guided-Signature/dp/0321503627/ref=sr\\_1\\_1?ie=UTF8&qid=1357932415&sr=8-1](http://www.amazon.co.uk/Growing-Object-Oriented-Software-Guided-Signature/dp/0321503627/ref=sr_1_1?ie=UTF8&qid=1357932415&sr=8-1)
- 

### 7.1.5 I’d like a new kernel please (2013-01-25 07:32)

I recently read [1]Linux Kernel Development by Robert Love

Loved the book. I’d read [2]The Design of the Unix Operating System by M.J.Bach when I was a student many years ago, but I’d recently come along some tutorial articles describing how you’d go about building your own operating system and wanted to see how these ideas were used to write Linux. You can easily get the Unix sources out of git, but it is useful to have some idea of the kernel’s design before you start exploring.

This books serves the purpose really well. It doesn’t go into detail about the actual kernel boot process, but does cover how to get the Linux sources from git. It then covers the various parts of the kernel architecture, starting with processes and the scheduler, the process for making system calls, the various important kernel data structures and their implementation. It moves on to interrupt handling, discussing top and bottom halves, the former being the code that runs in a fairly uninterruptible state in order to service the device, often by recording information that is later processed by a lower priority bottom half.

Kernel synchronisation and time management are covered in detail followed by information on devices and their interface, followed by a discussion of memory management and the process address space.

The book is really well written, and gives the reader a really good idea of the implementation of Linux. There are a few questions that weren't covered: how are signals delivered to a process (and how does the process handle them), and how is paging implemented, but this book is well worth reading if you want to start experimenting with the Linux kernel. I went out and grabbed Ubuntu and installed it on a virtual machine, and have been trying a copy of the kernel from a git checkout, which isn't a pain free process which I will cover in another blog post.

1. [http://www.amazon.co.uk/Linux-Kernel-Development-Implementation-Developers/dp/0672329468/ref=sr\\_1\\_1?ie=UTF8&qid=1357495064&sr=8-1](http://www.amazon.co.uk/Linux-Kernel-Development-Implementation-Developers/dp/0672329468/ref=sr_1_1?ie=UTF8&qid=1357495064&sr=8-1)
  2. [http://www.amazon.co.uk/Design-Operating-System-Prentice-Hall-Software/dp/0132017997/ref=sr\\_1\\_5?ie=UTF8&qid=1357502587&sr=8-5](http://www.amazon.co.uk/Design-Operating-System-Prentice-Hall-Software/dp/0132017997/ref=sr_1_5?ie=UTF8&qid=1357502587&sr=8-5)
- 

### **7.1.6 It's hard to get across the border (2013-01-28 07:40)**

Writing managed code always involves interoperation with low level C style APIs at some point. It is an interesting topic – you want to make it easy to pass arguments between the managed and the unmanaged worlds. You want to make the datatypes exposed by the other side look natural to the consumer, which can be hard as the semantics can be very different (think strings), and you want the performance to remain good – though you'll probably only be able to expose APIs that are not too fine grained.

I've recently read some articles on this subject - interoperation between the [1]CLR and the Windows Runtime and between [2]managed C++ and other code.

1. <http://go.microsoft.com/fwlink/?LinkId=243099>
  2. <http://blogs.msdn.com/b/abhinaba/archive/2012/11/14/c-cli-and-mixed-mode-programming.aspx>
- 

### **7.1.7 I want to see what's behind the window (2013-01-30 06:37)**

Windows 8 runs WinRT applications inside a sandbox which is designed to protect the OS from rogue applications – no more rootkits is the hope, and a control over applications that can use parts of the system like the camera and GPS. I thought [1]this article was a good start to understanding the app container model and it links to a blog post describing how the [2]app container idea applies to IE. The Mozilla wiki also has a good explanation of the runtime broker [3]here. There are already [4]articles on getting around Microsoft's controls to allow arbitrary

code to run on your ARM based Surface.

1. <http://recxltd.blogspot.co.uk/2012/03/windows-8-app-container-security-notes.html>
  2. <http://blogs.msdn.com/b/ieinternals/archive/2012/03/23/understanding-ie10-enhanced-protected-mode-network-security-addons-cookies-metro-desktop.aspx>
  3. [https://wiki.mozilla.org/Windows\\_8\\_Integration](https://wiki.mozilla.org/Windows_8_Integration)
  4. <http://surfsec.wordpress.com/2013/01/06/circumventing-windows-rts-code-integrity-mechanism/>
- 

## 7.2 February

### 7.2.1 No white pyjamas here (2013-02-19 07:28)

[1]Dojo - The Definitive Guide by Matthew A Russell

I've been doing some reading about HTML5 and WinJS lately, and so I'm fairly interested in the history behind the various JavaScript frameworks that have been developed since the AJAX revolution made JavaScript a serious technology for client side application development. I found this book in the Cambridge Public Library, and even though it was written in 2008, it is still a very interesting and useful read. The book has two parts.

Part one is an introduction to the Base and Core parts of Dojo which are concerned with making it easy to work with JavaScript and the DOM in a browser agnostic way - string utilities, a module system for modularising your Dojo libraries, DOM utilities (along the lines of jQuery), Ajax utilities, a pub/sub framework, utilities for simulating classes and inheritance, drag and drop and animation support. What I liked about the book is that the author describes the problem before showing how Dojo solves it - it's this insight into the design that makes the book interesting. For example, he covers the differences in the interpretation of the box model in early browsers, which leads to Dojo utilities for getting data adjusted for the box model that is in effect.

Part two is a discussion of Dijit, the Dojo widget system that is designed for writing components which can be embedded into web pages. This system introduces a lifecycle, allowing the user to inject functionality into the component at well defined points, and a series of layout widgets that allow you to write flexible layouts.

It's interesting to see how these facilities have made their way into more modern browser frameworks [2]like [3]WinJS.

1. [http://www.amazon.co.uk/Dojo-Definitive-Matthew-A-Russell/dp/0596516487/ref=sr\\_1\\_2?ie=UTF8&qid=1359795637&sr=8-2](http://www.amazon.co.uk/Dojo-Definitive-Matthew-A-Russell/dp/0596516487/ref=sr_1_2?ie=UTF8&qid=1359795637&sr=8-2)
  2. <http://stephenwalther.com/archive/2012/02/22/metro-namespaces-and-modules.aspx>
  3. <http://stephenwalther.com/archive/2012/02/27/metro-using-templates.aspx>
-

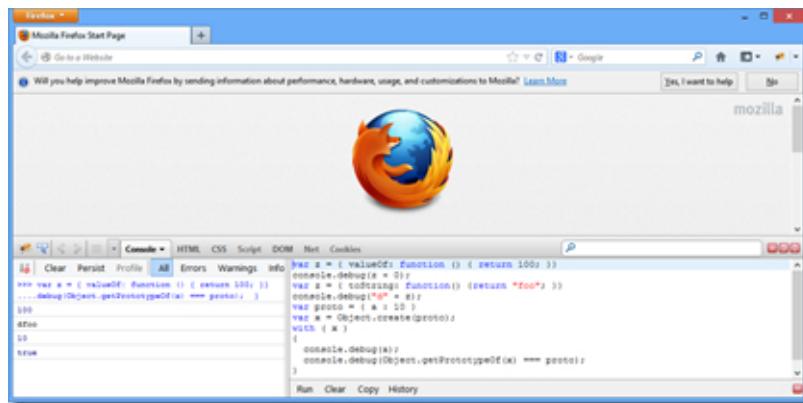
## 7.2.2 Just a few things to make it more efficient (2013-02-21 07:30)

[1]Effective JavaScript: 68 specific ways to harness the power of JavaScript by David Herman

Whenever you've programmed in a language for a while, there are always moments when you step back and want to understand how and why a piece of code works in the way you've come to expect. JavaScript is an interesting language, with a syntax that makes it look like a lot of other languages, but which really has semantics that make it very different - prototypical inheritance, scope chains that are make lookup interesting in constructs like with, integer values that are really floating point values in the runtime and variable scoping that isn't block structured.

This book is a really good read if you want to understand what is really going on. Moreover, it doesn't just cover what an ECMAScript 5 implementation would do, but covers how the implementation of a particular construct differs across browsers and across versions of JavaScript. Playing with the small snippets of example code in Firebug makes it very easy to see what is going on.

[2]



I've read a few JavaScript books, but this is the most illuminating. For example, it completely clarifies the difference between the prototype property for functions) and the prototype property of every object (Object.getPrototypeOf) in a few very well written pages. The discussion of implicit conversions is also very informative, covering material that I haven't really seen before.

As well as covering the language elements, the book also covers the design of libraries and the mechanisms for achieving concurrency, including promises and web workers. This makes it a thoroughly good way of getting up to speed for writing the mass of single page applications which are surely the future. All 68 gems are well worth a read.

1. [http://www.amazon.co.uk/Effective-JavaScript-Specific-Software-Development/dp/0321812182/ref=sr\\_1\\_1?ie=UTF8&qid=1361096247&sr=8-1](http://www.amazon.co.uk/Effective-JavaScript-Specific-Software-Development/dp/0321812182/ref=sr_1_1?ie=UTF8&qid=1361096247&sr=8-1)
2. <http://clivetong.files.wordpress.com/2013/02/firebug.png>

---

## 7.2.3 Press the meta key (2013-02-22 07:08)

[1]Metaprogramming in .NET by Kevin Hazzard and Jason Bock

Looking back at the many years I spent as a programmer using [2]Common Lisp, I now realise that one thing I loved about that language was the ability to easily do metaprogramming. It was a fairly trivial exercise to write code that generated other code, whether the need was an [3]embedded domain specific language (implemented via macros) or the generation of a test suite from a grammar definition (for an SQL interpreter that I wrote to enable external parties to query our object model).

This book has a great discussion of what metaprogramming is, and describes how you might do it on the .NET platform.

Chapter one discusses the need for metaprogramming and this is followed by chapter two which looks at the relationship between metaprogramming and Reflection. Reflection only goes so far in this dimension – it certainly allows you to choose methods at runtime and even find methods that didn't exist when your component was built. Reflection itself doesn't allow you to add methods to existing classes or even define new classes and methods though so it is a little limited.

The rest of the book looks at the various other .NET technologies for generating classes and methods, starting with techniques for generating and then compiling source code. T4 templates, which you might use to define a data access layer and which are supported at compile time inside Visual Studio, and the CodeDom are studied in some detail.

The book moves on to `Reflection.Emit` for generating new assemblies, though this involves working at the IL level when defining new methods. This can be a little limiting, though it is also powerful as you have access to IL constructs that cannot be expressed in C #, such as fault handlers. Expression trees, originally added to support LINQ, offer an easier way for defining dynamic methods, using a tree structure of language agnostic object instances to express the desired functionality.

[4]

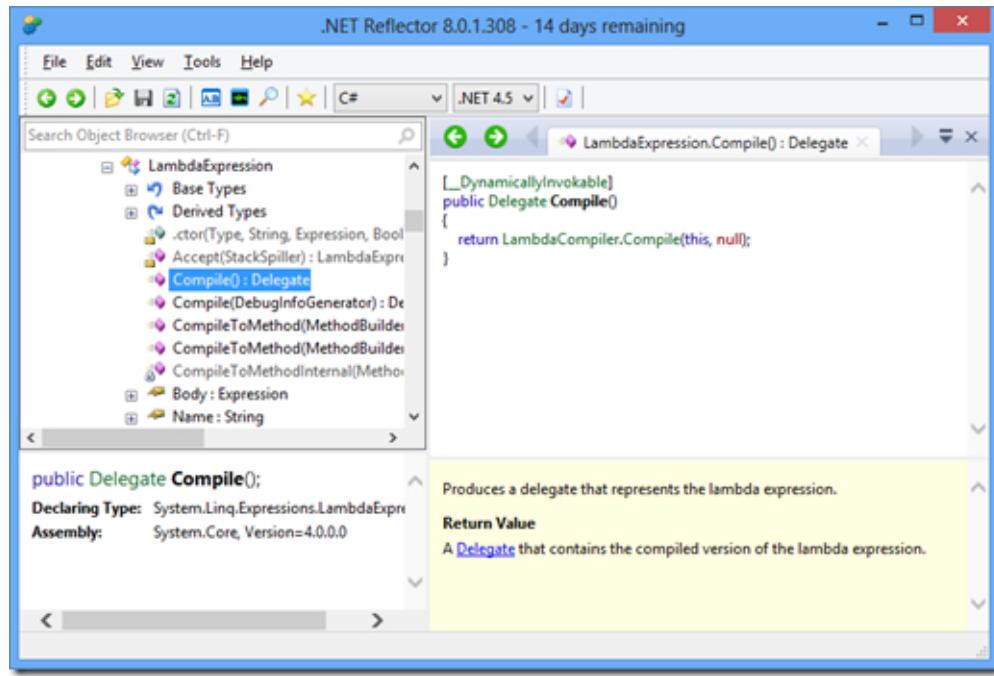
```
Expression<Func<int, int>> funcTree = x => x * 3;
var func = funcTree.Compile();
Console.WriteLine(func(10));

var method =
    typeof(Console)
        .GetMethod("WriteLine",
            System.Reflection.BindingFlags.Static | System.Reflection.BindingFlags.Public,
            null, new[] { typeof(string) }, null);

var block =
    Expression.Lambda<Func<int>>(
        Expression.Block(
            Expression.Call(method, Expression.Constant("Hello")),
            Expression.Constant(10)));
var blockCompiled = block.Compile();
Console.WriteLine(blockCompiled());
```

The `System.Core` assembly contains a compiler for such expression trees that you can access via the `System.Linq.Expressions.LambdaExpression` type with its `Compile` method which can build a delegate from an Expression tree.

[5]



The book also covers the alternative technologies of IL rewriting, using [6]Cecil and [7]Post-Sharp to produce and/or modify an existing assembly at the byte code level to add or augment classes and methods, and there is also information on the [8]DLR, a technology associated with the implementation of the dynamic type in C #, which also acted as a compilation target for [9]Ruby and [10]Python when such languages were ported to the .NET platform. The final chapter covers [11]Roslyn, the managed compiler which is currently available as a CTP. Roslyn supports compilation offering an open infrastructure that gives access to the parse trees, making it possible to easily write new refactorings inside Visual Studio, and also modify and recompile existing source code.

Metaprogramming has many practical uses and this book clearly shows how all this can be made to work on top of the .NET platform.

1. [http://www.amazon.co.uk/Metaprogramming-NET-Kevin-Hazzard/dp/1617290262/ref=sr\\_1\\_1?ie=UTF8&qid=1361101581&sr=8-1](http://www.amazon.co.uk/Metaprogramming-NET-Kevin-Hazzard/dp/1617290262/ref=sr_1_1?ie=UTF8&qid=1361101581&sr=8-1)
2. [http://en.wikipedia.org/wiki/Common\\_Lisp](http://en.wikipedia.org/wiki/Common_Lisp)
3. [http://en.wikipedia.org/wiki/Domain-specific\\_language](http://en.wikipedia.org/wiki/Domain-specific_language)
4. <http://clivetong.files.wordpress.com/2013/02/expressiontrees.png>
5. <http://clivetong.files.wordpress.com/2013/02/compiler.png>
6. <http://www.mono-project.com/Cecil>
7. <http://www.sharpcrafters.com/>
8. <http://dlr.codeplex.com/>
9. <http://ironruby.net/>
10. <http://en.wikipedia.org/wiki/IronPython>
11. <http://msdn.microsoft.com/en-us/vstudio/roslyn.aspx>

## 7.2.4 Smoke and mirrors - well, reflection anyway (2013-02-25 07:34)

[1]Goedels Proof by Ernest Nagel and James R. Newman

A short but informative book on the history, mechanics and consequences of Goedel's famous theorem.

The book starts out with the historical settings of the proof, covering [2]Hilbert's aim of converting mathematics into a symbol manipulation game and the subsequent attempt to formalise the foundations of mathematics in Russell and Whitehead's [3]Principia Mathematica. Hilbert wanted a proof of the consistency of these foundations using finite methods. The authors describe first order logic and how it can be used to encode formal theories, explaining the relationship between the symbolic inferences of the theory and the model of the underlying theory, along the way explaining what consistency means. They demonstrate this very effectively by defining a theory with two classes and some axioms, which is then shown to be consistent by interpreting the theory using the lines and vertices of a triangle – a simple but effective demonstration of the concepts at hand.

The second half of the book covers the method of Goedels proof. They spent a lot of time showing how the statements about number theory can be encoded in number theory itself, using the uniqueness of prime factorisation as the encoding method. They then show how this reflection of theorems about PA into PA itself allows the construction of a statement that asserts its own unproveability.

The book then goes on to discuss the repercussions of the incompleteness theorem, pointing out the distinction between truth and proof, and also mentioning the interesting result that [4]if you could show certain results in number theory could not be proved in PA then they would have to be true. [5]Goldbach's conjecture is often used as an example in these circumstances, but [6]this article covers some already proved examples.

1. [http://www.amazon.co.uk/Goedels-Proof-Routledge-Classics-Ernest/dp/0415355281/ref=sr\\_1\\_1?ie=UTF8&qid=1361569151&sr=8-1](http://www.amazon.co.uk/Goedels-Proof-Routledge-Classics-Ernest/dp/0415355281/ref=sr_1_1?ie=UTF8&qid=1361569151&sr=8-1)
  2. [http://en.wikipedia.org/wiki/David\\_Hilbert](http://en.wikipedia.org/wiki/David_Hilbert)
  3. [http://en.wikipedia.org/wiki/Principia\\_Mathematica](http://en.wikipedia.org/wiki/Principia_Mathematica)
  4. <http://mathoverflow.net/questions/27755/knuths-intuition-that-goldbach-might-be-unprovable>
  5. [http://en.wikipedia.org/wiki/Goldbach%27s\\_conjecture](http://en.wikipedia.org/wiki/Goldbach%27s_conjecture)
  6. <http://www.ams.org/notices/200604/fea-davis.pdf>
- 

## 7.2.5 Unbounded storage - how much? (2013-02-25 08:36)

[1]The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper On Computability And The Turing Machine by Charles Petzold

This is a very interesting book that sets the historical context of Turing's work, and then goes through his famous paper, taking the various paragraphs and explaining them. After covering the paper, the author also goes through the work that directly followed it.

These Turing machines aren't quite the same design that you see in Computer Science courses. Like all great proofs, the initial idea and proof steps got reworked, simplified and tidied by

various other contributors. It's still an interesting read though, and it's good to see what Turing's original contribution looked like.

1. [http://www.amazon.co.uk/The-Annotated-Turing-Historic-Computability/dp/0470229055/ref=sr\\_1\\_1?ie=UTF8&qid=1361616001&sr=8-1](http://www.amazon.co.uk/The-Annotated-Turing-Historic-Computability/dp/0470229055/ref=sr_1_1?ie=UTF8&qid=1361616001&sr=8-1)
- 

## 7.3 March

### 7.3.1 If the skies are grey, read something else (2013-03-04 07:01)

It's been a fairly busy week at work, and all the spare time I have I've been looking at [1]Actorfx, a framework of worker processes that manages state in the cloud. I'll be writing up my notes on that in a future blog post. It's been interesting playing around with the local Azure emulator, and getting to grips with various concepts like the fabric controller.

I did have a little time this week for reading a few posts that I've been queuing up for some time.

First some Haskell related stuff. [2]This book extract is a really good introduction to what laziness means in Haskell. It shows how you can use the interactive facilities in GHC to understand that values are often only evaluated to weak head normal form (WHNF). In this state the outermost part of the value has been calculated, but internal parts (think fields) of the datatype can remain unevaluated. In other work related to laziness and the advantages and disadvantages it brings, this [3]paper (which is part of the work covered on [4]this page) talks about the space leaks that Haskell's laziness can cause, and discusses a generator framework that can be used to get some of decoupling that generate and test offers, without some of the inherent problems mixing effects and laziness. The paper shows one of amazing properties of laziness, tying the knot, which is demonstrated effectively by the repmin function in the paper.

I've been writing some C++ at work, and enjoyed this [5]list of pitfalls. C++ is back as a strong contender in the WinRT world, and Microsoft have published an [6]interesting paper giving guidance on writing a WinRT application in C++. A post entitled "Welcome back to C++" gives some guidance on how [7]the language has moved on, and now includes automatic type inference (with auto) and easier memory management. It also includes lambda expressions allowing the WinRT code to offer an easy to use promises library.

In C # news, there's a presentation [8]here from the recent MVP summit that covers some of pitfalls of async that people are experiencing in the real world. If you've ever wondered how you might have implemented XNA, there's a codeplex project to produce a source compatible open framework called [9]ANX, which lets you see into the implementation.

On other news, the mechanical sympathy blog has [10]a great post on memory caches in modern cpus, and there's [11]another zero day Java exploit.

1. <http://actorfx.codeplex.com/>

2. [http://ofps.oreilly.com/titles/9781449335946/sec\\_par-eval.html](http://ofps.oreilly.com/titles/9781449335946/sec_par-eval.html)

3. <http://okmij.org/ftp/continuations/PPYield/yield-pp.pdf>
  4. <http://okmij.org/ftp/continuations/PPYield/index.html>
  5. <http://www.horstmann.com/cpp/pitfalls.html>
  6. <http://www.microsoft.com/en-in/download/details.aspx?id=35814>
  7. <http://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh279654.aspx>
  8. <http://blogs.msdn.com/b/pfxteam/archive/2013/02/20/mvp-summit-presentation-on-async.aspx>
  9. <http://anxframework.codeplex.com/>
  10. <http://mechanical-sympathy.blogspot.co.uk/2013/02/cpu-cache-flushing-fallacy.html>
  11. <http://blog.fireeye.com/research/2013/02/yaj0-yet-another-java-zero-day-2.html>
- 

### 7.3.2 That's not what you said earlier (2013-03-21 07:36)

[1]Mark Miller is well known for the work he has done on the security of JavaScript, including the introduction of proxies to ECMAScript 5 with the intention of making it easier to keep mash-ups secure. I have only recently got around to reading [2]his phd thesis, “Robust Composition: Towards a unified approach to access and concurrency control”. The thesis is a really good read. It introduces object capability based access control, as implemented in [3]the E language, together with discussion on how to allow objects to interact safely when third parties might be trying to exploit the situation. There is a [4]Caja project which deals with the practical use of the object capability model including revocable references via caretakers and membranes.

Some of the ideas from the thesis went into the design of proxies for ECMAScript 5, and [5]this paper is a great discussion of how proxies can be made to interact with object invariants such as JavaScript’s notion of frozen objects.

1. <http://research.google.com/pubs/author35958.html>
  2. <http://erights.org/talks/thesis/markm-thesis.pdf>
  3. <http://erights.org/elang/index.html>
  4. [http://en.wikipedia.org/wiki/Caja\\_project](http://en.wikipedia.org/wiki/Caja_project)
  5. [http://static.googleusercontent.com/external\\_content/untrusted\\_dlcp/research.google.com/en//pubs/archive/40736.pdf](http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/40736.pdf)
- 

### 7.3.3 That's a very sharp solution (2013-03-25 07:10)

[1]Web, Cloud & Mobile Solutions with F # by Daniel Mohl

I thought that this was a really excellent introduction to the practical use of F #. The author builds various kinds of applications - first a web application using MVC 4 which accesses various types of database and moves on to a cloud hosted service that uses [2]Web Api. The author covers lots of [3]NuGet packages that make such services friendlier to F #, and also covers

how we can write frontends in F # and have the JavaScript automatically generated using [4]Pit and [5]WebSharper.

The book is a really good, and fairly quick read. It teaches some of the concepts behind F #, not by the standard tutorial style, but by writing a part of the application and then explaining how the piece of code works. Using this technique, the author covers many of the main ideas behind F #, including computational workflows. This teaching style seems to work really well.

1. [http://www.amazon.co.uk/Building-Web-Cloud-Mobile-Solutions/dp/1449333761/ref=sr\\_1\\_1?ie=UTF8&qid=1364029493&sr=8-1](http://www.amazon.co.uk/Building-Web-Cloud-Mobile-Solutions/dp/1449333761/ref=sr_1_1?ie=UTF8&qid=1364029493&sr=8-1)
  2. <http://www.asp.net/web-api>
  3. <http://www.nuget.org/>
  4. <http://pitfw.posterous.com/>
  5. <http://websharper.com/home>
- 

## 7.4 April

### 7.4.1 I remember when multi-threaded was only on a single core (2013-04-08 06:16)

I spent the last couple of days watching [1]Herb Sutter's excellent presentations on the C++ 11 memory model and the atomic type that has been added to the language. C++ is an interesting language as it attempts to get you close to the hardware while at the same time trying to make things abstract enough that it is possible to write code that targets multiple platforms.

For C++ 11, the authors of the C++ specification have followed Java's lead, and defined the language to be [2]sequentially consistent for data race free programs – sequential consistency meaning that the result of running code always looks like an interleaving of the lines of the code as you'd see it running on a single thread (though this is perhaps a little ill-defined when you realise that a single line of source code may need to be mapped into a more base version of the language before compilation for the processor). Herb Sutter's talk is a great introduction to modern hardware, and the optimisations that the compiler might want to do to the code, and covers how these might interact to make the program behave in a way that is non-deterministic.

Why are memory models interesting? Most programs are going to be written using locks, which guarantee the view that multiple threads have of the lock controlled shared memory. However, there are always occasions where it feels like too much overhead to take a lock, and you just want to define a field in an object which multiple threads can use to share some information. Volatile, in C #, is the way that you can get some of the behaviour of C++ atomics, though the semantics of volatile in C # aren't always as tight as you may like (see the discussion of volatile write reordering [3]here). C++ and its relaxed atomics make it possible to see some of the trade offs, and the common patterns that can be implemented with weaker forms of memory model guarantees. To be honest though, the main reason to understand a memory model is to be able to point out to people the problems in their [4]implementation of lock free lazy initialization and make one aware of some of the danger areas when compiling ones [5]C # application for ARM architectures.

To me the whole area is just interesting as it demonstrates the trade off between the simplifications to allow fast hardware implementation against the difficulty this gives compiler writers. It will be interesting to see if hardware models continue to evolve towards sequential consistency as Herb Sutter suggests in his talk, making all of this memory model work fairly redundant in the future.

1. <http://herbsutter.com/2013/02/11/atomic-weapons-the-c-memory-model-and-modern-hardware/>
  2. <http://bartoszmilewski.com/2008/11/11/who-ordered-sequential-consistency/>
  3. <http://www.albahari.com/threading/part4.aspx>
  4. <http://msdn.microsoft.com/en-gb/magazine/jj883956.aspx>
  5. <http://msdn.microsoft.com/en-us/magazine/jj553518.aspx>
- 

#### **7.4.2 Ever onward (2013-04-10 06:54)**

Internet Explorer seems to be moving towards web standards in leaps and bounds. In the [1]recently leaked build of Windows “Blue”, the included version of IE11 seems to be about to [2]support both WebGL and SPDY, the latter at the operating system level. [3]This post summarises some of the changes. I came across it while reading [4]this interesting post on the web as a game platform (mainly by using an implementation of [5]asm.js).

1. <http://arstechnica.com/information-technology/2013/03/windows-blue-leaks-more-metro-more-multitasking/>
  2. <http://winsupersite.com/windows-8/blue-s-clues-webgl-support-ie-11>
  3. <http://fremycompany.com/BG/2013/Internet-Explorer-11-s-leaked-build-395/>
  4. <https://brendaneich.com/2013/03/the-web-is-the-game-platform/>
  5. <http://asmjs.org/>
- 

#### **7.4.3 Send the signal (2013-04-12 06:28)**

I've just finished reading the [1]free eBook on SignalR available from campusmvp.net. Apart from a couple of typos (including a piece of code that's supposed to demonstrate a five second time delay but which doesn't have any delay written into its use of TPL Tasks) I thought the book was a great introduction to SignalR. It has a number of good examples of multiple clients connected to each other via a web server – one has the browsers tracking the mouse positions of all of the others on a particular web page, and there's an example where all browsers share the view of a single drawing canvas on which they can all make edits.

The book gives a good introduction to the technology that SignalR is using under the covers – forever frame, long polling, server-sent events and websockets, and teaches how to use the JavaScript client side library by example. It covers the persistent connection basic level of SignalR and then covers the use of hubs, the higher level abstraction over connections and

groups. Great introduction to an exciting technology.

1. <http://www.campusmvp.net/signalr-ebook>

#### 7.4.4 Short and sweet (2013-04-22 06:09)

[1]Programming Entity Framework: DbContext by Julia Lerman and Rowan Miller

I'd been to a couple of talks on the Entity Framework, but I had never really used it. A colleague at work lent me this book, which focuses on the code first aspect of the Entity Framework by way of the DbContext, a type based on the original ObjectContext.

The DbContext API has really good support for Code First development – using convention over configuration it is really easy to define some plain old C # classes (POCOs), and create some tables in a database that store them.

```
public class MyTestDb : DbContext
{
    public DbSet<Person> People { get; set; }
}

public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public List<Datum> Data { get; set; }
}

public class Datum
{
    public int Id { get; set; }
    public string InfoItem { get; set; }
}
```

[2]

Making an instance of the context, we can create instances of the domain classes and add them into the context, which allows the context to track them and subsequently persist them when a call to SaveChanges is made. The book has a whole chapter in the change tracking that the context does, covering how you can determine if an object has been modified and change the tracked state, and how you can restore objects to their initially tracked values.

```
using (var db = new MyTestDb())
{
    db.People.Add(new Person { Name = "Fred" });
    db.SaveChanges();
}
```

[3]

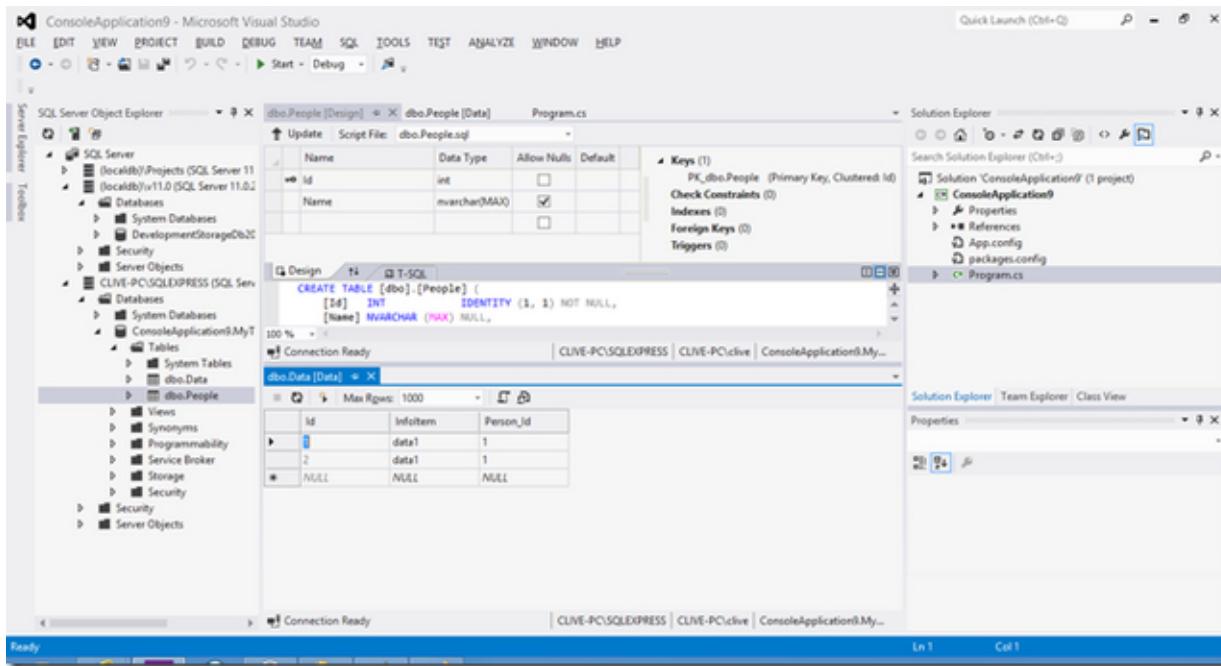
Without any configuration the Entity Framework connected to the local SqlExpress instance, and created a database and tables of the appropriate name.

[4]

```
Immediate Window
db.Database.Connection.ConnectionString
"Data Source=.\SQLEXPRESS;Initial Catalog=ConsoleApplication9.MyTestDb;Integrated Security=True;MultipleActiveResultSets=True;Application Name=Entity"
```

We can use the server object explorer in Visual Studio to look at the database changes that have been made.

[5]



The book covers the three ways that the Entity Framework handles loading of cross-table references. It can do this eagerly, on demand or lazily (assuming that the class is defined in a way that allows a proxy to be generated). The `Include` form in the following query forces the loading of the `Data` property of the class. The API allows the lambda syntax way of naming a property which is friendly to refactoring tools, though it also offers an overload that takes a string.

[6]

```
using (var db = new MyTestDb())
{
    var fred =
        (from p in db.People where p.Name == "Fred" select p)
        .Include(x => x.Data)
        .Single();
    fred.Data.Add(new Datum { InfoItem = "data1" });
    fred.Data.Add(new Datum { InfoItem = "data1" });
    db.SaveChanges();
}
```

The queries, implemented via Linq to Entities, allow you to write queries that can even include some standard SQL operators.

In order to write multiple tier applications you often need to run a query, take the objects in the result and ship them off to another tier for processing. After processing, you want to reconcile the changes made by the processing into the retrieved model so that it can be persisted back to the database. The book has an entire chapter on how you can go about this. The `DbContext` API supports Attaching and Detaching instances from the context, but also allows you to manually modify the data in the change tracking API to control whether properties are written back to the database.

The book also has a chapter on the .NET 4 `ValidationAttribute` and the `IValidateObject` interface that supports data validation. These items are explained and then there are loads of practical examples of their use.

The book is fairly short at 220 pages, but gives lots of practical information and allows you to dive straight into the Entity Framework.

1. [http://www.amazon.co.uk/Programming-Entity-Framework-Julia-Lerman/dp/1449312969/ref=sr\\_1\\_1?ie=UTF8&qid=1366447553&sr=8-1&keywords=dbcontext+julia+lerman](http://www.amazon.co.uk/Programming-Entity-Framework-Julia-Lerman/dp/1449312969/ref=sr_1_1?ie=UTF8&qid=1366447553&sr=8-1&keywords=dbcontext+julia+lerman)
  2. <http://clivetong.files.wordpress.com/2013/04/classes.png>
  3. <http://clivetong.files.wordpress.com/2013/04/creation.png>
  4. <http://clivetong.files.wordpress.com/2013/04/connection.png>
  5. <http://clivetong.files.wordpress.com/2013/04/database.png>
  6. <http://clivetong.files.wordpress.com/2013/04/navigation.png>
- 

#### 7.4.5 Follow that up (2013-04-23 06:11)

A couple of interesting blog posts from recent weeks.

Joe Duffy follows up on a recent blog post that announced a paper on the safe parallelism work he has been doing. [1]This interview with InfoQ covers some of the motivation and ideas behind the work - I liked the observation that Haskell isolates state using the IO monad, and this work has come from the other direction isolating the pure functional parts of the program.

I thought [2]this was a good description of how an ideal cipher would work.

Stephen Toub has a great [3]explanation of the Task monad and [4]this post on C # Async gotchas will teach you something. Async C # code can be confusing to reason about - at work recently there was discussion about why the following code runs synchronously, the key observation being that the code always runs synchronously until the first use of await.

```
static async Task<int> fact(int x)
{
    if (x <= 0)
        return 1;
    return x * await fact(x - 1);
}[5]
```

[6]Simon did a good write up on portable class libraries too (the secret being type forwarding and the retargetable attribute on assembly references).

1. <http://www.infoq.com/articles/Joe-Duffy-Safe-Parallelism>
  2. [http://blog.cryptographyengineering.com/2013/04/wonkery-mailbag-ideal-ciphers.html?utm\\_source=feedburner&utm\\_medium=feed&utm\\_campaign=Feed%3A+AFewThoughtsOnCryp](http://blog.cryptographyengineering.com/2013/04/wonkery-mailbag-ideal-ciphers.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+AFewThoughtsOnCryp)
  3. <http://blogs.msdn.com/b/pfxteam/archive/2013/04/03/tasks-monads-and-linq.aspx>
  4. <http://tomasp.net/blog/csharp-async-gotchas.aspx>
  5. <http://clivetong.files.wordpress.com/2013/04/async-recursive.png>
  6. <https://www.simple-talk.com/blogs/2013/04/19/inside-portable-class-libraries/>
-

## 7.5 May

### 7.5.1 Bits 'n' pieces (2013-05-13 05:47)

Last weekend was taken up travelling to run a half marathon on the South Downs, so I've dropped behind in my reading, though I've also been working through a book on computational complexity and re-reading Windows Internals and both are fairly time consuming. Chess seems to be making its way back as a lunchtime game at Red Gate towers, so I've also started working through [1]some chess problems which has also been taking up my free time.

Xamarin recently held their Evolve conference, and the keynote is available [2]here. They did promise to make the session videos available, but I don't think they are available yet (EDIT: there's now a [3]link to the first batch). Some of the sessions are really interesting. The keynote emphasises the rapid uptake in Smartphones, and a vast market for applications that is on the verge of happening. [4]Xamarin have bet their venture funding on making the [5]Mono (.NET) runtime available on iOS and Android, allowing the developer to write the main logic of their application in portable C# and offering GUI interface libraries for the target platforms which allow the developer to access the native widgets. This seems to hit the sweet spot of rapid development with C# whilst still writing applications that use the widgets and therefore look good on the base platform. The keynote points out some of the successes that they have had.

The [6]Channel 9 Defrag Tools show has some interesting episodes on [7]debugging the CLR GC using Windbg. The episodes cover the use of the SOS extension, and the commands that it offers which give you information of the state of the garbage collector and its various heaps. Along the way, the CLR GC developer, [8]Maoni Stephens, gives various interesting titbits around the design of the collector and what users should expect from it.

The [9]fourth episode of this mini-series covers [10]PerfView which can process the ETW messages that the CLR runtime logs in order to let you see if your applications is causing the GC undue grief. I thought this was really interesting. Lots of profiling around garbage collection seems to focus on memory leaks, with the main technique being the use of snapshots. You take a snapshot by doing a full garbage collection followed by dumping all of the heap objects and their relationships. You then compare the snapshots to see if objects are still being held in the live object graph when you'd expect them to have been collected. Most profilers seem to be missing the other part of the available information. How frequent are collections and how much time is the garbage collector taking? What object allocations are provoking garbage collections? Are temporary objects getting promoted into higher generations?

Innovation and whether it is happening inside Windows is covered a little by [11]this blog post. It's an interesting question whether product development should be driven by feature addition rather than feature improvement. The post by the developer on the Windows Kernel team is obviously a little over the top, but some of the underlying issues are worth some thought.

I also enjoyed this [12]introductory article on WebGL, and for the wannabe mathematician in me, this article on [13]factoring using number sieves brought back some memories. While we are down memory lane, our Lisp runtime went in to space on board the Deep Space 1 probe in the 1990s, and [14]this talk discusses why Lisp was dismissed as a programming language by NASA despite some obvious advantages.

1. [http://www.amazon.co.uk/Tactical-Chess-Puzzles-American-Puzzle/dp/080692733X/ref=sr\\_1\\_1?ie=UTF8&qid=1368353407&sr=8-1&keywords=303+tactical+chess](http://www.amazon.co.uk/Tactical-Chess-Puzzles-American-Puzzle/dp/080692733X/ref=sr_1_1?ie=UTF8&qid=1368353407&sr=8-1&keywords=303+tactical+chess)

2. <http://xamarin.com/evolve/2013>

3. <http://blog.xamarin.com/first-wave-of-cross-platform-evolve-sessions-available/>
  4. <http://xamarin.com/>
  5. [http://www.mono-project.com/Main\\_Page](http://www.mono-project.com/Main_Page)
  6. <http://channel9.msdn.com/search?term=defrag+tools>
  7. <http://channel9.msdn.com/Shows/Defrag-Tools/Defrag-Tools-33-CLR-GC-Part-1>
  8. <http://channel9.msdn.com/posts/Maoni-Stephens-CLR-45-Server-Background-GC>
  9. <http://channel9.msdn.com/Shows/Defrag-Tools/Defrag-Tools-36-CLR-GC-Part-4>
  10. <http://blogs.msdn.com/b/vancem/archive/2012/06/21/next-version-of-perfview-has-been-released.aspx>
  11. <http://blog.zorinaq.com/?e=74>
  12. <https://hacks.mozilla.org/2013/04/the-concepts-of-webgl/>
  13. <http://www.ams.org/notices/199612/pomerance.pdf>
  14. [http://www.youtube.com/watch?v=\\_gZK0tW8EhQ](http://www.youtube.com/watch?v=_gZK0tW8EhQ)
- 

### 7.5.2 Windbg and the case of the unfreed garbage (2013-05-30 06:39)

Virtual machine technology has brought many benefits. The runtime can be much more featured than a standard runtime library of a language such as C, offering memory management, exceptions and thread management at a level that is easy to program, and in a manner that is much more cross platform. There are trade-offs at work here, as there are in all implementations. In particular, the garbage collector is implemented in user mode code (as is the whole CLR) and you get a complete garbage collectible heap in every process that is running managed code. Moreover, garbage collection is driven by allocation failure (or explicit user calls to the GC.Collect methods), waiting until there is no space in a generation before scheduling a collection, and using a number of heuristics in order to decide which of the generations should be collected. These heuristics also control when memory is returned back to the operating system. Typically, the larger the heap, the more efficient the garbage collection process, so memory can be held for longer than you might naively expect.

We've all used the task manager to look at the size of managed processes. When the process is blocked, no allocations means that no garbage collections are scheduled, and hence there is nothing which is going to release unused pages back to the operating system.

Of course it would be nice if the garbage collector could respond to the OS when memory gets low. But just does it do this? We might not have reflection in the unmanaged world, but a combination of symbols from a symbol server, and other tools makes it possible to investigate.

First, I knew that you can subscribe to the OS event that is raised when memory is running low via the function [1]CreateMemoryResourceNotification. The first thing to check is whether the standard dlls that make up the CLR import this function from the OS interface libraries. Doing dumpbin on clr and some other key dlls

```
dumpbin c:\windows\Microsoft.NET\Framework \v4.0.30319\clr.dll /imports
```

listed imports for a number of libraries, but not this function.

Right, time to get dynamic. I wrote a quick console application that blocked inside Console.ReadLine() and then started it under WinDbg. Setting things up to use the Microsoft symbol server, and then using the examine command to find the symbol

```
.symfix c:\localsymbols  
x *!*CreateMemoryResource*
```

found the symbol, so I could then breakpoint it.

```
74870b4e KERNELBASE!CreateMemoryResourceNotification  
bp KERNELBASE!CreateMemoryResourceNotification
```

We hit the symbol and I stepped up. This left the stack looking like this.

```
0:000> k  
ChildEBP RetAddr  
00dbfa14 64773428 clr!WKS::gc_heap::initialize_gc+0xb0  
00dbfa1c 6475e0da clr!WKS::GCHeap::Initialize+0x2f  
00dbfa24 6476ae02 clr!ExecuteDLL+0x36d  
00dbfb9c 6475ba2f clr!EEStartupHelper+0x846  
00dbfbe4 647744fe clr!EEStartup+0x1e  
00dbfc28 64801f39 clr!EnsureEEStarted+0xea  
00dbfc68 64804162 clr!_CorExeMainInternal+0x8f  
00dbfcfa4 71fff5a3 clr!_CorExeMain+0x4d  
00dbfce0 72077f16 mscoreei!_CorExeMain+0x10a  
00dbfcf0 72074de3 MSCOREE!ShellShim__CorExeMain+0x99  
00dbfcf8 74e68543 MSCOREE!_CorExeMain_Exported+0x8  
00dbfd04 7725bf39 KERNEL32!BaseThreadInitThunk+0xe  
00dbfd48 7725bf0c ntdll!_RtlUserThreadStart+0x72  
00dbfd60 00000000 ntdll!_RtlUserThreadStart+0x1b
```

The return register contains a handle

```
0:000> r eax  
eax=00000120  
0:000> !handle 120 f  
Handle 120  
Type Event  
Attributes 0x10  
GrantedAccess 0x100001:  
Synch  
QueryState  
HandleCount 22  
PointerCount 4573373  
Name \KernelObjects\LowMemoryCondition  
Object Specific Information  
Event Type Manual Reset  
Event is Waiting
```

So the question is what waits on the handle? If we continue the application, we get to a point where there are only two managed threads running.

```
0:004> .loadby sos clr  
0:004> !Threads  
ThreadCount: 2  
UnstartedThread: 0  
BackgroundThread: 1
```

```

PendingThread: 0
DeadThread: 0
Hosted Runtime: no
Lock
ID OSID ThreadOBJ State GC Mode GC Alloc Context Domain Count Apt Exception
0 1 2210 0119d608 2a020 Preemptive 02E34188:00000000 011655b0 1 MTA
2 2 23c0 0116d458 2b220 Preemptive 00000000:00000000 011655b0 0 MTA (Finalizer)

```

The main thread is waiting inside a call to Console.ReadLine, The finalizer thread would be a potential thread that might wait on our event. After all, it spends most of its time waiting. So we can switch to it.

```

0:004> 2s
eax=00000000 ebx=64cdf388 ecx=00000000 edx=00000000 esi=02cff850 edi=00000000
eip=7722e1a4 esp=02cff728 ebp=02cff8a8 iopl=0 nv up ei pl nz na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000206
ntdll!NtWaitForMultipleObjects+0xc:
7722e1a4 c21400 ret 14h
0:002> k
ChildEBP RetAddr
02cff724 7484c752 ntdll!NtWaitForMultipleObjects+0xc
02cff8a8 64787bc5 KERNELBASE!WaitForMultipleObjectsEx+0x10b
02cff8f4 647885f6 clr!WKS::WaitForFinalizerEvent+0xbe
02cff934 646e9843 clr!WKS::GCHeap::FinalizerThreadWorker+0x6e
02cff9c4 646e98fd clr!REGUTIL::EnvGetString+0xfc
02cff9d0 646e998a clr!SHash<StringSHashTraits< _ConfigStringKeyValuePair,unsigned short,CaseSensitiveStringCompareHash<unsigned short>>::Lookup+0x11
02cffa58 647741c6 clr!EEConfig::GetConfiguration _DontUse _+0x1b0
02cffad4 647ca0c1 clr!WKS::GCHeap::FinalizerThreadStart+0x198
02cffb6c 74e68543 clr!Thread::intermediateThreadProc+0x4d
02cffb78 7725bf39 KERNEL32!BaseThreadInitThunk+0xe
02cffbbc 7725bf0c ntdll! __RtlUserThreadStart+0x72
02cffbd4 00000000 ntdll! __RtlUserThreadStart+0x1b

```

Now dump the stack to try to find parameters to the function we just returned from.

```

0:002> dd esp
02cff728 7484c752 00000003 64cdf388 00000001
02cff738 00000000 00000000 525d7525 011a6f40
02cff748 00000000 74e5002e 00000004 00000018
02cff758 40000060 00000000 7484ee9c 767f3b30
02cff768 00000000 0000020c 00000000 00000004
02cff778 64735f28 64735f28 7681301e 011b2ad0
02cff788 011b2abc 7722dc34 74841129 00000158
02cff798 00000000 74841151 525d7a75 000007d0

```

Looking at the values pointed to by the third item on the stack, it looks like a series of handles, the first of which is the handle to the notification object.

```

0:002> dd 64cdf388
64cdf388 00000120 00000158 000001b8 011a6f20
64cdf398 00000000 00000000 6475d2f4 00000005

```

So it looks like the handles are passed into the kernel wait method, and we need to see what

happens to the return value from WaitForMultipleObjectsEx. I happen to know that it returns the index of the handle that was signalled, so we need to see what happens if the value zero is returned.

```
0:002> u 64787bc5
clr!WKS::WaitForFinalizerEvent+0xbe:
64787bc5 03c6 add eax,esi
64787bc7 83e800 sub eax,0
64787bca 0f844b451d00 je clr!WKS::WaitForFinalizerEvent+0x12a (6495c11b)
64787bd0 83e802 sub eax,2
64787bd3 7589 jne clr!WKS::WaitForFinalizerEvent+0x292 (64787b5e)
64787bd5 33f6 xor esi,esi
64787bd7 c7442414fffffff mov dword ptr [esp+14h],0FFFFFFFh
64787bdf 89742418 mov dword ptr [esp+18h],esi
```

We test and branch if the result is zero to the following code.

```
0:002> u 6495c11b
clr!WKS::WaitForFinalizerEvent+0x12a:
6495c11b 8b0d18f6cd64 mov ecx,dword ptr [clr!GCHeap::FinalizerThread (64cdf618)]
6495c121 c7410801000000 mov dword ptr [ecx+8],1
6495c128 833d84f4cd6400 cmp dword ptr [clr!g _TrapReturningThreads (64cdf484)],0
6495c12f 7405 je clr!WKS::WaitForFinalizerEvent+0x145 (6495c136)
6495c131 e8d324efff call clr!Thread::RareDisablePreemptiveGC (6484e609)
6495c136 8b0d88f4cd64 mov ecx,dword ptr [clr!g _pGCHeap (64cdf488)]
6495c13c 6a02 push 2
6495c13e 8b01 mov eax,dword ptr [ecx]
6495c140 6a01 push 1
6495c142 6a00 push 0
6495c144 ff5064 call dword ptr [eax+64h]
6495c147 8b0d18f6cd64 mov ecx,dword ptr [clr!GCHeap::FinalizerThread (64cdf618)]
6495c14d c741080000000000 mov dword ptr [ecx+8],0
6495c154 8b4104 mov eax,dword ptr [ecx+4]
6495c157 a85f test al,5Fh
6495c159 7405 je clr!WKS::WaitForFinalizerEvent+0x16f (6495c160)
```

The target address is fetched by indirection, so we have to look at the relevant memory locations.

```
0:002> dd 64cdf488
64cdf488 0116c760 64ce6d70 646b1cec 00000000
0:002> dd 0116c760
0116c760 6475e0e8 abababab abababab feefeeee
0:002> dd 6475e0e8 +64h
6475e14c 6478c77f 64794b2c 64a09013 647877ac
```

And so we arrive at a function named GarbageCollect, which probably does what it says on the can.

```
0:002> u 6478c77f
clr!WKS::GCHeap::GarbageCollect:
6478c77f 55 push ebp
6478c780 8bec mov ebp,esp
```

```
6478c782 51 push ecx
6478c783 53 push ebx
6478c784 56 push esi
6478c785 8b7508 mov esi,dword ptr [ebp+8]
6478c788 57 push edi
6478c789 894dfc mov dword ptr [ebp-4],ecx
```

In summary then, low memory notifications cause our process to do a garbage collection, presumably with the intention of releasing memory back to the system. Though we are missing metadata, as long as we don't need to go too deep into the code of an unmanaged method, it is possible to figure out what is going on.

1. [http://msdn.microsoft.com/en-gb/library/windows/desktop/aa366541\(v=vs.85\).aspx](http://msdn.microsoft.com/en-gb/library/windows/desktop/aa366541(v=vs.85).aspx)

---

GC performing full blocking Gen2 collections under SustainedLowLatency despite there is no LowMemory notification from OS - Jan Krivanek - Site Home - MSDN Blogs (2013-12-09 20:54:05)

[...] to CLR via C # book (and you can verify that through debugger like demonstrated in this post) GC subscribes to low memory notifications through usage of CreateMemoryResourceNotification [...]

Alois Kraus (2015-03-08 21:33:56)

This was really a great investigation. I have found that when my application is really slow it is due to constant full GCs coming from the Finalizer thread. Now the question remains when this notification is fired to prevent getting into this state.

GC performing full blocking Gen2 collections under SustainedLowLatency despite there is no LowMemory notification from OS – JanKrivanek (2018-05-14 08:43:49)

[...] to CLR via C # book (and you can verify that through debugger like demonstrated in this post) GC subscribes to low memory notifications through usage of CreateMemoryResourceNotification [...]

### 7.5.3 MLWorks-yes it does! (2013-05-31 06:40)

In 1989 I started my first full time programming job at a software company called [1]Harlequin, based near Cambridge, UK. I'd had summer jobs and an industrial year before this, working for [2]ICL in Manchester and had heard some really good things about a small software house of really smart people who were doing some great work on compilers and programming environments for the [3]Common Lisp language. So I applied. After an interview, where the boss gave me and my bike a lift home, they offered me the job.

During my first year there, I first worked on the Common Lisp system, improving the performance of the system on various numerical benchmarks and adding some missing parts to the runtime system. After some time, I was offered the chance to work on an [4]ML compiler that was being written as a contract job for [5]RSRE. There were three of us working on this – the runtime work was done by [6]Colin who did the parser and type checker, while I wrote the lexer and the code generator. The target language was [7]Ten15, a high level virtual machine that ran on the PERQ computer, and it was this technology that got me interested in virtual machines.

I left a year or so later to start a Phd at Cambridge, and when I returned to Harlequin the ML compiler was being rewritten to target SPARC, as the MLWorks project. I joined this team for a while, helping to finish off the runtime system and writing an initial sampling profiler. I later started to add CORBA support to the system. I left to join another team when offered the chance to port the [8]LispWorks system to the DEC Alpha, which was fantastic opportunity to get deep knowledge about a platform (and code generators and link loaders and shared libraries and memory management and cross platform compilers).

The MLWorks system was developed over the years by various people and it is nice to see that the code has just been [9]released as open source. I'm not quite sure how relevant the technology is now – this was in the days of single threaded runtimes and machines that were considerably slower than today's, but it will be nice seeing the system build and get some use. I'll just be digging through the change logs to see what code changes I made in those days.

1. [http://en.wikipedia.org/wiki/Harlequin\\_\(software\\_company\)](http://en.wikipedia.org/wiki/Harlequin_(software_company))
  2. [http://en.wikipedia.org/wiki/International\\_Computers\\_Limited](http://en.wikipedia.org/wiki/International_Computers_Limited)
  3. [http://en.wikipedia.org/wiki/Common\\_Lisp](http://en.wikipedia.org/wiki/Common_Lisp)
  4. [http://en.wikipedia.org/wiki/Opaque\\_ascription](http://en.wikipedia.org/wiki/Opaque_ascription)
  5. <http://en.wikipedia.org/wiki/RSRE>
  6. <http://www.linkedin.com/pub/colin-meldrum/1/127/966>
  7. <http://en.wikipedia.org/wiki/Ten15>
  8. <http://en.wikipedia.org/wiki/LispWorks>
  9. <https://github.com/Ravenbrook/mlworks>
- 

## 7.6 June

### 7.6.1 Stop darting about (2013-06-03 06:40)

The usual mass of interesting of talks from Google I/O. The V8 guys have done an amazing job at improving the performance of JavaScript over the years - [1]this talk discusses the ideas that they recycled in order to make the massive leaps in performance. [2]This case study discusses how the performance of a browser based application was improved using the logging that V8 provides – the material covered brings up a number of interesting issues (that I won't reveal here). It's hard to know if we've reaching the limits of optimisation for JavaScript, and whether we need to move to a language with semantics that make it easier to deal with. Some of DART's benefits are covered [3]here. Alternatively, we could just demand that runtimes optimise certain specific subsets of JavaScript.

There are also some good talks on optimising the performance of rendering - [4]Jank free rendering in Chrome, [5]Android graphics performance and [6]web page design with the GPU in mind. All have the theme that we all need to understand more about how the GPU is being used to compose graphics layers.

1. [http://www.youtube.com/watch?v=huawCRlo9H4&list=WLvHnZeCkR\\_dSLUb8X4ACDWygfcuqyI4R1](http://www.youtube.com/watch?v=huawCRlo9H4&list=WLvHnZeCkR_dSLUb8X4ACDWygfcuqyI4R1)
2. [http://www.youtube.com/watch?v=VhpdsjBUS3g&list=WLvHnZeCkR\\_dSLUb8X4ACDWygfcuqyI4R1](http://www.youtube.com/watch?v=VhpdsjBUS3g&list=WLvHnZeCkR_dSLUb8X4ACDWygfcuqyI4R1)

3. [http://www.youtube.com/watch?v=euCNWhs7ivQ&list=WLvHnZeCkR\\_dSLUb8X4ACDWygfcuqyI4R1](http://www.youtube.com/watch?v=euCNWhs7ivQ&list=WLvHnZeCkR_dSLUb8X4ACDWygfcuqyI4R1)
  4. [http://www.youtube.com/watch?v=n8ep4leoN9A&list=WLvHnZeCkR\\_dSLUb8X4ACDWygfcuqyI4R1](http://www.youtube.com/watch?v=n8ep4leoN9A&list=WLvHnZeCkR_dSLUb8X4ACDWygfcuqyI4R1)
  5. [http://www.youtube.com/watch?v=vQZFaec9NpA&list=WLvHnZeCkR\\_dSLUb8X4ACDWygfcuqyI4R1](http://www.youtube.com/watch?v=vQZFaec9NpA&list=WLvHnZeCkR_dSLUb8X4ACDWygfcuqyI4R1)
  6. [http://www.youtube.com/watch?v=8uAYE5G1gSs&list=WLvHnZeCkR\\_dSLUb8X4ACDWygfcuqyI4R1](http://www.youtube.com/watch?v=8uAYE5G1gSs&list=WLvHnZeCkR_dSLUb8X4ACDWygfcuqyI4R1)
- 

### 7.6.2 That's too complex for me (2013-06-05 05:57)

[1]P, NP and NP-Completeness: The basis of Computational Complexity by Oded Goldreich

I wanted a book that would take me through computational complexity in a fairly detailed mathematical way while still giving more of an idea of the motivation behind the various concepts. This book does just that! It is designed for teaching and at the start of each of the five chapters, there is a good overview discussing how the material can be taught, the motivating examples and what parts of the chapter could be left for a second reading. There's an appendix at the end which gives a brief history of complexity theory, and the contributions of the three early discoverers – Cook, Karp and Levin.

At college I sat through the usual lectures on this subject, with its introduction of Turing machines, the demonstration that they have equivalent computational power to Register machines, and then the introduction of non-deterministic Turing machines and the class NP. This book very nearly leaves out non-deterministic Turing machines – it introduces the difference between verification (the class P) and proving (the class NP) via proof systems, and relates these two classes to the more modern search formulations via the PC and PF classes. I'd not realised before that while Cook was formulating P/NP via decision problems, Levin in the East was coming at the same ideas by way of search problems (which are the kind of problems you find more often day to day).

The book is brilliant. It discusses why polynomial time problems are considered tractable – most problems can be mapped between the different models of computation (say between one tape and two tape Turing machines) in a polynomial way, the philosophical implications of P and NP being the same, and makes a big noise about the existence of NP-Complete problems. I remember, when we studied this at university, being completely amazed that such a class exists. There is a lot of other material – the existence of problems in NP that aren't NP-complete (assuming that P is not equal to NP as expected), and some discussion of promise problems and zero knowledge proofs. There are illuminating proofs of some NP-complete problems from first principles (using the target object to simulate a Turing machine), as well as examples of reducing one problem to another.

A fascinating book full of good discussion as well as a rigorous mathematical treatment of the subject area.

1. [http://www.amazon.co.uk/NP-NP-Completeness-Basics-Computational-Complexity/dp/0521122546/ref=sr\\_1\\_1?ie=UTF8&qid=1369846147&sr=8-1&keywords=P+NP+and+Np+completen](http://www.amazon.co.uk/NP-NP-Completeness-Basics-Computational-Complexity/dp/0521122546/ref=sr_1_1?ie=UTF8&qid=1369846147&sr=8-1&keywords=P+NP+and+Np+completen)
-

### 7.6.3 No change here (2013-06-06 05:11)

The BCL immutable [1]collections were released on [2]Nuget some time ago, but I've only just got to play with them. There has recently been [3]an accompanying Channel 9 interview discussing the implementation.

So called [4]persistent collections have been the main data structure in the [5]Clojure language for quite some time. Clojure emphasis functional paradigms, and its implementation of persistent datatypes offer good performance while keeping the benefits of no mutation – friendliness to concurrency and removal of the need for compensation actions if an exception is thrown when state is being modified.

Persistent trees seem to have become popular at Microsoft because of the Roslyn project. This project is a rewrite of the C # compiler in C #, and it is in this scenario that there is a need to efficiently modify large abstract syntax trees without copying, a scenario that perfectly matches the persistent types.

It's easy to add the immutable collections to your project.

```
Install-Package Microsoft.Bcl.Immutable -Pre
```

And then you can start making instances using the slightly strange pattern.

```
var x = ImmutableList.Create<int>();
```

The pattern follows the example of the Tuple type in .NET 4. C # cannot infer the instantiation required for the type parameters of a generic class at the class level, but it can do so for method level type parameters. This syntax avoids the need to specify the type parameters in examples like:

```
var x2 = ImmutableList.Create(1, 2, 3);
```

Using Reflector it is interesting to look at the implementation of the type itself which is based on a binary tree. Such an implementation makes it possible to frequently share most of the tree except nodes along the spine that is modified by an operation.

There are a few optimisations in the implementation. Immutable trees can be converted to mutable trees, which can implement some operations more efficiently. These can then be converted back into immutable trees, as in the following example.

```
var x = ImmutableList.Create<int>();
var y = x.AddRange(new [] {1,2,3,4 });
var z = y.ToBuilder();
z.Add(10);
var z2 = z.ToImmutable();
```

The idea is that we can locally make things mutable, in order to improve efficiency, and then convert back to immutable before passing the value out of the local context. This kind of idea is used in F #, where lots of implementation is locally mutable, but non-locally things look very functional and are therefore easier to reason about.

The implementation of Sort, as shown by Reflector, pushes the contents into an Array, sorts the array and then reproduces an immutable tree.

```
internal ImmutableList<T>.Node Sort(int index, int count, IComparer<T> comparer)
{
    Requires.Range(index >= 0, "index", null);
    Requires.Range(count >= 0, "count", null);
    Requires.Argument((index + count) <= this.Count);
```

```

Requires.NotNull<IComparer<T>>(comparer, "comparer");
T[] array = new T[this.Count];
this.CopyTo(array);
Array.Sort<T>(array, index, count, comparer);
return ImmutableList<T>.Node.NodeTreeFromList(array.AsOrderedCollection<T>(), 0,
this.Count);
}

```

In contrast, the Reverse operation, uses the representation,  
internal ImmutableList<T>.Node Reverse(int index, int count)

```

{
Requires.Range(index >= 0, "index", null);
Requires.Range(count >= 0, "count", null);
Requires.Range((index + count) <= this.Count, "index", null);
int num = index;
for (int i = (index + count) - 1; num < i; i--)
{
T local = node[num];
T local2 = node[i];
node = node.ReplaceAt(i, local).ReplaceAt(num, local2);
num++;
}
return node;
}

```

relying on the fact that ReplaceAt uses a Mutate operation which can work slightly more efficiently if the tree is mutable (ie the frozen property is set to false). This frozen property requires that the whole tree is traversed in order to set the frozen property of each node in it.

```

private ImmutableList<T>.Node Mutate(ImmutableList<T>.Node left = null, ImmutableList<T>.Node right = null)
{
if (this.frozen)
{
return new ImmutableList<T>.Node(this.key, left ?? this.left, right ?? this.right, false);
}
if (left != null)
{
this.left = left;
}
if (right != null)
{
this.right = right;
}
this.height = 1 + Math.Max(this.left.height, this.right.height);
this.count = (1 + this.left.count) + this.right.count;
return (ImmutableList<T>.Node) this;
}

```

To see how the tree gets rebalanced, look at the RotateLeft and RotateRight on the Node inner class in the ImmutableList<T> class.

The assembly also contains an immutable stack, which has Peek and Pop operations, and a pair of such stacks are used to implement immutable queues in the classic way (We push enqueued items on to one stack, and dequeue elements by popping – when there are no more elements to pop, we take the push stack and reverse it and then construct a new pop stack from the resulting elements). There are also DebuggerProxy implementations which allow Visual Studio's visualizers to present a clearer view of the datatypes when debugging. They also throw in Dictionary, Set and Sorted Dictionaries giving a very useful set of datatypes which I'm dying to use for real.

1. <http://blogs.msdn.com/b/bclteam/archive/2012/12/18/preview-of-immutable-collections-released-on-nuget.aspx>
  2. <http://www.nuget.org/>
  3. <http://channel9.msdn.com/Shows/Going+Deep/Immo-Landwerth-and-Andrew-Arnott-Inside-Immutable-Collections>
  4. [http://en.wikipedia.org/wiki/Persistent\\_data\\_structure](http://en.wikipedia.org/wiki/Persistent_data_structure)
  5. <http://en.wikipedia.org/wiki/Clojure>
- 

#### 7.6.4 Where I come from (2013-06-10 05:33)

Before [1]Eric Lippert cancelled his blog series on tail calls in .NET, which started [2]with this post on the 3rd June, the content of the post really brought home one very important point about call stacks - they show you where the code will go next, and don't necessarily show you where you came from.

Just prior to this post, I'd been debugging a problem at work where all we had was a call stack that went through some third party code, and I'd been using [3]Reflector to try to figure out the flow of the code. It was clear that there were some jumps in the stack trace, with the trace missing certain calls. In the past, I'd been led to believe that the C # compiler never outputs tail calls, and that any tail optimisation was the work of the 64 JIT compiler, which sometimes decides to emit a tail call in place of a standard call in release mode (whereas the 32 bit version of the JIT never does this). Eric's original post confirmed this behaviour.

Of course, IL does have a tail call prefix that can be placed on call instructions, as a means of requesting that the JIT compiler does its best to generate a tail call, and this is indeed used by the F # compiler to get the kinds of stack space guarantees that users expect of a functional language – though the F # compiler also optimises some self tail calls into loops in the front end of the compiler so that it doesn't need to rely on the JIT to carry out this optimisation.

You can see the difference in behaviour when running a program like the following:

```
class Program
{
    static void Main(string[] args)
    {
        Doit(0);
    }

    static void Doit(int x)
    {
```

```
Console.WriteLine(x);
Doit(x + 1);
}
}
```

Running as Release/x86

171790

Process is terminated due to StackOverflowException.

Running as Release/x64 it keeps going. If you disassembly the code, you can see the jmp

```
0:003> .loadby sos clr
```

```
0:003> !Name2EE ConsoleApplication18.exe ConsoleApplication18.Program
```

...

```
MethodTable: 000007fccfe23940
```

```
0:003> !DumpMT -md 000007fccfe23940
```

...

```
000007fccfff4d50 000007fccfe23928 JIT ConsoleApplication18.Program.Doit(Int32)
```

```
0:003> !DumpMD 000007fccfe23928
```

...

```
CodeAddr: 000007fccfff4d50
```

```
0:003> u 000007fccfff4d50
```

```
000007fc'cff4d50 53 push rbx
```

```
000007fc'cff4d51 4883ec20 sub rsp,20h
```

```
000007fc'cff4d55 8bd9 mov ebx,ecx
```

```
000007fc'cff4d57 baae000000 mov edx,0AEh
```

...

```
000007fc'cff4dad ffc3 inc ebx
```

```
000007fc'cff4daf ebbf jmp 000007fc'cff4d70
```

This blog series looked like it was going to be informative and interesting and its a shame that Eric has cancelled it.

1. <https://twitter.com/ericlippert>
2. <http://ericlippert.com/2013/06/03/tail-recursion-part-one/>
3. <http://www.reflector.net/>

---

## 7.6.5 Stunned-I could hardly get going again (2013-06-13 05:23)

[1]Stunning CSS3: Project-based Guide to the Latest in CSS by Zoe Mickley Gillenwater

I keep hearing about CSS 3 and HTML5, which seem to be offering a load of features aimed at making web pages more and more interactive – canvas for drawing, better layout with CSS and richer UIs with animations and transitions implemented in a declarative way without the need for programming in JavaScript. This book comes at CSS3 from the point of view of the designer, illustrating the new features by means of a series of worked examples. I must admit

that I didn't actually work through the examples, but read the book as a means of getting familiar with the new CSS features.

Chapter one is a great introduction to the state of the world, clearly describing vendor prefixes, progressive enhancement and the state of browser support for CSS 3, with additional discussion of how to structure your CSS file to avoid slowing the browser down.

The next chapters consider the graphical effects that CSS 3 enables, including things such as rounded corners, and some tricks such as generating triangles using the borders of a zero size content div (see [2]here for an example of what this can achieve). Along with the examples, the text describes the steps that can be taken to get the same effect (when possible) on down-level browsers, often by using a JavaScript library to polyfill the missing functionality. We see examples of border-radius, transparent backgrounds, web fonts, text shadows and gradients, all of which are demonstrated by progressively enhancing a rather basic start page. There is also information on the new attribute selectors of CSS3, and some use of the new pseudo classes (for example nth-child or nth-child-of-type, and target for the ID matching a fragment identifier) and pseudo elements (first-line, first-letter, before and after). I particularly liked the sections on transitions and animations, which can declaratively add some spectacular effects to a page. There is a good demo page for animations [3]here and the specification [4]here.

Chapter 6 discusses different screen size, illustrating the use of media queries to target CSS at particular types of device, and the final chapter discusses the new layouts of CSS 3 including the flexible box model.

I thought that reading this book was a really good way to catch up with the new features of CSS 3, and get a clear idea of how these features could be used for real. The examples are good and thorough (even though I didn't download them and actually do the exercises) and the book is really well written. Brilliant!

1. [http://www.amazon.co.uk/Stunning-CSS3-Project-Based-Latest-Voices/dp/0321722132/ref=sr\\_1\\_1?ie=UTF8&qid=1370980256&sr=8-1&keywords=stunning+css3+a+project-based](http://www.amazon.co.uk/Stunning-CSS3-Project-Based-Latest-Voices/dp/0321722132/ref=sr_1_1?ie=UTF8&qid=1370980256&sr=8-1&keywords=stunning+css3+a+project-based)
  2. <http://queue.acm.org/detail.cfm?id=2436698>
  3. <http://css3.brashawenterprises.com/cfimg/#cfimg1>
  4. <http://www.w3.org/TR/css3-animations/>
- 

## 7.6.6 Here's your script (2013-06-20 05:46)

There's a new version of TypeScript, version 0.9 - the specification can be found [1]here. The timing is interesting, coming just after [2]Google I/O where the Dart team suggested that JavaScript is getting to the point where it can't be optimised much more, and after [3]this blog post which discusses the fork in the way forward. Some of the team from Microsoft discuss these claims in [4]the Channel 9 video as well as talking about the changes to the language between 0.8 and 0.9. The main change is generics, which are present at compile time and do not appear at all in the final runtime in the emitted JavaScript.

[5]The Playground is a good place to test the ideas out, and after a little playing around it does appear that the typing system (which uses structural rather than nominal typing) leads to a

few weird effects (as in the ex2 line below), but which is definitely the right choice when you need to handle partially typed source code.

[6]

```
1 interface Pair<T, U> {
2     getFirst(): T;
3     getSecond(): U;
4 }
5
6 class Point implements Pair<number, number> {
7     constructor(public x: number, public y: number) {}
8
9     getFirst(): T {
10         return this.x;
11     }
12
13     getSecond(): U {
14         return this.y;
15     }
16 }
17
18 function foo(pt: Point) {
19 }
20 var ex1 = foo(new Point(2, 3));
21
22 var ex2 =
23     foo(
24         {x: 1, y: 2,
25          getFirst: function() { return 1; },
26          getSecond: function() { return 2; }});
27
```

```
1 var Point = (function () {
2     function Point(x, y) {
3         this.x = x;
4         this.y = y;
5     }
6     Point.prototype.getFirst = function () {
7         return this.x;
8     };
9
10    Point.prototype.getSecond = function () {
11        return this.y;
12    };
13    return Point;
14 })();
15
16 function foo(pt) {
17 }
18
19 var ex1 = foo(new Point(2, 3));
20
21 var ex2 = foo({
22     x: 1,
23     y: 2,
24     getFirst: function () {
25         return 1;
26     },
27     getSecond: function () {
28         return 2;
29     }
30});
```

Privacy Statement | Terms of Use | Trademarks © 2012, 2013 Microsoft  
The code you enter in the TypeScript playground runs entirely in your browser and is not sent to Microsoft.

1. <http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf>
2. <https://developers.google.com/events/io/>
3. <http://blog.tojicode.com/2013/06/a-tale-of-two-web-technologies.html?showComment=1371328826902&m=1>
4. <http://channel9.msdn.com/Blogs/Charles/Anders-Hejlsberg-Steve-Lucco-and-Luke-Hoban-Inside-TypeScript-09>
5. <http://www.typescriptlang.org/Playground/>
6. <http://clivetong.files.wordpress.com/2013/06/typescript.png>

## 7.7 July

### 7.7.1 Time for a rest (2013-07-12 05:12)

It'll soon be time for my six week sabbatical, so I've decided to start reading [1]Iain M. Banks' Culture series. At the same time, I've been trying to prepare for time off by getting up to date with Windows Store application development and also improve my knowledge of C++ - several book reviews to come in the near future.

At work I've been writing lots of C# and C++ while working on unmanaged memory profiling support for [2]ANTS Memory Profiler. [3]Red Gate offers many opportunities for self improvement, with regular two-weekly lightning talks and a regular [4]Clojure lunch where a few of us

get together and try to write some Clojure code. We have an hour a week, and the current idea is to write a PEG based parser for a functional language, and then write a type checker that type checks the resulting abstract syntax tree using constraint solving. The work we have done so far can be found here - [5]ClojureLunch.

[6]//build happened at the end of last month and I've only just started working my way through some of the talks.

Anders Hejlsberg does a great presentation on [7]TypeScript, working inside Visual Studio with a TypeScript language extension and showing how TypeScript fixes a number of common problems with JavaScript. Typescript does this while still keeping a base language that translates into straightforward JavaScript.

The Windows Runtime is based on a more metadata rich reworking of COM, and in this talk the [8]Windows Runtime threading model is explained. Old style COM used to require GUI components resided in an STA (single threaded apartment), and this has been reworked as an ASTA (Application STA) which stop the re-entrancy of the old model. [9]InfoQ has a couple of articles that cover this talk in these [10]two [11]parts.

Chris Anderson discusses the ways you might build a GUI application on the three technologies of WinRT in this talk: [12]Building a UI on the Windows Runtime. It has a good discussion of the pros and cons.

This fantastic blog post has recently been mentioned in a lot of places. It discusses [13]why GC isn't suitable for mobile apps and offers a lot of hard evidence to back up its claims. It is well worth a read as it explains how low powered some mobile devices are.

1. [https://en.wikipedia.org/wiki/Culture\\_series](https://en.wikipedia.org/wiki/Culture_series)
2. <http://www.red-gate.com/products/dotnet-development/ants-memory-profiler/>
3. <http://www.red-gate.com/>
4. <http://clojuredocs.org/>
5. <http://www.github.com/red-gate/clojurelunch>
6. <http://www.buildwindows.com/>
7. <http://channel9.msdn.com/Events/Build/2013/3-314>
8. <http://channel9.msdn.com/Events/Build/2013/4-107>
9. <http://www.infoq.com/>
10. <http://www.infoq.com/news/2013/07/WinRT-Threading-1>
11. <http://www.infoq.com/news/2013/07/WinRT-Threading-2>
12. <http://channel9.msdn.com/Events/Build/2013/2-192>
13. <http://sealedabstract.com/rants/why-mobile-web-apps-are-slow/>

---

### 7.7.2 Where worlds collide (2013-07-15 05:38)

At work, we came across one of those interesting situations where two different abstractions look at the same type in different ways, making things fine if you view the type through one abstraction, but confusing it you know about both.

There was a piece of code that started a task that threw an exception – obviously it was more complicated than this, but it gives the idea:

```
Task<int> t = Task.Run<int>((Func<int>)delegate { throw new ArgumentException(); });
```

And a piece of code that started this and caught the exception.

```
try
{
    var a = t.Result;
}
catch (AggregateException)
{
    // ...
}
```

Note that we need to catch AggregateException. The [1]TPL offers some operations on tasks that can raise multiple exceptions. AggregateException is a wrapper that allows multiple exceptions to be returned.

Now using the modern [2]C # 5 async abstraction, we might change the code to

```
var a = await t;
```

The whole async mechanism is built on top of tasks, but it tries to keep the manipulation of Tasks out of the picture. It would therefore be confusing to the users if they needed to catch AggregateException around any calls to await. Hence the TaskAwaiter, which comes into the picture because

```
await t
is essentially translated to
t.GetAwaiter().GetResult()
looks at the Task and takes only the first exception.
```

The result is that the catch suddenly starts failing to catch the exception. The problem is that if you know about the TPL, you know about AggregateException and expect it, but if you only understand async, you know tasks are involved as you need to use them in the return types of the methods, but probably don't expect your exceptions to get wrapped. There's a good write up of the rational behind the decision [3]here.

While we are on the subject, Stephen Toub did another brilliant presentation on Async at //build which can be viewed [4]here.

1. <http://msdn.microsoft.com/en-us/library/dd460717.aspx>
  2. <http://msdn.microsoft.com/en-US/library/vstudio/hh156513.aspx>
  3. <http://blogs.msdn.com/b/pfxteam/archive/2011/09/28/10217876.aspx>
  4. <http://channel9.msdn.com/Events/Build/2013/3-301>
- 

### 7.7.3 Go...go...go... but only at the right time (2013-07-24 05:40)

Rich Hickey has gone and done it again! He's borrowed some ideas from other languages, reorganised them and then added them as language constructs into [1]Clojure. In this case the [2]new experimental core.async library for Clojure.

Clojure has a number of interesting constructs for dealing with concurrency. These include Agents (based on Actors) and software transactional memory, both of which can interact with the non-functional part of the language which uses state containers called Atoms and Refs. This new library uses channels, an idea which dates back at least as far as [3]Hoare's CSP, and inversion of control ideas, which can be found in C#'s async method code, in order to allow the user to write linear code which doesn't lock a thread unless it is making forward progress.

At the simple level, user code allocates a channel, and then issues blocking reads and writes on this channel using the >!! and <!! functions. In the following code we create a channel, push a value to it using future to offload the write to another thread, and then read the value in the third expression. This returns the value 11.

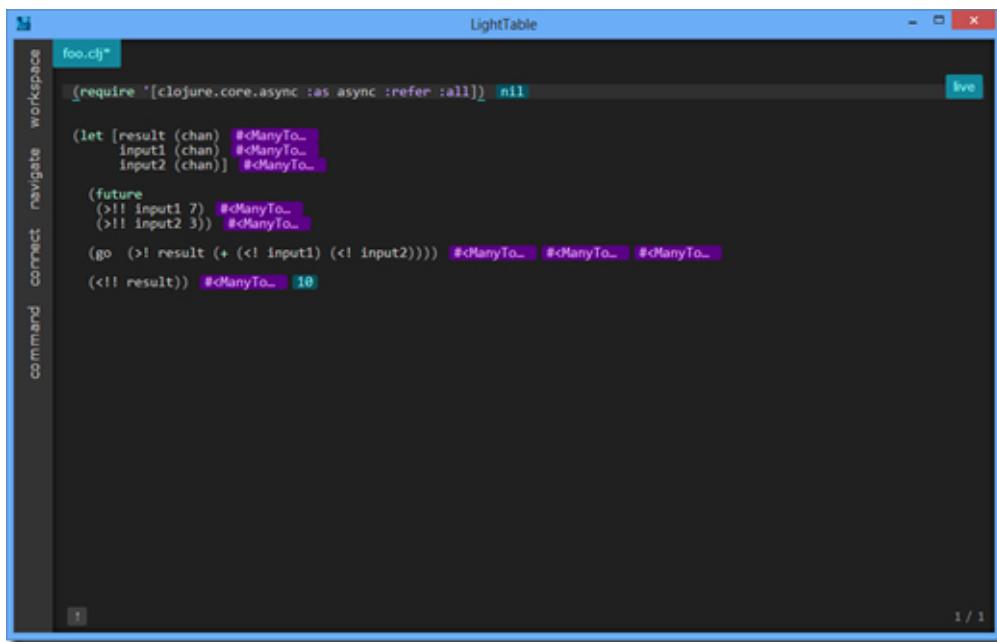
```
(def c (chan))
(future (>!! c 11))
(<!! c)
```

Channels come in a number of varieties. By default they can only have a single element in transit between writer and reader, but the library offers buffered and sliding window channels too.

The library goes further though. In the above example we were blocking the current thread between the point that we asked for a value and the point at which the background thread wrote the value to the channel. In order to use our threads more efficiently, we'd like to return a thread to the thread pool until a value is available and there is more work to be done.

Consider the following example, which shows a session running in [4]LightTable. LightTable allows you to run an interactive REPL that displays the values of expressions that appear in the session, but, best of all, it takes care of setting up a Clojure environment with the required project already loaded using [5]leinigen. In order to play with core.async I simply had to [6]clone the Git repository and then load the examples file into LightTable. Running a REPL on the buffer containing the file caused LightTable to load the relevant project and all of its dependencies. The purple highlights are added by LightTable so that you see values as functions within your application are called.

[7]



The screenshot shows the LightTable application window. On the left, there is a sidebar with tabs for 'workspace', 'navigate', 'connect', and 'command'. The main area is a code editor with a dark theme. A buffer titled 'foo.clj' is open, containing the following Clojure code:

```
(require '[clojure.core.async :as async :refer :all]) nil

(let [result (chan) #<ManyTo...
      input1 (chan) #<ManyTo...
      input2 (chan)] #<ManyTo...

  (future
    (>! input1 7) #<ManyTo...
    (>! input2 3)) #<ManyTo...

  (go (>! result (+ (<! input1) (<! input2)))) #<ManyTo... #<ManyTo... #<ManyTo...
  (<!! result)) #<ManyTo... 10
```

The example has a Go block. These can be used to set up a context where a thread is only borrowed from the thread pool while the contained code moves from a blocking state (waiting to read from the channel) to the next blocking state (<!, >! or alts! where the latter waits on a number of channels and returns the first value from a channel that offers a value). Clojure has macros, which make it very easy to write transformations on the abstract syntax tree and Go blocks are implemented by such a macro in macros.clj. The actual transformation is implemented by converting the contained code into SSA form and then generating a state machine between the various blocking points.

The Go block actually returns a channel which will contain the result of the block. In the code above, which I simplified from other code, we do not use the channel that is returned.

Using macroexpand we can see the result of the transformation on the Go block in the example, which I have included below. I have tried to comment the code to show the purpose of the various sections, and have removed some of the namespace information and reordered the case expression.

```
(let* [c __6067 __auto __ (chan 1)
captured-bindings __6068 __auto __ (getThreadBindingFrame)]
;; We execute the following state machine function
(dispatch/run
(fn
[]
(let [f __6069 __auto __
(fn
state-machine __5938 __auto __

;; Initialize the state machine.
;; The state is stored as an array
;; of six elements.
;; The state machine function goes in slot 0.
;; The state in slot 1. The value from the last read in slot 2. Thread bindings in slots 3.
;; (see ioc_helpers.clj for other slot offsets)
([])
(ioc-macros/aset-all!
(java.util.concurrent.atomic.AtomicReferenceArray. 6)
0 state-machine __5938 __auto __ 1 1))

([state _6908]
(let
[old-frame __5939 __auto __ (getThreadBindingFrame)]
(try
;; Reset the thread bindings
(resetThreadBindingFrame (ioc-macros/aget-object state _6908 3))
(loop
[]
(let
[result __5940 __auto __
;; Get the current state, and carry out a state transition.
(case
(int (ioc-macros/aget-object state _6908 1))
;; Initial state which waits for input on channel bound to input1,
```

```

;; and will use state 3 on restart
1 (let
[state _6908 state _6908]
(ioc-macros/take! state _6908 3 input1))
;; Get the return value out of slot 2 in the state array - this is
;; used to pass in the next value. Store that in slot 5.
;; Read from the input2 channel, setting state 4 as the return state.
3 (let
[inst _6901 (ioc-macros/aget-object state _6908 2)
state _6908 (ioc-macros/aset-all! state _6908 5 inst _6901)]
(ioc-macros/take! state _6908 4 input2))
;; Add the value we read earlier (stored in slot 5) and the
;; value we read last (communicated in slot 2).
;; Write the value out, setting state 2 as the state after
;; the write finishes.
4 (let
[inst _6901 (ioc-macros/aget-object state _6908 5)
inst _6903 (ioc-macros/aget-object state _6908 2)
inst _6904 (+ inst _6901 inst _6903)
state _6908 state _6908]
(ioc-macros/put! state _6908 2 result inst _6904))
;; The write has happened. Finish the state machine.
2 (let [inst _6906 (ioc-macros/aget-object state _6908 2)
state _6908 state _6908]
(ioc-macros/return-chan state _6908 inst _6906)])
(if (identical? result _5940 _auto __ :recur)
(recur)
result _5940 _auto __))
(finally (resetThreadBindingFrame old-frame _5939 _auto __)))))


```

```

;; Set up the initial state and then move on one state
state _6070 _auto __
(-> (f _6069 _auto __)
(ioc-macros/aset-all!
ioc-macros/USER-START-IDX c _6067 _auto __
ioc-macros/BINDINGS-IDX captured-bindings _6068 _auto __))

(ioc-macros/run-state-machine state _6070 _auto __))
c _6067 _auto __)


```

Unlike the C# `async` keyword, which requires a new C# compiler, macros mean that this can all be implemented in the new `core.async` library, which makes it very easy to experiment with different implementations. The library isn't yet part of the core language, though it may be at some point in the future.

Rich Hickey talks about `core.async` library in an [8]episode of the ThinkRelevance podcast. And [9]this post has some more discussion of how these constructs can be used.

One feature that seems to be missing is a means to cancel running tasks along the lines of the Task Parallel Library's cancellation tokens. There are a couple of good posts on cancellation [10]here and [11]here. These emphasise the need for cooperative cancellation and I don't yet

know of a way of doing these kind of things in Clojure.

1. <http://clojure.org/>
  2. <http://clojure.com/blog/2013/06/28/clojure-core-async-channels.html>
  3. [http://en.wikipedia.org/wiki/Communicating\\_sequential\\_processes](http://en.wikipedia.org/wiki/Communicating_sequential_processes)
  4. <http://www.lighttable.com/>
  5. <http://leinigen.org/>
  6. <https://github.com/clojure/core.async>
  7. <http://clivetong.files.wordpress.com/2013/07/lighttable.png>
  8. <http://thinkrelevance.com/blog/tags/clojure>
  9. <http://martintrojer.github.io/clojure/2013/07/07/coreasync-and-blocking-io/>
  10. <http://blogs.msdn.com/b/pfxteam/archive/2009/05/22/9635790.aspx>
  11. <http://www.drdobbs.com/parallel/interrupt-politely/207100682>
- 

#### **7.7.4 Haswell, will end well? (2013-07-26 05:41)**

[1]TheRegister has a good article on the [2]new Haswell processor architecture from Intel. I was particularly interested in the Transactional Synchronization eXtensions, a new set of instructions that try to optimise concurrent programs, either by giving the CPU a chance to execute an atomic region optimistically or by using restricted hardware transactional memory. The hardware can use the management of cache lines to determine if CPUs are contending for items inside a transactional boundary.

For details on these new instructions, and others that have been added, see chapter 8 of [3]this document. DrDobbs also has a [4]small article which covers them as does [5]this blog post.

[6]This page offers a number of pdf documents that cover the timings and operations of the various Intel chips. It looks interesting and I intend to read it over the summer.

1. <http://www.theregister.co.uk/>
  2. [http://www.theregister.co.uk/2013/06/03/feature\\_inside\\_haswell\\_intel\\_4g\\_core/print.html](http://www.theregister.co.uk/2013/06/03/feature_inside_haswell_intel_4g_core/print.html)
  3. <http://download-software.intel.com/sites/default/files/m/3/2/1/0/b/41417-319433-012.pdf>
  4. <http://www.drdobbs.com/parallel/transactional-synchronization-in-haswell/232600598>
  5. <http://software.intel.com/en-us/blogs/2012/02/07/transactional-synchronization-in-haswell>
  6. <http://www.agner.org/optimize/>
- 

#### **7.7.5 More, more, more (2013-07-29 05:44)**

[1]More Effective C++ by Scott Meyers

I've written a fair amount of C in my time, but have needed to write C++ for the last few months at work - I'm currently working on the core of [2]ANTS Memory Profiler with the intention of

adding some support for determining the ownership of un-freed unmanaged memory. C++ seems to have become a very complicated language since the time that I first saw it... you might even regard it as a toolkit as it allows the user to override the meaning of very many of the language's operators (new, ->, \*, +, -, explicit and implicit conversions), and I wonder how the user is supposed to understand the language cost model without actually stepping into the assembler that is generated.

C was nice and simple with its values and pointers, and C++ references have taken some time to understand. Dereferencing the result of new still looks a little strange to me:

```
Test(*new Foo());
```

when passing by reference to a function with a signature such as

```
void Test(Foo &foo) { }
```

In Visual Studio, I have found it really useful to flip the editor into disassembly mode and see the compiler's interpretation of what I wrote.

[3]

The screenshot shows the Microsoft Visual Studio Disassembly window. The assembly code is as follows:

```
Address: wmain(int, wchar_t **)
Viewing Options
00E21427 mov     eax,0CCCCCCCCh
00E2142C rep stos    dword ptr es:[edi]
Test(*new Foo());
00E2142E mov     dword ptr [ebp-0D4h],8
00E21438 mov     eax,dword ptr [ebp-0D4h]
00E2143E push    eax
00E2143F call    operator new (0E21195h)
00E21444 add    esp,4
00E21447 mov     dword ptr [ebp-0C8h],eax
00E2144D cmp     dword ptr [ebp-0C8h],0
00E21454 je      wmain+6Ch (0E2147Ch)
00E21456 mov     ecx,dword ptr [ebp-0D4h]
00E2145C push    ecx
00E2145D push    0
00E2145F mov     edx,dword ptr [ebp-0C8h]
00E21465 push    edx
00E21466 call    _memset (0E21078h)
00E2146B add    esp,0Ch
00E2146E mov     eax,dword ptr [ebp-0C8h]
00E21474 mov     dword ptr [ebp-0DCh],eax
00E2147A jmp     wmain+76h (0E21486h)
00E2147C mov     dword ptr [ebp-0DCh],0
00E21486 mov     ecx,dword ptr [ebp-0DCh]
00E2148C push    ecx
00E2148D call    Test (0E21089h)
Test(*new Foo());
00E21492 add    esp,4
return 0;
```

I believe the best way to learn the language is to read lots of existing code in the language, to see how the language constructs are used and to pick up the common idioms. This book is a brilliant way to learn such things. The book is a mix of tips on common C++ language misunderstandings, some common design failings in C++ programs, some ideas for implementing certain programs in C++ and some ideas on where the language is going.

The book is well written in a fairly flippant style, but isn't afraid to get into the specification when explaining why code works in a certain way. My worry was that the book may be a little old, being written in 1996, but the 35 tips seem to be very relevant and enormously helpful in improving the reader's understanding of the language. I think that I'd like a lot more information about templates and their uses, and don't remember many uses of templates in the book, but there obviously wasn't room in the current volume to cover these without displacing some of the existing tips which are very good.

The book taught me loads about the best way to use exceptions in C++, the use of destructors to prevent resource leaks and the different meaning of new and delete across the various types in the language. The information on smart pointers was really useful and there is an appendix with a demonstration implementation of auto\_ptr which is a good read. Reference counting pointers are also covered and I very much liked the discussion on implementing objects that prevent construction on the heap or the stack, and this also brought home the notion of private inheritance. A tip on proxy classes and their uses was also useful – they can be used as a way to delay the un-sharing of immutable data between instances of classes like the string class where edits are made very infrequently to the managed data.

There's also information that overlaps some that I read in [4]Stan Lippman's book on the C++ object model and a discussion on implementing multiple dispatch.

Of course, [5]Wikipedia has a mass of C++ information with pages like that on the [6]rule of three, though it is obviously not organised as a book. The book contains material that the author has sieved from a mass of possibilities, and is a very good way to get up to speed with C++.

As a side note, if you are new to C++, the C++ [7]FQA is also a very good read.

1. [http://www.amazon.co.uk/More-Effective-Programs-Professional-Computing/dp/020163371X/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1374426887&sr=1-1&keywords=more+effective+C%](http://www.amazon.co.uk/More-Effective-Programs-Professional-Computing/dp/020163371X/ref=sr_1_1?s=books&ie=UTF8&qid=1374426887&sr=1-1&keywords=more+effective+C%)
  2. <http://www.red-gate.com/products/dotnet-development/ants-memory-profiler/>
  3. <http://clivetong.files.wordpress.com/2013/07/disassembly.png>
  4. [http://www.amazon.co.uk/Inside-Object-Model-Stanley-Lippman/dp/0201834545/ref=sr\\_1\\_1?ie=UTF8&qid=1374493719&sr=8-1&keywords=inside+the+C%2B%2B+object+model](http://www.amazon.co.uk/Inside-Object-Model-Stanley-Lippman/dp/0201834545/ref=sr_1_1?ie=UTF8&qid=1374493719&sr=8-1&keywords=inside+the+C%2B%2B+object+model)
  5. <http://www.wikipedia.org/>
  6. [http://en.wikipedia.org/wiki/Rule\\_of\\_three\\_\(C%2B%2B\\_programming\)](http://en.wikipedia.org/wiki/Rule_of_three_(C%2B%2B_programming))
  7. <http://yosefk.com/c++fqa/>
- 

## 7.7.6 Don't store this for too long (2013-07-31 05:47)

[1]Windows Store App Development: C # and XAML by Pete Brown

It would be really hard not to give this 600 page book a solid five out of five stars. Previously I'd read the author's [2]Silverlight book that provided a very practical guide to writing [3]Silverlight applications. In this book he continues the theme with a book full of practical information which answers all of the questions I had about Windows Store App development using C # and XAML.

The book has some great chapters. It starts by getting the reader to write a simple hello world in chapter one. Chapter two discusses Modern UI (Metro), covering the history and influences, typography and the importance of the grid layout. There is a brief discussion of the Windows 8 UI elements such as the app bar, the charms bar and the use of tiles for giving the user a head-up display on the state of the application. Chapter three covers the Windows Runtime and its relationship to .NET, discussing the modifications to old style COM and how interaction with .NET is very straightforward.

Chapter four starts a series of chapters on the UI with a great (and short) introduction to XAML. This chapter is followed (very logically) by a chapter on the multipass layout algorithm, and

then chapters on panels, brushes and graphics, styles and resources and a chapter on displaying beautiful text.

XAML is designed to work against [4]MVVM and so the book next has a chapter on controls, binding and MVVM. There are examples which use the [5]MVVM Light toolkit which makes it easy to separate out the UI from the model, making it easy to use binding to flow information in one direction, and commands to flow events in the other. The next chapter covers view controls, semantic zoom and navigation.

Making the application work well on Windows 8 is the theme for the next few chapters. First a chapter on the app bar, and then a chapter the splash screen, app tile and tile notifications, followed by a chapter on app states which allow LayoutAwarePages to change their layout as the size of the client area changes. Two chapters follow on contracts (for interacting with other applications) and accessing the file system.

The Windows 8 model requires very little blocking of the UI thread in order to keep the UI responsive and chapter 16 discusses the IAsync\* interfaces of the Windows Runtime and how they can be made to work well with C # Tasks.

What we have learned so far is demonstrated using a small chat application that is written over the next few chapters. Along the way we find out about networking and sockets, user controls and more information about handling input from the user. The author explains the code really well and discusses many issues along the way.

Windows 8 applications need to appear to be running all of the time when they are in fact suspended by the OS in order to conserve the battery. The penultimate chapter of the book covers the suspend and activation events that the application needs to handle, as well as the points at which state needs to be saved.

The final chapter discusses the Windows Store and the kinds of tests that the application needs to pass to be a valid Windows Store application. The chapter explains how you can run the verification utilities locally and how you make an application that can be side loaded for testing.

The book is really good. Whenever I had a question, the author invariably got around to answering it, and it appears that he has spent a fair amount of times developing Windows Store applications. Now to put it all into practice.

1. [http://www.amazon.co.uk/Windows-Store-App-Development-XAML/dp/1617290947/ref=sr\\_1\\_1?ie=UTF8&qid=1374498822&sr=8-1&keywords=windows+store+app+development](http://www.amazon.co.uk/Windows-Store-App-Development-XAML/dp/1617290947/ref=sr_1_1?ie=UTF8&qid=1374498822&sr=8-1&keywords=windows+store+app+development)
2. [http://www.amazon.co.uk/Silverlight-5-Action-Pete-Brown/dp/1617290319/ref=sr\\_1\\_1?ie=UTF8&qid=1374498952&sr=8-1&keywords=Silverlight+Pete+Brown](http://www.amazon.co.uk/Silverlight-5-Action-Pete-Brown/dp/1617290319/ref=sr_1_1?ie=UTF8&qid=1374498952&sr=8-1&keywords=Silverlight+Pete+Brown)
3. <http://en.wikipedia.org/wiki/Silverlight>
4. [http://en.wikipedia.org/wiki/Model\\_View\\_ViewModel](http://en.wikipedia.org/wiki/Model_View_ViewModel)
5. <http://mvvmlight.codeplex.com/>

## 7.8 August

### 7.8.1 I say container, you say no brainer (2013-08-02 05:48)

[1]Effective STL: 50 Specific ways to improve your use of the Standard Template Library by Scott Meyers

The [2]STL is a bit of a shock when you first come across it. C++ starts out as a kind of extended C, and then there's this massive collection and algorithm library bolted onto the side of the language. It has a usage style that revolves around iterators and is highly dependent on templates which are another late addition to C++. That said, the library itself is very powerful and offers very good performance for typical CS container types. It even includes the idea of first class functions (via functors).

The book is divided into 7 chapters – containers, vector and string, associative containers, iterators, algorithms, functors and programming with the STL. Each chapter contains at least four tips associated with the subject of the chapter. These tips are very well chosen and shed light on the design and implementation of the STL.

This is a very informative book which should be accompanied with a good book on C++ templates. It will take several reads to get a good handle on the material that it covers, but I really enjoyed it.

1. [http://www.amazon.co.uk/Effective-STL-Specific-Professional-Computing/dp/0201749629/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1374496554&sr=1-1&keywords=effective+stl](http://www.amazon.co.uk/Effective-STL-Specific-Professional-Computing/dp/0201749629/ref=sr_1_1?s=books&ie=UTF8&qid=1374496554&sr=1-1&keywords=effective+stl)
  2. [http://en.wikipedia.org/wiki/Standard\\_Template\\_Library](http://en.wikipedia.org/wiki/Standard_Template_Library)
- 

### 7.8.2 Almost written at the same time (2013-08-05 05:36)

There have been several interesting posts about the implementation of Clojure's core.async. [1]This post in particular is really good, diving into the implementation and even answering some of the questions that were raised in [2]an unrelated post that asks about the implementation of new channel types. The author takes a small go block and analyses the generated code, looking in detail at the state machine and the support code in the runtime library. Channels are based on the Handler protocol which itself is quite interesting. It allows the channel to either do its action synchronously, avoiding any blocking and letting the caller take responsibility for continuing the computation, or to instead queue a closure which will be called when a value is available. This closure is used to make the state machine take another transition when the necessary values are ready. Understanding the implementation helps one understand the trade-offs involved in using this language feature.

The author has also written [3]this illuminating post on the timeouts available in core.async.

ClojureScript is a variant of Clojure that compiles down to JavaScript, and there are some interesting examples [4]here and [5]here of programs that use the core.async variant that runs on top of JavaScript. They demonstrate how it is possible to share the single threaded

browser UI between worker tasks.

1. <http://hueypetersen.com/posts/2013/08/02/the-state-machines-of-core-async/>
  2. <http://pepijndevos.nl/2013/07/29/anatomy-of-a-channel.html>
  3. <http://hueypetersen.com/posts/2013/07/10/code-read-of-core-async-timeouts/>
  4. <http://swannodette.github.io/2013/07/12/communicating-sequential-processes/>
  5. [http://swannodette.github.io/2013/08/02/100000-dom-updates/?utm\\_source=feedburner&utm\\_medium=feed&utm\\_campaign=Feed%3A+clojure+%28Planet+Clojure%29](http://swannodette.github.io/2013/08/02/100000-dom-updates/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+clojure+%28Planet+Clojure%29)
- 

### 7.8.3 That's not my tactic (2013-08-06 05:37)

[1]303 Tactical Chess Puzzles by Fred Wilson and Bruce Alberston

It's taken me ages to work through these problems, but I'm very glad that I took the time. The book starts easy and gets much harder towards the end, and the problems cover a wide range of themes and illustrate a very useful set of tactics. Well worth a read if you like this kind of thing.

I've now moved onto [2]The Complete Chess Workout.

1. [http://www.amazon.co.uk/Tactical-Chess-Puzzles-American-Puzzle/dp/080692733X/ref=sr\\_1\\_3](http://www.amazon.co.uk/Tactical-Chess-Puzzles-American-Puzzle/dp/080692733X/ref=sr_1_3)
  2. [http://www.amazon.co.uk/Complete-Chess-Workout-Train-Puzzles/dp/1857445325/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1375529970&sr=1-1&keywords=the+complete+chess+workou](http://www.amazon.co.uk/Complete-Chess-Workout-Train-Puzzles/dp/1857445325/ref=sr_1_1?s=books&ie=UTF8&qid=1375529970&sr=1-1&keywords=the+complete+chess+workou)
- 

### 7.8.4 It's as sharp as it has always been (2013-08-08 05:41)

Despite the demise of C # being slightly exaggerated, there are still a lot of interesting blog posts being written about it. [1]Eric Lippert's blog has recently published a couple of really good posts on some parts of the C # language. No matter how often you've read the specification, or the various annotated guides to the language, Eric always manages to explain the rationale behind parts of the language in a way that I've never seen before.

[2]Here he talks about the type of the null literal in C #. And explains that there are several constructs in C # that lead to objects which don't have a type.

[3]Here he talks about type checking a call to a covariant interface method. And in so doing makes it clear why the type inference mentions upper and lower bounds in the specification.

[4]Here he talks about a probably long term bug in the Microsoft C # compiler, showing the desired fields could be expressed in IL and isn't expressly forbidden by the C # specification.

In other C # news, I also came across [5]this blog post that discusses the design of the cancellation framework in the TPL. It clearly lays out the reasons for the various parts of the

design.

1. <http://ericlippert.com/>
  2. <http://ericlippert.com/>
  3. <http://ericlippert.com/>
  4. <https://communities.coverity.com/blogs/development-testing-blog/2013/07/31/c-stringextensions-and-const-fields>
  5. <http://blogs.msdn.com/b/pfxteam/archive/2009/05/22/9635790.aspx>
- 

### 7.8.5 What type of theory? (2013-08-09 05:40)

When I studied mathematics in college, I really loved [1]algebraic topology, [2]set theory and (first-order) [3]logic. After a graduate course in computer science, I was fascinated by automated theorem proving, [4]constructive proof and [5]type theory. The [6]foundations of mathematics has also been a subject that has interested me.

I came across the [7]Homotopy Type Theory site a few weeks ago while following a link from [8]this blog post where the author discusses the meaning of proof in mathematics. HTT is an attempt to re-launch the foundations of mathematics on a more constructive computational footing. [9]A book written as part of a [10]year long special program at the IAS is available [11]here with an introductory chapter that gives a very good overview of Type Theory.

There are introductory slides [12]here, a blog [13]here and a discussion of the meaning of equality in this setting [14]here.

One of the interesting parts of Type Theory, that can be instantly related to computer science applications is the notion of dependent types. [15]This paper shows some of the uses of this concept.

There team are working hard to get many of their proofs verified by proof assistants such as [16]COQ, so perhaps we really entering a new age of mathematics.

1. [http://en.wikipedia.org/wiki/Algebraic\\_topology](http://en.wikipedia.org/wiki/Algebraic_topology)
2. [http://en.wikipedia.org/wiki/Set\\_theory](http://en.wikipedia.org/wiki/Set_theory)
3. [http://en.wikipedia.org/wiki/First-order\\_logic](http://en.wikipedia.org/wiki/First-order_logic)
4. [http://en.wikipedia.org/wiki/Constructive\\_proof](http://en.wikipedia.org/wiki/Constructive_proof)
5. [http://en.wikipedia.org/wiki/Type\\_theory](http://en.wikipedia.org/wiki/Type_theory)
6. [http://en.wikipedia.org/wiki/Foundations\\_of\\_mathematics](http://en.wikipedia.org/wiki/Foundations_of_mathematics)
7. <http://homotopytypetheory.org/>
8. <http://rjlipton.wordpress.com/2013/07/14/surely-you-are-joking/>
9. <http://homotopytypetheory.org/book/>
10. <http://uf-ias-2012.wikispaces.com/>
11. <http://homotopytypetheory.org/book/>
12. <http://www.andrew.cmu.edu/user/awodey/hott/CMUslides.pdf>
13. <http://homotopytypetheory.org/blog/>
14. <http://www.andrew.cmu.edu/user/awodey/preprints/siu.pdf>

- 
15. <http://www.cse.chalmers.se/~peterd/papers/DependentTypesAtWork.pdf>
  16. <http://en.wikipedia.org/wiki/Coq>

## 7.8.6 You need to be clear with your instructions (2013-08-12 05:42)

I've been doing loads of reading about the x86/x64 architecture recently, mainly as a result of coming across the fantastic papers written by Agner Fog on his [1]optimisation page. The [2]microarchitecture paper is a really good read. It discusses how instructions are broken down into [3]micro-ops on the various Intel processors, and how this relates to instruction scheduling, branch prediction and speculative execution. It was very interesting to see a clear explanation of how instructions are scheduled across the many different execution units on the chip, and the interplay between fully utilising all of the on-chip resources and the amount of logic that is required on the chip to schedule all of the potential work.

There are a number of papers on optimisation at [4]the assembler and higher levels. [5]This one discusses [6]CPU dispatching by the Intel and GNU C++ compilers. I hadn't come across this idea before - the idea is that you recognise the CPU you are running on and then generate several specialisations of the code customised to use the facilities that the particular CPU offers. This reminded me of [7]one of the arguments for JIT compilers - the JIT certainly knows the architecture it is running on and can target its special instructions. There is obviously a trade off here though, as the target language for the JIT is often at a higher level than the CPU, making it necessary to [8]allow access to more CPU specific instructions for use in high performance applications.

Chips seem to be offering more and more hardware support for things that are traditionally implemented in software libraries. This Intel blog post talks about the [9]hardware transactional memory of the current Intel chip generation. Agner Fog has a few posts [10]lamenting the competition between the hardware vendors who are competing to add [11]different instructions to their chips.

- 
1. <http://www.agner.org/optimize/>
  2. <http://www.agner.org/optimize/microarchitecture.pdf>
  3. [http://www.agner.org/optimize/instruction\\_tables.pdf](http://www.agner.org/optimize/instruction_tables.pdf)
  4. [http://www.agner.org/optimize/optimizing\\_assembly.pdf](http://www.agner.org/optimize/optimizing_assembly.pdf)
  5. [http://www.agner.org/optimize/optimizing\\_cpp.pdf](http://www.agner.org/optimize/optimizing_cpp.pdf)
  6. <http://www.agner.org/optimize/blog/read.php?i=167>
  7. <http://bad-concurrency.blogspot.co.uk/2012/08/arithmetic-overflow-and-intrinsics.html>
  8. <http://psy-lab-saw.blogspot.co.uk/2012/12/atomiclazyset-is-performance-win-for.html>
  9. <http://software.intel.com/en-us/blogs/2013/07/25/fun-with-intel-transactional-synchronization-extensions>
  10. <http://www.agner.org/optimize/blog/read.php?i=25>
  11. <http://www.agner.org/optimize/blog/read.php?i=49>

## **7.8.7 Blast from the past (2013-08-14 05:42)**

I loved [1]this recent post which points to some of the old copies of BYTE magazine. In particular the legendary issue on Smalltalk, which I hadn't read before, as well as some good issues on Lisp and Pascal.

Some other interesting articles to read are [2]this one on the subject of co-monads (the dual to a monad) and [3]this one on the G1 collector, a high performance garbage collector for Java with an [4]associated presentation.

1. <http://www.agner.org/optimize/blog/read.php?i=49>
  2. <http://www.clojure.net/2013/02/02/Comonads/>
  3. <http://www.infoq.com/articles/G1-One-Garbage-Collector-To-Rule-Them-All>
  4. <http://www.infoq.com/presentations/java-g1>
- 

## **7.8.8 That's some art (2013-08-16 05:43)**

[1]The Art of Unit Testing by Roy Osherove

This book has been around for a number of years, and I think I've come along to it rather late... Unit testing, perhaps mixed with developer integration tests, has been around for a long time and the concepts are fairly well known. This meant that Part One of the book, which introduces Unit Testing by way of some fairly simple examples, felt a little too basic and unnecessary. Part Two which covers core techniques also covered information that I didn't need, though it did use Rhino Mocks in one of the examples and I haven't tried using an auto-mocking framework before so will do some more reading about that.

Part Three of the book was useful though. It discusses the organisation and management of unit tests, based on the three pillars of readability, maintainability and trustworthiness, and the associated discussion brought up a load of very interesting points. There is also a good discussion of how to get unit testing accepted by your organisation (though to be honest I'd rather this had been a purely technical book without the politics of introduction).

The book is a fairly quick read and brings up a number of interesting points, though personally I'd rather borrow it from a library than have to buy it.

1. [http://www.amazon.co.uk/The-Art-Unit-Testing-Examples/dp/1933988274/ref=sr\\_1\\_1?ie=UTF8&qid=1375486154&sr=8-1&keywords=the+art+of+unit+testing](http://www.amazon.co.uk/The-Art-Unit-Testing-Examples/dp/1933988274/ref=sr_1_1?ie=UTF8&qid=1375486154&sr=8-1&keywords=the+art+of+unit+testing)
- 

## **7.8.9 Exceptionally good post (2013-08-19 05:56)**

You know how it is. You're thinking about JIT in .NET and the notion of safe points, and then you start wondering how safe points and asynchronous interrupts of a thread mix (though I

am certainly no fan of exceptions injected using Thread.Abort). After a quick Google, I found [1]Chris Brumme's brilliant blog, something that I've gone to again and again to get the design rationale behind various parts of .NET. The blog hasn't been updated for a number of years, and some parts of it are slightly dated, but it is a treasure trove of detailed design ideas behind the .NET framework.

[2]This post answered all of my questions, and it was really interesting to read through [3]the article on SEH that it referenced. As usual it taught me something. For example, the stack trace associated with an exception only contains the stack between the frame in which it is raised, and the frame it which it is caught (and not the full stack trace), though when you rethrow the exception the stack trace is extended with the new information. For example, in the following code, the first stack trace that is printed is only going to contain three frames (when there is no inlining).

```
static void Main(string[] args)
{
    try
    {
        A();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.StackTrace);
    }
}

static int A()
{
    try
    {
        return B();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.StackTrace);
        throw;
    }
}

static int B()
{
    return C();
}

static int C()
{
    throw new Exception();
}
```

[4]

Time to go back through all of the old blog posts one more time!

And while we're talking about VMs, [5]this post on using Java virtual machines in low latency environments is also really interesting.

1. <http://blogs.msdn.com/b/cbrumme/>
2. <http://blogs.msdn.com/b/cbrumme/archive/2003/10/01/51524.aspx>
3. <http://www.microsoft.com/msj/0197/exception/exception.aspx>
4. <http://clivetong.files.wordpress.com/2013/08/exceptions.png>
5. <http://www.infoq.com/articles/low-latency-vp>

---

## 7.8.10 Dead or Alive? (2013-08-21 05:56)

[1]The Quantum Story: A history in 40 moments by Jim Baggott

When I was studying mathematics at university, I found Quantum mechanics really confusing. We started with Schrodinger's Wave equation and used it to derive all sorts of results, but there was never any discussion of the process of discovery. This book is one that I wish I'd read all of those years ago.

It links together 40 moments in the history of Quantum mechanics, with several sections covering the various parts of the story. Part one, Quantum of action was the most interesting to me as it starts with Planck's work on black body radiation and then traces the development of quantum mechanics via both the matrix mechanics and the wave equations. Along the way, it makes it clear how counter-intuitive the discovery was, and the difficulties that have been around ever since in interpreting the meaning of the equations. Einstein appears in several of the moments, usually in opposition to the seeming action at a distance (the famous EPR result) and the difficulty of meshing the ideas with general relativity.

The book is easy to read, with each moment not taking too much space, though each has a prologue which gives the context of the moment to allow you to see how the entire story unfolds. It is written in a clear style and makes clear the difficulties that still plague this discipline.

I learned a lot from reading it and recommend it.

1. [http://www.amazon.co.uk/Quantum-Story-history-40-moments/dp/0199655979/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1376586895&sr=1-1&keywords=the+quantum+story](http://www.amazon.co.uk/Quantum-Story-history-40-moments/dp/0199655979/ref=sr_1_1?s=books&ie=UTF8&qid=1376586895&sr=1-1&keywords=the+quantum+story)
- 

### 7.8.11 How clear is the proof? (2013-08-23 05:56)

[1]Mechanizing Proof (Computing, Risk and Trust) by Donald MacKenzie

During the 1970s and 1980s automated verification of program correctness promised so much. You'd be able to take your program, mathematically specify what you expected the program to compute, and then let a theorem prover check that the program satisfied the requirements. Indeed, when I was doing my Phd in 1991, one of the potential projects was the verification of the specification of a washing machine controller.

Things turned out not to be so easy though. What did it mean to prove that a program was correct? How can you be sure that a positive result from a verification tool really means that the program is correct? How do you know that the specification is correct?

This book addresses these questions and at the same time gives the history of the automated verification from its early days until the turn of the century. The author has taken lots of material and converted it into an informative and provocative book which discusses what mathematical proof actually is, and whether this kind of proof is really what we want when we say that a program has been verified. he asks whether it is meaningful to verify a program without also verifying the hardware on which the program runs.

Fascinating, interesting and thought provoking.

1. [http://www.amazon.co.uk/Mechanizing-Proof-Computing-Inside-Technology/dp/0262133938/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1376586960&sr=1-1&keywords=mechanizing+proof](http://www.amazon.co.uk/Mechanizing-Proof-Computing-Inside-Technology/dp/0262133938/ref=sr_1_1?s=books&ie=UTF8&qid=1376586960&sr=1-1&keywords=mechanizing+proof)

---

## 7.9 September

### 7.9.1 Surface Detail does Matter (2013-09-06 06:16)

I've just returned to work after a six week sabbatical, so I haven't had too much time for computing related blogging. Over the six weeks I finished off reading the whole of the [1]Culture series, a series of ten fantastic science fiction novels set in a semi-anarchist utopia with very current human themes running throughout the stories. And also spent way too much time thinking about [2]August's IBM Ponder (which is fairly easy when you hit the right approach).

Anyway, just a few notes on some good videos and blog posts that I have come across over the six weeks.

Two videos from Channel9 which (for some reason) they delayed publishing. [3]Compiling MSIL to JS talks about some work Microsoft did to translate IL into JS – one of the many transcompilers available. The interview discusses some of the issues involved, and I think a version of this work is openly available [4]here. A second video provides a good description of the marshalling that goes on when the [5]CLR and the WinRT worlds mix, and the representation of WinRT types in MDIL files. This is covered in some articles on MSDN, but face-to-face white boarding makes things quite clear.

There's a good [6]Javascript Jabber on REACT, Facebook's Javascript framework. What I found interesting about this framework is that the implementation works on a shadow copy of the DOM which is then synchronised with the real DOM, avoiding the need to make multiple changes to the DOM between screen repaints. As some Google webcasts point out, interacting with the DOM often involves reference counting with the unmanaged world, and can cost a lot in performance.

Promises are very heavily used in WinJS, and this blog post demonstrates [7]implementing Promises in Javascript, making lots of interesting observations along the way.

Lisp based languages such as Clojure offer macros, a way to manipulate the abstract syntax tree at compile time, a concept that can be a little confusing at first. This [8]InfoQ presentation about Clojure macros is a straightforward discussion of how they work, and this video on [9]macros vs monads discusses where the two technologies have a place.

Lastly, [10]5 myths about Actors debunks some of the misinformation that covers this technology. And as a curiosity, [11]this post demonstrates that C++ templates are Turing complete.

I'm off to learn more about [12]homomorphic encryption and [13]dependently typed meta-programming.

1. [http://en.wikipedia.org/wiki/Culture\\_series](http://en.wikipedia.org/wiki/Culture_series)

2. [http://domino.research.ibm.com/Comm/wwwr\\_ponder.nsf/Challenges/August2013.html](http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/Challenges/August2013.html)

3. <http://channel9.msdn.com/Blogs/Charles/From-the-Archives-Erik-Meijer-and-Mark-Shields-Compiling-MSIL-to-JS>

4. <https://github.com/Reactive-Extensions/IL2JS>

5. <http://channel9.msdn.com/posts/Shawn-Farkas-WinRT-and-the-CLR>
  6. <http://javascriptjabber.com/073-jsj-react-with-pete-hunt-and-jordan-walke/>
  7. <http://modernjavascript.blogspot.co.uk/2013/08/promisesa-understanding-by-doing.html>
  8. <http://www.infoq.com/presentations/macros-clojure-west-2013>
  9. <http://www.infoq.com/presentations/Macros-Monads>
  10. <http://tech.kinja.com/5-myths-about-actors-1190754834>
  11. <http://matt.might.net/articles/c++-template-meta-programming-with-lambda-calculus/>
  12. [http://en.wikipedia.org/wiki/Homomorphic\\_encryption](http://en.wikipedia.org/wiki/Homomorphic_encryption)
  13. [http://www.youtube.com/playlist?list=PL\\_shDsyy0xhKhsBUaVXTJ2uJ78EGBpvQa](http://www.youtube.com/playlist?list=PL_shDsyy0xhKhsBUaVXTJ2uJ78EGBpvQa)
- 

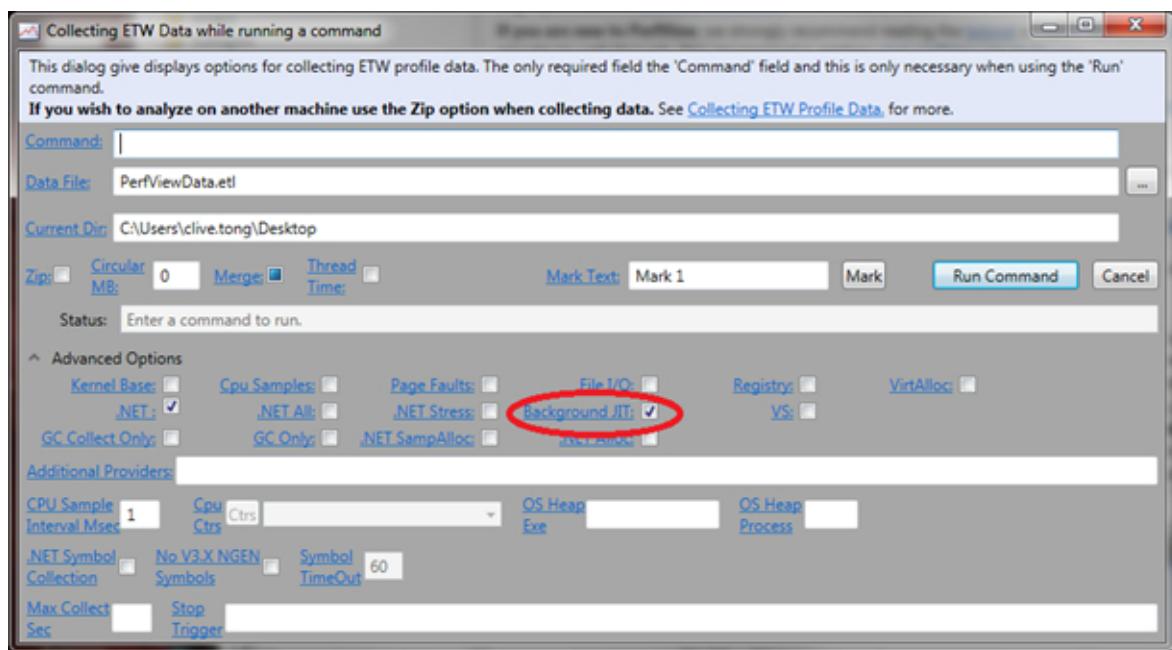
### 7.9.2 I'll use multiple cores, but not for long (2013-09-10 06:35)

I heard about [1]multi-core JIT a long time ago, but have only just got to try it in our product at work. Aside from needing to use Reflection to let our application run normally on .NET 4, and [2]only use the multi-core JIT when it is running on 4.5, it is fairly easy to plug in to the Main method. I set things up so that the profile output file would be written to the desktop, ran the application, exited and waited for the system to create the state file (which lists the methods that have been jitted so that on the next run these can be jitted in the background). Nothing happened.

I took a simple Windows Forms application and added the same profiler initialization code to that. After running it, the relevant trace file appeared on my desktop.

It was a while before I found out that I could use ETW via the [3]PerfView tool to get a trace of the activity of what PerfView calls background JIT.

[4]



I ran the application and looked at the JIT stats. Sure enough the background jitting was being aborted.

[5]

#### JIT Stats for for Process 11844: RedGate.MemoryProfiler.UI

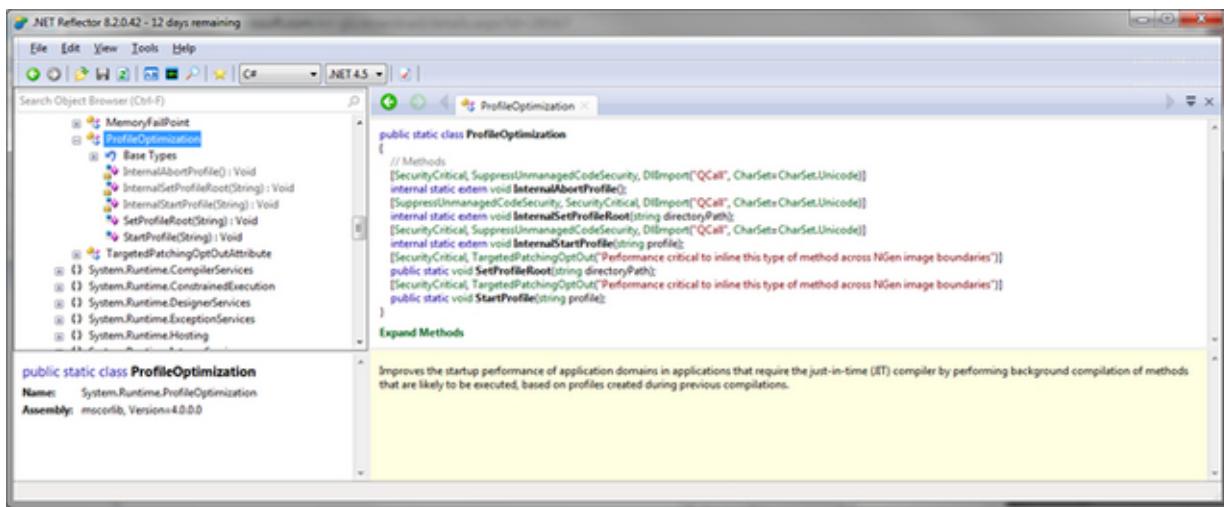
- CommandLine: "E:\Hg\AntsMemoryProfiler2\build\Debug\RedGate.MemoryProfiler.UI.exe"
- Process CPU Time: 10,911 msec
- This process uses Background JIT compilation (System.Runtime.ProfileOptimize)
  - WARNING: Background JIT aborted at 1,834.041 Msec
  - The last assembly before the abort was 'RedGate.Memory.NotUI.resources' loaded unsuccessfully at 1,833.202
  - [View Raw Background Jit Diagnostics](#)
- See [Guide to Background JIT](#) for more on background JIT
- Total Number of JIT compiled methods : 8,894
- Total MSec JIT compiling : 8,135
- JIT compilation time as a percentage of total process CPU time : 74.6%

So it looked like the background jitting was being aborted after the system had failed to load a .resources file. After some thought it turned out that this file was being probed as part of the lookup of satellite resources. The forms in the application were being localized and so that system tries to find the resources in a number of places before falling back to those that embedded into the main assembly.

I modified a newly created WindowsForms application to make the main form Localizable, which caused Visual Studio to add a resx file and to start using resource lookups. The multi-core JIT file was still generated.

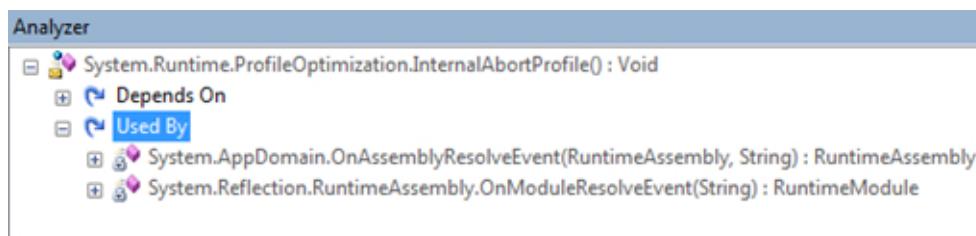
In order to investigate further I got out Reflector and had a look at the ProfileOptimization static class. I noticed the InternalAbortProfile method.

[6]



Using the Analyse function of Reflector we can see where the abort method is called.

[7]



Looking at the first of these methods, we can see that the abort is called if an assembly load fails and if there are any subscribers to the AssemblyResolve event.

[8]

```
[SecurityCritical]
private RuntimeAssembly OnAssemblyResolveEvent(RuntimeAssembly assembly, string assemblyFullName)
{
    ResolveEventHandler handler = this._AssemblyResolve;
    if (handler != null)
    {
        ProfileOptimization.InternalAbortProfile();
        Delegate[] invocationList = handler.GetInvocationList();
        int length = invocationList.Length;
        for (int i = 0; i < length; i++)
        {
            RuntimeAssembly runtimeAssembly = GetRuntimeAssembly(((ResolveEventHandler) invocationList[i])(this,
                assembly));
            if (runtimeAssembly != null)
            {
                return runtimeAssembly;
            }
        }
    }
    return null;
}
```

Sure enough. Taking my default WindowsForms application and adding the line  
AppDomain.CurrentDomain.AssemblyResolve += delegate { return null; };

stops the multi-core JIT. This seems like a shame. I imagine most windows applications out there are going to be using resource lookup and so the multi-core JIT isn't going to work if there are any subscribers to the AssemblyResolve event. Subscribing to this event isn't as unlikely as you may think - for example, the DevExpress components that we are using are subscribing to this event to control the way the assemblies are found.

The best thing about this problem was that I got the chance to read up about PerfView and ETW (event tracing for windowing) which provides a mass of debugging information that I'm sure to find useful in the future.

1. <http://blogs.msdn.com/b/dotnet/archive/2012/10/18/an-easy-solution-for-improving-app-launch-performance.aspx>
2. <http://blog.getpaint.net/2012/09/08/using-multi-core-jit-from-net-4-0-if-net-4-5-is-installed/>
3. <http://www.microsoft.com/en-gb/download/details.aspx?id=28567>
4. <http://clivetong.files.wordpress.com/2013/09/perfview.png>
5. <http://clivetong.files.wordpress.com/2013/09/jitoff.png>
6. <http://clivetong.files.wordpress.com/2013/09/reflector.png>
7. <http://clivetong.files.wordpress.com/2013/09/analyse.png>
8. <http://clivetong.files.wordpress.com/2013/09/abort.png>

---

### 7.9.3 I'm flagging (2013-09-30 06:22)

[1]Energy, the subtle concept by Jennifer Coopersmith

Energy is a central concept in modern physics, and I hadn't fully realised the long, long path before it was recognised as a central theme. People started out being fascinated by perpetual motion machines, and after many failing designs it became clear that such devices were impossible. But why?

Experimental work by Huygens on collisions had shown experimentally that  $mv^2$  was being preserved, and Newton's laws had shown that collisions preserved  $mv$  but this was just the beginning of the story. Galileo's principle of relativity made people wonder about the conservation of such quantities, but it was really the need to explain heat and cold (thermodynamics) that made it necessary to understand energy.

Hence, things really look off with the invention of the steam engine. It was very unclear how a temperature gradient was being converted by a machine into useful work. Carnot developed his famous cycle, which used the gas laws to explain the work carried out during a single cycle of a heat machine. These laws had been found by various experimentalists over the years. The whole time, the calorific theory of heat was holding the theory back and it wasn't until the acceptance of the atomic theory that things really moved forwards. Statistical mechanics and the Hamiltonian principle of least action helped with understanding, and eventually Boltzmann generalised the laws of thermodynamics with the notion of entropy.

This book develops the story well, with brief biographical information about the various players. There is very little mathematics, but many of the experiments are described. It is really interesting to see how the many experimental results were explained away, often with incorrect explanations that just happened to work, and good to see the years where experiment led the way, with theory often taking a long time to explain observed behaviour. All while the engineers were actively using the observed behaviour.

A great read!

Over the holiday I also read [2]The big questions: Physics by Michael Brooks. This gives answers to the twenty most frequently asked questions about physics and was also a good read (though it is hard to know how they picked the particular questions).

1. [http://www.amazon.co.uk/Energy-Subtle-Concept-discovery-Feynmans/dp/0199546509/ref=sr\\_1\\_1?ie=UTF8&qid=1380439217&sr=8-1&keywords=energy+coopersmith](http://www.amazon.co.uk/Energy-Subtle-Concept-discovery-Feynmans/dp/0199546509/ref=sr_1_1?ie=UTF8&qid=1380439217&sr=8-1&keywords=energy+coopersmith)
  2. [http://www.amazon.co.uk/The-Big-Questions-Michael-Brooks/dp/1849161461/ref=sr\\_1\\_1?ie=UTF8&qid=1380453741&sr=8-1&keywords=big+questions+of+physics+michael+brooks](http://www.amazon.co.uk/The-Big-Questions-Michael-Brooks/dp/1849161461/ref=sr_1_1?ie=UTF8&qid=1380453741&sr=8-1&keywords=big+questions+of+physics+michael+brooks)
- 

## 7.10 October

### 7.10.1 When you leave, I leave too (2013-10-02 05:13)

There was discussion at work about assembly unloading. In order to get the CLR to unload an assembly, you need to use a fresh AppDomain and only load a given type into that AppDomain. When the AppDomain is unloaded, the Assembly's reference count is decremented, and the Assembly can be unloaded.

It is fairly easy to use windbg to track the module load and unload events to check this behaviour. We can build an Assembly that we intend to load into a new AppDomain.

```

namespace ClassLibrary1
{
    public class Class1
    {
        static Class1()
        {
            System.Diagnostics.Debug.WriteLine("Instance made");
        }
    }
}

```

We can then write a Main method which creates a new AppDomain and then loads this Assembly in to it.

[2]

```

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            var appDomain = AppDomain.CreateDomain("myAppDomain");
            appDomain.DoCallBack(Doit);
            AppDomain.Unload(appDomain);
            Console.ReadLine();
        }

        static void Doit()
        {
            AppDomain.CurrentDomain.Load(@"ClassLibrary1");
            var x = new ClassLibrary1.Class1();
        }
    }
}

```

If we run this executable from Windbg, as long as we have set the event filtering to output module unloads (which isn't the default behaviour), we see the module is loaded into the AppDomain and then unloaded after the AppDomain is killed.

[3]

```

ModLoad: 684a0000 68573000  C:\WINDOWS\SysWOW64\MSVCR110_CLR0400.dll
ModLoad: 75b40000 75c76000  C:\WINDOWS\SysWOW64\combase.dll at 75b40000
(3518.1efc): Unknown exception - code 04242420 (first chance)
ModLoad: 04d70000 05262000  mscorlib.dll
ModLoad: 05270000 05762000  mscorlib.dll
Unload module mscorlib.dll at 04d70000
ModLoad: 75280000 75288000  C:\WINDOWS\SysWOW64\VERSION.dll
ModLoad: 68430000 6849e000  C:\Windows\Microsoft.NET\Framework\v4.0.30319\clrjit.dll
ModLoad: 75b40000 75c76000  C:\WINDOWS\SysWOW64\combase.dll
ModLoad: 6b060000 6b07a000  C:\WINDOWS\SysWOW64\bcrypt.dll
ModLoad: 76470000 76589000  C:\WINDOWS\SysWOW64\ole32.dll
ModLoad: 74480000 7449a000  C:\WINDOWS\SysWOW64\CRYPTSP.dll
ModLoad: 74440000 7447e000  C:\WINDOWS\SysWOW64\rsaenh.dll
ModLoad: 05050000 05058000  ClassLibrary1.dll
ModLoad: 05070000 05078000  ClassLibrary1.dll
Unload module ClassLibrary1.dll at 05050000
ModLoad: 05770000 05abc000  System.dll
ModLoad: 05ac0000 05e0c000  System.dll
Unload module System.dll at 05770000
ModLoad: 050e0000 05142000  System.Configuration.dll
ModLoad: 05150000 051b2000  System.Configuration.dll
Unload module System.Configuration.dll at 050e0000
ModLoad: 05770000 05a04000  System.Xml.dll
ModLoad: 05e10000 060a4000  System.Xml.dll
Unload module System.Xml.dll at 05770000
Instance made
Unload module ClassLibrary1.dll at 05070000
Unload module System.dll at 05ac0000
Unload module System.Configuration.dll at 05150000
Unload module System.Xml.dll at 05e10000

```

All as expected, but it is good to check one's expectations from time to time.

1. <http://clivetong.files.wordpress.com/2013/10/classlibrary1.png>
  2. <http://clivetong.files.wordpress.com/2013/10/driver.png>
  3. <http://clivetong.files.wordpress.com/2013/10/unloads.png>
- 

### 7.10.2 Something for the weekend (2013-10-04 05:15)

Here are various bits and pieces I have been reading recently.

This [1]talk from Going Native 2013, on the implementation of Async in C++, was really interesting. The presenter talks about the implementation which currently uses side-stacks, 4KB stack segments which can be used for functions that are going to need to be suspended. Back in the day, we had a similar technique for preserving stacks of C code in our Lisp system. In the questions section, the presenter talks about the use of [2]Fibers on Windows as the implementation technique.

The slide decks from the recent StrangeLoop conference can be found [3]here. The talks cover a wide range of diverse topics, and there are loads of interesting observations. I particularly liked [4]Odersky's talk on the trouble with types.

On the subject of typed functional languages, [5]this GitHub project is an F # library for manipulating IL. One of samples is the start of an implementation of an IL interpreter. There is a lot of work to do, but this is the kind of thing that I've been thinking about doing with Reflector for quite some time, so I might just put some time into getting this code going.

[6]This Mozilla blog contains lots of interesting implementation details on the various Mozilla JavaScript engines. Some of the articles are again well worth a read. On a JavaScript theme, [7]this article on JavaScript Promise Patterns is also interesting.

1. <http://channel9.msdn.com/Events/GoingNative/2013/Bringing-await-to-Cpp>
  2. [http://msdn.microsoft.com/en-us/library/windows/desktop/ms682661\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms682661(v=vs.85).aspx)
  3. <https://github.com/strangeloop/StrangeLoop2013/tree/master/slides/sessions>
  4. [http://en.wikipedia.org/wiki/Martin\\_Odersky](http://en.wikipedia.org/wiki/Martin_Odersky)
  5. <https://github.com/robertpi/AbstractIL>
  6. <http://blog.mozilla.org/javascript/>
  7. <http://modernjavascript.blogspot.co.uk/2013/09/promise-patterns.html>
- 

### 7.10.3 It's not what you do, it's the way that you use it (2013-10-10 05:06)

Sometimes a really simple feature can lead to quite unexpected consequences. In this case I'm thinking of the [1]XMLHttpRequest object and its [2]Open method which allows a client to make a HTTP request to a server. JavaScript runs in a single threaded fashion, and making a synchronous call via this object blocks the calling thread until the response is received. This gives us a technique for blocking the JavaScript execution.

In some Lisp systems stepping is implemented by walking the abstract syntax tree and wrapping interesting call points by instrumenting the code to enter the stepper GUI. [3]Aardwolf does just this in JavaScript in order to make the code remotely debuggable. Its runtime can parse JavaScript before it is delivered to the target platform, inserting calls to the Aardwolf.updatePosition function before each source line in the original JavaScript. When this function is called at runtime, the synchronous XMLHttpRequest blocks the JavaScript engine, and the data passed across the call can identify the line at which the engine is blocked. The debugging server has knowledge of the line currently being executed. By returning a message back to the client, the debugging server can get the client to call arbitrary JavaScript, allowing the debugger to set additional breakpoints or evaluate expressions in context (using eval).

The author has a video [4]here showing the library doing its stuff. The [5]GitHub code is an interesting read and in the [6]aardwolf.js file you can see what the updatePosition function actually does.

It's a very clever scheme for debugging JavaScript code in place. The runtime takes care of recording breakpoints, and although things are going to be slower (as there's lots of code calling updatePosition which is essentially polling for whether a break is needed) it will be interesting to see if the technique is practical.

1. <http://en.wikipedia.org/wiki/XMLHttpRequest>
  2. [http://en.wikipedia.org/wiki/XMLHttpRequest#The\\_open\\_method](http://en.wikipedia.org/wiki/XMLHttpRequest#The_open_method)
  3. <https://github.com/lexandera/Aardwolf>
  4. <http://lexandera.com/aardwolf/>
  5. <https://github.com/lexandera/Aardwolf>
  6. <https://github.com/lexandera/Aardwolf/blob/master/js/aardwolf.js>
- 

#### 7.10.4 I didn't know that! (2013-10-14 06:12)

At work this week, I noticed and then learned a couple of things that I hadn't realised before.

We are doing some work on making our in-process COM component report errors back to us when something goes wrong. We use [1]structured exception handling to catch all errors. The handler can capture a stack trace and send it back to us by passing it back to a managed .NET application which sends the error back using some existing technology ( [2]SmartAssembly ). What surprised me was that when we continued a deliberately faulting application in the debugger, the application seemed to be stuck on the line that raises the error.

I eventually simplified an example of this behaviour to a standalone piece of C++ that looks like this:

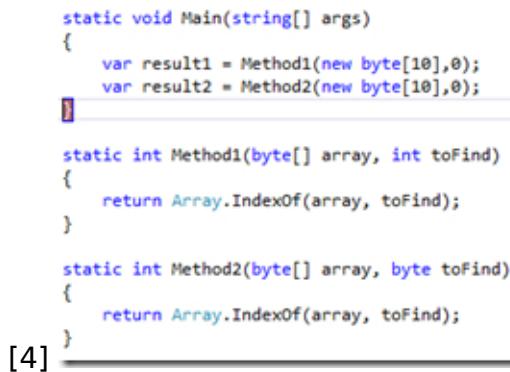
```
int _tmain(int argc, _TCHAR* argv[])
{
    int * p = 0;
    return *p;
}
```

[3]

If you continue in Visual Studio after the exception is trapped, you get the same exception raised again. Single stepping into the code that that is invoked, you can see that the system

checks for a debugger being attached, and then deliberately chooses to restart on the problem instruction. Without a debugger, it drops through to the usual uncaught exception handler which potentially dumps the process memory. Thinking about it, this behaviour is good, as if you accidentally continue in the debugger, you do not lose your error state. I'm not sure I'd noticed this behaviour in the past though.

While writing some code that dealt with byte arrays, I also came across the following behaviour.



A screenshot of the Visual Studio code editor. A tooltip is displayed over the call to `Array.IndexOf` in the `Method1` and `Method2` methods. The tooltip contains the following text:

```
Method1(byte[] array, int toFind)
{
    return Array.IndexOf(array, toFind);
}

Method2(byte[] array, byte toFind)
{
    return Array.IndexOf(array, toFind);
}
```

The number [4] is visible at the bottom left of the tooltip area.

It is initially surprising that `result1` is -1, compared to `result2` being 0 ie zero isn't a member of the byte array. To understand why, hover the mouse in Visual Studio over the call to `IndexOf`.

1. [http://en.wikipedia.org/wiki/Microsoft-specific\\_exception\\_handling\\_mechanisms](http://en.wikipedia.org/wiki/Microsoft-specific_exception_handling_mechanisms)
2. <http://www.red-gate.com/products/dotnet-development/smartassembly/features/#automated-error-reporting>
3. <http://clivetong.files.wordpress.com/2013/10/cppseh.png>
4. <http://clivetong.files.wordpress.com/2013/10/bytarray.png>

---

## 7.10.5 Debug the debugger (2013-10-21 05:21)

It came up at work the other day, and so I guess it's worth writing down here. Using a debugger on a process affects the process you are debugging. The target process, of course, can ask whether there is a debugger attached and therefore do different things depending on whether one is present or not. Without using reflection though, it is common for extra threads to come into existence in the target process in order to support the debugger break operation, which is going to affect the target.

[I should add that we are talking about debugging unmanaged code here. Debugging a managed process is achieved by talking to an extra debugging support thread that the CLR creates. This is needed because it isn't sufficient for the debugger to use only the usual low level modification API to insert breakpoints, as the insertion of a breakpoint needs to synchronise with the code actually being jitted in the first place]

To see what's going on, start two instances of [1]Windbg and connect the first instance to the second. In the first instance set a breakpoint on `DbgUiIssueRemoteBreakin`.

[2]

740

```

0:001> .symfix d:\localsymbols
0:001> .reload
Reloading current modules

0:001> x *!*DbgUiIssueRemoteBreakin
771bfd7b      ntdll!DbgUiIssueRemoteBreakin (<no parameter info>)
755e1094      KERNEL32!_imp_DbgUiIssueRemoteBreakin = <no type infor
0:001> bp ntdll!DbgUiIssueRemoteBreakin
0:001> g

```

Attach the second instance to a running process or open an executable in it. When it is successfully debugging something, press the break button.

The first instance will show that we enter DbgUiRemoteBreakin.

[3]

```

ntdll!DbgUiIssueRemoteBreakin:
771bfd7b 8bff    mov     edi,edi
0:000 u
ntdll!DbgUiIssueRemoteBreakin:
771bfd7b 8ff    mov     edi,edi
771bfd7d 55    push    ebp
771bfd7d 8bec    mov     esp,ebp
771bfd80 83ec0c    sub     esp,0Ch
771bfd83 8d45f4    lea     eax,[ebp-0Ch]
771bfd86 33d2    xor     edx,edx
771bfd88 56    push    esi
771bfd89 50    push    eax
0:000 u
ntdll!DbgUiIssueRemoteBreakin+0xf:
771bfd8a 8d45fc    lea     eax,[ebp-4]
771bfd8d 50    push    eax
771bfd8e 33c0    xor     eax,eax
771bfd90 50    push    eax
771bfd91 68c4fd1b77    push    offset ntdll!DbgUiRemoteBreakin (771bfcd4)
771bfd96 51    push    ecx
771bfd97 8d4d08    mov     ecx,dword ptr [ebp+8]
771bfd98 6800400000    push    4000h
0:000 u
ntdll!DbgUiIssueRemoteBreakin+0x24:
771bfd9f 50    push    eax
771bfda0 50    push    eax
771bfda1 6a02    push    2
771bfda3 e86b7ffcff    call    ntdll!RtlpCreateUserThreadEx (77187d13)

```

and by disassembling we can see that it is going to create a thread in the target process. From the arguments, the new thread is going to run the DbgUiRemoteBreakin function in ntdll.dll.

Press “g” in the first instance, and the second instance will show that we have broken inside the process that it is debugging. We can see that the “int 3” instruction has been called from the DbgUiRemoteBreakin function in the target process’ ntdll.dll.

[4]

```

0:005> g
(1094:e60): Break instruction exception - code 80000003 (first chance)
eax=7df56000 ebx=00000000 ecx=00000000 edx=771bfcd4 esi=00000000 edi=00000000
eip=7713879c esp=054dfa40 ebp=054dfa6c icpl=0 nv up ei pl zr na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246
ntdll!DbgBreakPoint:
7713879c cc        int     3
0:006> t
eax=7df56000 ebx=00000000 ecx=00000000 edx=771bfcd4 esi=00000000 edi=00000000
eip=7713879d esp=054dfa40 ebp=054dfa6c icpl=0 nv up ei pl zr na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246
ntdll!DbgBreakPoint+0x1:
7713879d c3        ret
0:006> t
eax=7df56000 ebx=00000000 ecx=00000000 edx=771bfcd4 esi=00000000 edi=00000000
eip=771bfcd4 esp=054dfa44 ebp=054dfa6c icpl=0 nv up ei pl zr na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246
ntdll!DbgUiRemoteBreakin+0x39:
771bfcd4 eb07    jmp    ntdll!DbgUiRemoteBreakin+0x42 (771bfe06)
0:006> u 771bfcd4
ntdll!DbgUiRemoteBreakin:
771bfcd4 6a08    push    0
771bfcd6 6818fe1b77    push    offset ntdll!DbgUiRemoteBreakin+0x54 (771bfe18)
771bfcd8 e860084f9ff    call    ntdll!RtlInitializeCriticalSectionAndSpinCount+0x3e
771bfdd0 64a130000000    mov     eax,dword ptr fs:[00000030h]
771bfdd6 80780200    cmp     byte ptr [eax+2],0
771bfdd8 7509    jne    ntdll!DbgUiRemoteBreakin+0x21 (771bfde5)
771bfdd9 f605d402fe7f02    test    byte ptr [SharedUserData+0x2d4 (77fe02d4)].2
771bfddc 7428    je     ntdll!DbgUiRemoteBreakin+0x49 (771bfe0d)
0:006> u
ntdll!DbgUiRemoteBreakin+0x21:
771bfde5 64a118000000    mov     eax,dword ptr fs:[00000018h]
771bfdeb f680ca0f000020    test    byte ptr [eax+0FCAh],20h
771bfdd2 7519    jne    ntdll!DbgUiRemoteBreakin+0x49 (771bfe0d)
771bfdd4 8365fc00    and     dword ptr [ebp-4],0
771bfdd8 e89e89f7ff    call    ntdll!DbgBreakPoint (7713879c)
771bfdd9 eb07    jmp    ntdll!DbgUiRemoteBreakin+0x42 (771bfe06)
771bfddc 33c0    xor     eax,eax
771bfddc 40    inc     eax

```

Likewise setting a breakpoint on WriteProcessMemory and inserting a breakpoint via the second instance, we can see that the debugger writes to the other process's memory to add a breakpoint instruction.

[5]

```
0:001> bp KERNELBASE!WriteProcessMemory
0:001> g
Breakpoint 1 hit
eax=04c9f024 ebx=00000000 ecx=03958430 edx=04c9f05c esi=7713879d edi=00000000
eip=76d109b9 esp=04c9f00c ebp=04c9f028 iopl=0 nv up ei pl nz na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000206
KERNELBASE!WriteProcessMemory:
76d109b9 8bff        mov     edi,edi
0:016> k
ChildEBP RetAddr
04c9f008 5aadf26d KERNELBASE!WriteProcessMemory
04c9f028 5aa1effd dbgeng!LiveUserDebugServices::WriteVirtual+0x2b
04c9f064 5a9a8baf dbgeng!BaseX86MachineInfo::InsertBreakpointInstruction+0xe0
04c9f0b4 5a9a361c dbgeng!LiveUserTargetInfo::InsertCodeBreakpoint+0x52
04c9f0ec 5a9a50b0 dbgeng!CodeBreakpoint::Insert+0x74
04c9f95c 5aa0d819 dbgeng!InsertBreakpoints+0x655
04c9f9c4 5a9c14b0 dbgeng!PrepareForExecution+0x48d
04c9fa24 5a9c1b7f dbgeng!RawWaitForEvent+0x32
04c9fa5c 00e5f605 dbgeng!DebugClient::WaitForEvent+0xb7
04c9fa80 7557495d windbg!EngineLoop+0x1b5
04c9fa8c 771598ee KERNEL32!BaseThreadInitThunk+0xe
04c9fad0 771598c4 ntdll!__RtlUserThreadStart+0x20
04c9fae0 00000000 ntdll!__RtlUserThreadStart+0x1b
```

The debugger also has a way to register for debugging events in the target process, but that is going to have to wait for another blog post.

In other news this post on [6]ten things about C # seems to have generated some controversy (see the comments). I'm reminded of it because at work on Thursday we had to get around point #9, you can suppress ILDASM by using an attribute. I hasten to say that this attribute is no protection at all - you simply use a hex editor to find the string SuppressILDasm and then modify one of the bytes in the name to some other character. ILDASM will then disassembly it without complaint. So be warned. True obfuscation takes a lot more effort.

1. <http://en.wikipedia.org/wiki/WinDbg>
2. <http://clivetong.files.wordpress.com/2013/10/breakonthread.png>
3. <http://clivetong.files.wordpress.com/2013/10/breakincode.png>
4. <http://clivetong.files.wordpress.com/2013/10/breakinonnewthread.png>
5. <http://clivetong.files.wordpress.com/2013/10/writememory.png>
6. <http://escapelizard.wordpress.com/2013/10/16/10-things-you-maybe-didnt-know-about-c/>

---

## 7.10.6 To the cloud (2013-10-25 06:18)

[1]Azure in Action by Chris Hay and Brian H Prince

This is a really good introduction to Azure and was just what I needed to get up to speed in this area. I wanted a book that would tell me how to write Azure applications - the book covers Azure web and worker roles, but at the same time I wanted to get some understanding of how it all works under the covers. The book tries to do all of these things.

The book has six sections.

"Welcome to the Cloud" is a good discussion of the advantages of moving your application to the cloud - in particular scalability and you pay only for machines when they are doing work.

“Understand the Azure Service Model” talks about the basic Azure configuration and deployment model.

“Running your site with web roles” demonstrates, via some examples, the basic web role. In this role, your application runs on top of a hosted IIS. The chapter covers the various trust models and the ability to call out into native code.

“Working with BLOB storage” covers the basic storage service underlying Azure. Binary storage is demonstrated using a video encoding service, and the authors look under the covers at the REST API that sits underneath the Microsoft client library. Some of the Azure implementation details are covered when the authors talk about how the load balancer is used to ensure that incoming requests are prevented from seeing replication happening under the covers.

“Working with structured data” covers Azure table storage and moves onto SQL Azure. Again the authors demonstrate the technology by way of good code examples, and also poke into the REST API.

“Doing work with messages” talks about using worker roles and message queues to move work out of the web role and into processing that can be decoupled. This chapter also covers monitoring the application, both to record problems and to determine when the applications needs to be scaled out (more instances). This is always a better solution than scaling up (using a bigger and faster, and therefore more expensive virtual machine instance).

The book is a informative and easy read with a chatty style. My only reservation would be how up to date it is - Azure seems to be changing rapidly and any book is probably going to get out of date fairly quickly. However, it looks like a good introduction before diving into the various Microsoft training materials.

1. [http://www.amazon.co.uk/Azure-Action-Chris-Hay/dp/193518248X/ref=sr\\_1\\_1?ie=UTF8&qid=1382374396&sr=8-1&key%0D%0Awords=azure+in+action](http://www.amazon.co.uk/Azure-Action-Chris-Hay/dp/193518248X/ref=sr_1_1?ie=UTF8&qid=1382374396&sr=8-1&key%0D%0Awords=azure+in+action)
- 

### 7.10.7 Not essential, but useful anyway (2013-10-28 06:19)

[1]PhoneGap Essentials: Building Cross-Platform Mobile Apps by John M. Wargo

I wish that I'd read this book a long time ago, before I had a fairly good understanding of [2]PhoneGap. That said, part one of the book, which gives a good introduction to PhoneGap, is very good. It discusses why hybrid applications are important and the business reasons behind them, and discusses the PhoneGap technology, how it works and what it has to offer.

Part two covers the PhoneGap development tools. There's a chapter for each of the major platforms that PhoneGap supports – Android, bada, BlackBerry, iOS, Symbian and Windows Phone, so unfortunately only one of the chapter was really of interest to me. The last chapter of this section covers PhoneGap Build, a cloud service that takes your PhoneGap build which you have tested on your development machine, and generates a packaged version of the application on the other platforms.

Part three is the part of the book that really interested me. It is a whirlwind tour of the PhoneGap APIs covering accelerometer, camera, capture, compass, connection, contacts, device, events,

file, geolocation, media, notification and storage. I thought this section was really good as it provides cross platform examples of using the various APIs from Javascript and gives you a very good idea of what facilities you can use from a cross platform hybrid application.

Nice read, but you may find you know some of material already.

1. [http://www.amazon.co.uk/PhoneGap-Essentials-Building-Cross-platform-Mobile/dp/0321814290/ref=sr\\_1\\_1?ie=UTF8&qid=1382373757&sr=8-1&keywords=phonegap+essentials](http://www.amazon.co.uk/PhoneGap-Essentials-Building-Cross-platform-Mobile/dp/0321814290/ref=sr_1_1?ie=UTF8&qid=1382373757&sr=8-1&keywords=phonegap+essentials)
  2. <http://phonegap.com/>
- 

### 7.10.8 Get inside (2013-10-30 05:22)

[1]Inside Windows Debugging: Practical Debugging and Tracing Strategies by Tarik Soulami

Quite simply, this is the best computing related book I have ever read.

I'm interested in the low level details of the kernel, managed and unmanaged code and design of the Windows operating system. This book offers a lot in all of these areas.

It starts by describing how debuggers work, both in user mode and at the kernel level. It goes so far as to actually list the Win32 API functions that are used in the implementation, and in a beautiful reflective example it uses one version of WinDbg to debug another instance. The author takes away the fear of kernel debugging, using a mass of very informative examples to show how to do the common tasks that you need to do in debugger – finding functions you are interested in and setting break and watch points. Getting good stack traces is the key for understanding code you didn't write, and the author shows how to get the maximum information.

I loved the diverse examples that were picked to demonstrate the various debugging techniques. As well as teaching some of the less known but useful WinDbg commands, and pointing out some good techniques to use, the examples poke into interesting parts of the implementation of Windows and teach you something useful about its design at the same time. The application verifier, the page heap and the use of gflags are just a few of the things that the book covers.

More than this though, the book also covers ETW (Event Tracing for Windows), in about a third of its pages. ETW is a low overhead logging mechanism (costing about 2 % at 10000 events a second) that is built into the Windows kernel and which can be used by other event providers for logging. Event tracing generates large log files which contains all events of particular selected categories over a particular time interval. Tools like xperf and perfview can be used to extract information from these logs in order to display informative graphs and statistics. The latter tool is aimed at helping with managed code, whereas the former is the original ETW viewer.

ETW is a great technology letting you get information even at the level of context switches, and moreover can be used as a kind of sampling profiler taking (by default) 1000 stack traces per second on the various CPUs. Again the author covers the technology in good detail, and then picks a set of debugging scenarios which demonstrate the use of the technology. And again the examples are informative and teach a lot.

I can't really say much else. To me the perfect technical book.

1. [http://www.amazon.co.uk/Inside-Windows-Debugging-Practical-Strategies/dp/0735662789/ref=sr\\_1\\_1?ie=UTF8&qid=1382516572&sr=8-1&keywords=inside+windows+debugging](http://www.amazon.co.uk/Inside-Windows-Debugging-Practical-Strategies/dp/0735662789/ref=sr_1_1?ie=UTF8&qid=1382516572&sr=8-1&keywords=inside+windows+debugging)
- 

## 7.11 November

### 7.11.1 Keep it simple (2013-11-01 05:22)

[1]Secure Coding: Principles and Practice by Mark G Graff and Kenneth R van Wyk

This book isn't a set of example vulnerabilities in pieces of software, but rather takes a step back and looks at the reasons that insecure software is produced in the first place. This leads to interesting discussions about designing software to avoid vulnerabilities, and also discussions on how to sandbox software in order to make up for deficiencies in the design.

The book lists a load of good practices, bad practices and ways to manage the risk. At all times, the author emphasises that there are business decisions to be made, and not all software needs to be completely secure – some risks have such a low probability that it would be a waste of money to try handling them. The key insight is to know what the risks are, and which risk scenarios you have chosen to ignore in the design of the software.

This was a fairly good read, and it is always good to look at software design from the business perspective, but I think I would have enjoyed a book that worked closer to particular technologies that I use.

1. [http://www.amazon.co.uk/Secure-Coding-Principles-Mark-Graff/dp/0596002424/ref=sr\\_1\\_1?ie=UTF8&qid=1382373316&sr=8-1&keywords=secure+coding+principles](http://www.amazon.co.uk/Secure-Coding-Principles-Mark-Graff/dp/0596002424/ref=sr_1_1?ie=UTF8&qid=1382373316&sr=8-1&keywords=secure+coding+principles)
- 

### 7.11.2 That's not what I expected (2013-11-03 10:20)

I upgraded my laptop to Windows 8.1, and the windows store icon disappeared and the mail application started failing to start. Fortunately, I found the following [1]post. Running the Microsoft Apps Troubleshooter, it told me that it had fixed a number of security permissions. Then running WsReset got everything running again.

1. [http://answers.microsoft.com/en-us/windows/forum/windows\\_8-windows\\_store/windows-8-store-icon-is-missing-from-start-menu/f51295fa-4c22-49fa-ac3b-b272ab759a32](http://answers.microsoft.com/en-us/windows/forum/windows_8-windows_store/windows-8-store-icon-is-missing-from-start-menu/f51295fa-4c22-49fa-ac3b-b272ab759a32)
-

### **7.11.3 I'm a little blue today (2013-11-04 10:22)**

We're [1]currently re-organising at work, and my time working on the .NET profilers has come to an end. In a couple of weeks I'll be working on products for [2]Windows Azure which should prove to be an interesting challenge. I'll be working on code at a much higher level than I have for a while... I did enterprise email archiving when I first moved to [3]Red Gate, but have worked on .NET Reflector (a decompiler) and the [4].NET profilers in the meantime.

Azure seems to be moving on in massive leaps. Just the other day, they announced yet [5]more improvements such as the ability to debug your Azure instances using Visual Studio as a remote debugger.

In order to get up to speed with Azure, I've been watching a lot of videos. To get a good idea of what is happening at a low level, there are some good talks by Mark Russinovich [6]here, [7]here and [8]here. [9]This talk on Windows Azure storage is also very good, as is this [10]discussion of auto-scaling. Scott Hanselman is also doing a new series called [11]Windows Azure Friday – the [12]first episode demonstrates the new 2.2 SDK mentioned above.

1. <http://dev.red-gate.com/restructuring/>
2. <http://www.windowsazure.com/en-us/>
3. <http://www.red-gate.com/>
4. <http://www.red-gate.com/products/dotnet-development/ants-memory-profiler/solving-memory-problems/understanding-and-troubleshooting-unmanaged-memory>
5. <http://weblogs.asp.net/scottgu/archive/2013/10/22/windows-azure-announcing-release-of-windows-azure-sdk-2-2-with-lots-of-goodies.aspx>
6. <http://channel9.msdn.com/Events/Build/BUILD2011/SAC-852F>
7. <http://channel9.msdn.com/Events/TechEd/NorthAmerica/2013/WAD-B402#fbid=>
8. <http://channel9.msdn.com/Events/Build/BUILD2011/SAC-853T>
9. <http://channel9.msdn.com/Events/TechEd/NorthAmerica/2013/WAD-B406#fbid=>
10. <http://channel9.msdn.com/Events/TechEd/NorthAmerica/2013/WAD-B401#fbid=>
11. <http://channel9.msdn.com/Shows/Windows-Azure-Friday/feed/mp4high>
12. <http://channel9.msdn.com/Shows/Windows-Azure-Friday/Getting-Started-with-Windows-Azure-the-SDK-and-Visual-Studio>

---

### **7.11.4 Puzzling (2013-11-18 06:51)**

[1]Algorithmic Puzzles by Anany Levitin and Maria Levitin

I am not really sure what the "Algorithmic" means in the title, as this is mainly a book of maths puzzles. There were very few questions that asked you to design an algorithm that worked with a particular asymptotic complexity or maximum number of steps. Most questions are just straight forward maths puzzles. That said, I enjoyed the book very much.

The first thirty pages are a general tutorial on techniques for solving such puzzles using, for example, divide-and-conquer, transform-and-conquer and dynamic programming. It's a very good summary of the techniques and demonstrates them by working through a number of examples.

The main part of the book is a list of 150 puzzles, divided into three categories of difficulty. They were all very interesting and I was pleased that I hadn't seen most of them before. Quite a few seemed to be solved by choosing an appropriate invariant. The puzzles are listed, then there is a chapter of small one sentence hints for each one, and then the solution of each puzzle is described in detail, often with references to the place where the puzzle was originally posted.

If you like puzzles, this is a really good book to work through.

1. [http://www.amazon.co.uk/Algorithmic-Puzzles-Anany-Levitin/dp/0199740445/ref=sr\\_1\\_1?ie=UTF8&qid=1384615321&sr=8-1&keywords=algorithmic+puzzles](http://www.amazon.co.uk/Algorithmic-Puzzles-Anany-Levitin/dp/0199740445/ref=sr_1_1?ie=UTF8&qid=1384615321&sr=8-1&keywords=algorithmic+puzzles)
- 

### 7.11.5 This language has more than one dimension (2013-11-25 06:04)

I'm really enjoying the [1]Principles of Reactive Programming course which is currently running on [2]Coursera. We are currently at week three, and have had some interesting lectures on [3]monads, and how they can be used to model effects such as latency. This theory was used as a way of introducing the [4]Future and [5]Promise constructs in Scala which we have used in the implementation of a scalable web server.

The programming language of the course is [6]Scala, a language that I hadn't used before. It has been fairly straightforward to get to grips with the language while programming the examples, though I must admit to having had some troubles with debugging using Eclipse.

Scala seems to have a massive range of language features. As well as being a straight forward functional/object hybrid (think F #), it has interesting notions such as objects being callable simply by having an [7]apply method defined on them, a type representing a [8]PartialFunction and [9]traits – interfaces that can contain implementation, and which can be multiply inherited by concrete classes. I think the style of [10]Mixin programming that this encourages is very powerful.

I am a massive fan of Common Lisp macros, and it is interesting to see [11]these in Scala too. Macros are used to implement all sorts of library features including [12]async, a feature somewhat like the C # async facility. It allows you to create of a future together with a means to await the completion of other futures, with the key point being that the code remains linear and it is the compiler that is doing the lifting of code blocks into completion callbacks. In Clojure, macros have been used to implement this kind of functionality too modelled on Go blocks. Scala's implementation is a little different in semantics from the C # feature of the same name which is based on the [13]Task<> type. In C # the code runs [14]synchronously until the first wait; in Scala the async block instantly creates a future which spawns work elsewhere.

Scala also offers [15]call-by-name semantics for function parameters, and a type of [16]language level dependency injection via implicits. Sometimes, the number of language features feels a little overwhelming.

Macros are a great feature for implementing new language features. In a call-by-value language without call-by-need parameters, they are often used to implement language constructs such as IF where we want to avoid implementing some code depending on another parameter without the need to wrap it into a construct like a lambda expression to delay the execution.

Using code walking, the entire function definition can be expanded and analysed (as in the implementation of [17]async in Clojure where the code is translated into [18]SSA form and essentially transformed into code with a different meaning). There's a good introduction to code walking [19]here.

Code walking (macro expansion) is also vital if you want to do type inference in an untyped language such as Clojure or Common Lisp. [20]This paper discusses some of the work that went into Racket Scheme for doing this. This formed the basis of [21]Typed Clojure, which is also covered in [22]the author's thesis. When I worked on a Lisp compiler in my first job after university, I spent a little of my spare time trying to implement type inference for Common Lisp based on the work of Henry Baker on the [23]Nimble type inferencer. This style of inference lacks the elegance of the type checking in a language like ML, where the compiler does virtually all of the type inference you'll ever need using [24]Algorithm W or in more recent times constraint solving, on a language that constrains the types to make this possible.

I'm not quite sure how I feel about gradual typing and would like to try it on some larger examples. In a language like ML, once the code type checks, it seems to work. In a language like C #, the type checking sometimes gets in the way of prototyping and yet even correctly typed code fails at runtime. In a language like Common Lisp, you can prototype very quickly, and pick up the type problems in the unit tests. I also think you can be using type checking for correctness and for improving the speed of the compiled code, and I've always relied on type checking for the latter in the code I've written.

The developer for core.typed is interviewed in [25]this podcast.

1. <https://class.coursera.org/reactive-001/class>
2. <https://www.coursera.org/>
3. [http://en.wikipedia.org/wiki/Monad\\_\(functional\\_programming\)](http://en.wikipedia.org/wiki/Monad_(functional_programming))
4. <http://docs.scala-lang.org/overviews/core/futures.html>
5. <http://docs.scala-lang.org/overviews/core/futures.html>
6. <http://docs.scala-lang.org/>
7. <http://stackoverflow.com/questions/9737352/what-is-the-apply-function-in-scala>
8. <http://www.scala-lang.org/api/current/index.html#scala.PartialFunction>
9. <http://twitter.github.io/effectivescala/>
10. <http://en.wikipedia.org/wiki/Mixin>
11. <http://scalamacros.org/>
12. <http://docs.scala-lang.org/sips/pending/async.html>
13. [http://msdn.microsoft.com/en-us/library/dd321424\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/dd321424(v=vs.110).aspx)
14. [http://msdn.microsoft.com/en-us/library/hh873173\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/hh873173(v=vs.110).aspx)
15. <http://omgo.wordpress.com/2010/06/07/scala-call-by-name-vs-call-by-type/>
16. <http://www.drmaciver.com/2008/03/an-introduction-to-implicit-arguments/>
17. <http://clojure.github.io/core.async/>
18. [http://en.wikipedia.org/wiki/Static\\_single\\_assignment\\_form](http://en.wikipedia.org/wiki/Static_single_assignment_form)
19. <http://blog.fokus.me/2013/07/17/an-introduction-to-deep-code-walking-macros-with-clojure/>
20. <http://www.ccs.neu.edu/racket/pubs/icfp10-thf.pdf>
21. <https://github.com/clojure/core.typed>
22. <http://cloud.github.com/downloads/frenchy64/papers/ambrose-honours.pdf>
23. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.9203>
24. [http://en.wikipedia.org/wiki/Hindley%E2%80%93Milner\\_type\\_system#Algorithm\\_W](http://en.wikipedia.org/wiki/Hindley%E2%80%93Milner_type_system#Algorithm_W)
25. <http://thinkrelevance.com/blog/2013/10/08/ambrose-bonnaire-sergeant-cognicast-episode-042>

---

### **7.11.6 That's no barrier to my understanding (2013-11-27 07:07)**

It's all very well knowing the theory of [1]multi-core cache consistency and the associated compiler and hardware memory models on various languages and processors (which make it necessary to insert memory barriers), but there's nothing like seeing real examples. The [2]Preshing on Programming blog has some great posts showing consistency problems happening when doing lock free programming on an [3]iPhone and on [4]x86. Some of the comments on this second post point to the [5]diy tool from Inria which lets you try out assembler sequences to determine the effects, allowing one to check scenarios quickly and automatically on different processor models.

A couple of other interesting articles this week. [6]This one discusses the reasons behind zero indexed arrays, and also links to [7]this email message which points to a tool for time travel debugging which works by hooking and logging entry to the kernel, allowing these calls to be replayed later while the process runs under the debugger.

1. [http://en.wikipedia.org/wiki/MESI\\_protocol](http://en.wikipedia.org/wiki/MESI_protocol)
  2. <http://preshing.com/>
  3. <http://preshing.com/20121019/this-is-why-they-call-it-a-weakly-ordered-cpu/>
  4. <http://preshing.com/20120515/memory-reordering-caught-in-the-act/>
  5. <http://diy.inria.fr/>
  6. <http://exple.tive.org/blarg/2013/10/22/citation-needed/>
  7. <http://lwn.net/1999/0121/a/mec.html>
- 

## **7.12 December**

### **7.12.1 It all seems very functional (2013-12-02 06:30)**

After trying to learn [1]Scala by osmosis as I go through the exercises for the Coursera [2]Principles of Reactive Programming course, I decided to have a quick look for good online tutorials. Fortunately I came across this [3]book. [4]Martin Odersky (the creator of the language) is one of the authors of the book and it is really good read. It really pushes the case for Scala being a truly scalable programming language. Reading this book cleared up a lot of the concepts that I'd been introduced to by example - companion classes for example and the use of implicits. The book also covers the idea of traits really well, and contains a number of fairly large examples which show some great features of the language. The book is a little old so doesn't cover some of the more recent language features, but it is certainly a really good foundation.

While I was looking for example code in Scala, I came across [5]MetaScala, a cutdown JVM implemented in Scala. Once the reactive programming course finishes, I'm going to have a

close look at it. The Scala book discusses the flow based type inference that the language uses... this week I also came across [6]this discussion of type flow (for optimisation purposes) in the SBCL Common Lisp implementation.

I'm also going to be attending [7]CodeMesh in London this week, so I've started to select talks on the various tracks. One that looks good is the History of Lazy Functional Programming Languages by David Turner. Lots of the material will probably come from [8]this recent paper. [9]SASL holds a special place in my memory. It was the first lazy functional programming language that I came across, and the clever compilation scheme into [10]SKI combinators was amazingly clever. I never got to use it and didn't get to program in a lazy functional language until I came across Haskell – previously I had used the lazy [11]LispKit Lisp but that didn't have the algebraic data types of languages like [12]Hope.

While we're on the subject of lazy functional languages, [13]this paper discusses some of the "space leak" issues that are often associated with laziness, though it points out that non-lazy languages can also have issues when closures are used. One interesting point made by the paper is that the garbage collector can be used to force the evaluation of cheap call-by-need thunks to avoid the leaks.

CodeMesh has a load of other interesting talks which I'll write up when I get back.

In other news, there was some interesting [14]news of potential compilation of managed code into native, and [15]a quick introduction to the ARM architecture.

1. [http://en.wikipedia.org/wiki/Scala\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Scala_(programming_language))
2. <https://www.coursera.org/course/reactive>
3. <http://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=2&cad=rja&ved=0CEkQFjAB&url=http%3A%2F%2Fwww.cs.ucsb.edu%2Fbenh%2F162%2FProgramming-in-Sca>
4. [http://en.wikipedia.org/wiki/Martin\\_Odersky](http://en.wikipedia.org/wiki/Martin_Odersky)
5. <https://github.com/lihaoyi/Metascala>
6. <http://www.pvk.ca/Blog/2013/11/22/the-weaknesses-of-sbcls-type-propagation/>
7. <https://github.com/lihaoyi/Metascala>
8. <http://www.cs.kent.ac.uk/people/staff/dat/tfp12/tfp12.pdf>
9. <http://www.eis.mdx.ac.uk/staffpages/dat/saselman.pdf>
10. <http://content.lib.utah.edu/utils/getfile/collection/uspace/id/3102/filename/image>
11. [http://en.wikipedia.org/wiki/Lispkit\\_Lisp](http://en.wikipedia.org/wiki/Lispkit_Lisp)
12. [http://en.wikipedia.org/wiki/Hope\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Hope_(programming_language))
13. <http://queue.acm.org/detail.cfm?id=2538488>
14. <http://blogs.microsoft.co.il/sasha/2013/11/25/speculating-about-microsoft-project-n/>
15. <http://blogs.msdn.com/b/ntdebugging/archive/2013/11/22/understanding-arm-assembly-part-1.aspx>

---

### 7.12.2 Meshing ideas (2013-12-09 06:45)

I spent two days last week at [1]CodeMesh in London. This was a thoroughly entertaining conference, with two keynotes and six talks on each of the two days. There was an additional tutorial day on the day before the main conference but I didn't go to that.

It's hard to put my finger on exactly what CodeMesh is. It is tagged as the alternative programming conference, but what does that mean? From my point of view, it took ideas that are starting to make their way into the mainstream, say functional-first programming, immutable data, data as streams and lightweight processes as a means of handling failure, and showing how they have been used successfully and why they may be useful in the future. It was very interesting conference which pushed a lot of ideas. Lots of the talks were good quality, and there were talks that ranged from industrial use of the ideas to talks of a more academic nature. The language emphasis was on languages like Erlang, Haskell, Scala and Clojure, most of which mix functional ideas into a more standard OO setting.

Some of the slides are available on the [2]individual speaker's page, and the videos will come online over the coming months. Hence most of what I write below comes from memory a couple of days after the conference, so some of the information is a little sparse.

[3]Day one started with a keynote on [4]Going Reactive. This really set the scene for the conference. The world is moving to multi-core scalable reliable applications, and this demands access to many ideas that were present in languages like Erlang (which were designed for this use case). The talk pushed the Actor model (and the [5]Akka framework), pointing out that we need to embrace failure by basing these new distributed system on failure domains (such as lightweight processes). The talk was informative and exciting.

I followed this with [6]Snake Bitten, Bruce Tate's thesis being that a language is popular often because it solves one particular issue (say deployment as in the case of Java) with complete disregard for the other dimensions (such as correctness, learnability or elegance). This was just the type of things us functional programmers wanted to hear,

[7]Apache CouchDB Internals was next and I was hoping to get lots of information about the internals of CouchDB. Unfortunately, it didn't live up to my hopes.

Cliff Click's talk on [8]An API For Distributed Computing was informative, stressing as it did that the way to handle large datasets is to split them in-memory across multiple nodes, and then to run the map-reduce inside these nodes, keeping inter-node communication down to a minimum. The API for these large vectors is implemented in Java.

[9]End to end Reactive Programming at Netflix was really just an introduction to Reactive programming. The talk was really good, but I didn't learn anything more than I've seen before – certainly nothing was mentioned about the deployment of this technology at Netflix.

[10]Erjang – With the JVM under the hood was the best talk of the conference. It discussed how the speaker has written a system that takes the Erlang BEAM bytecode format and converts it to run on the JVM. It talked about the implementation, giving examples of the translation of various functions and also showed that the JVM is able to run such code very efficiently, with the only problem being the scheduling of the lightweight processes (IRC the usual Erlang VM is single threaded with the scheduler swapping lightweight processes after a certain number of instructions).

Day two started with another keynote, [11]50 Shades of Green, which started out with a discussion of the [12]Sapir-Whorf hypothesis and then moved on to the richness of expression in programming languages. Along the way we got to see demonstrations of macros inside the Elixir language.

[13]Querying Events by Greg Young was a very good talk. He talked about some map-reduce functionality in his company's [14]product but at the same time talked about the functional ideas behind event sourcing. He pushed the notion of an event store where events are never deleted, and emphasised how easy it makes testing and coding when the past and future are

treated the same. Information is made available to processors as streams – this included using streams for indexes (where the contained values are references to other events) and fixing erroneous event store entries by having a second stream that fixes the first. Additionally, snapshots of states (after processing a sequence of events) are also treated as a stream. It was clever to have a single abstraction, and his idea of an event processor being simply a pure function of state X event -> state, meant that errors can be easily debugged by simply replaying the function on the state before the function executed... they use JavaScript as the language that they run on multiple nodes and debugging consists of simply attaching Chrome to V8 as it replays the execution.

[15]Meta-programming in Logic Programming was an interesting talk on writing a meta-interpreter and using it to prevent a non-terminating tree search. I'd seen these kinds of things in meta-circular Lisp interpreters in the past, but never seen them applied to logic languages. It was interesting to see miniKanren, a logic programming language embedded into Scheme, and there was a good discussion of how these ideas had been used to allow one of the speakers to define a type checker using straightforward declarative logic rules, and then using meta-interpretation to get more information about the type errors that were found in programs [really I guess we were seeing how a DSL could be interpreted in multiple ways where the semantics of the DSL were implemented in the semantics of the original logic program]

[16]Idris: Programming with dependent types was a great introduction to dependent types by a lively speaker. He started with the standard vectors with length type examples, and then used the theorem proving side of dependent types to get the Idris system to write the code. There was a little discussion of using dependent types to track effects, and he showed that we had a real programming language here, by showing a basic implementation of space invaders. Dependent types are something I will be reading about a lot more.

[17]Akka in heterogeneous environments was the speaker writing a Scala frontend (using Akka) which communicated with a C++ vision processing library using RabbitMQ. It emphasised how easy it was to write actor code in Scala (running on the JVM).

[18]Practicing at the cutting edge. Martin talked through his job history and discussed how performance has changed over the years. For today's truly high performance systems, all data needs to be in memory to avoid the massive overhead of going to the disk. Indeed, he stated that a processor to processor message can be achieved in 10 microseconds, so often it is quicker to get data from another processor's memory, than having to go to the disc to fetch information from the data set. Equally, rather than transactionally writing to a disc, some systems write multiple in-memory copies to other nodes, which then write to disc at some later point. With enough replication, persistent changes are effectively guaranteed. An interesting talk, though it wasn't clear how close to these high performance boundaries usual applications are running.

[19]Clojure Core.Async. This was essentially the speaker running the standard demonstration script for Core.Async. He went through channels, covered how you put and get to these channels, and then covered the Go blocks which use macros to transform the code so that the current thread is released until the channel is in a state that would allow the operation to complete without blocking. After a discussion of the different types of channel, the speaker showed everything running inside a ClojureScript demo. The Go block non-blocking co-routines allow the generated JavaScript to simulate a large number of logically concurrently running blocks.

A very interesting conference with lots of interesting talks. Well worth attending!

1. <http://www.codemesh.io/>

2. <http://www.codemesh.io/#speakers>

3. <http://www.codemesh.io/#wednesday>
  4. <http://www.codemesh.io/#wednesday>
  5. <http://akka.io/>
  6. #bruce-tate
  7. <http://www.codemesh.io/#klaus-trainer>
  8. <http://www.codemesh.io/#cliff-click>
  9. <http://www.codemesh.io/#jafar-husain>
  10. <http://www.codemesh.io/#kresten-krab-thorup>
  11. <http://www.codemesh.io/#jose-valim>
  12. [http://simple.wikipedia.org/wiki/Sapir-Whorf\\_hypothesis](http://simple.wikipedia.org/wiki/Sapir-Whorf_hypothesis)
  13. <http://www.codemesh.io/#gregory-young>
  14. <http://geteventstore.com/>
  15. <http://www.codemesh.io/#william-byrd>
  16. <http://www.codemesh.io/#edwin-brady>
  17. <http://www.codemesh.io/#jan-machacek>
  18. <http://www.codemesh.io/#martin-thompson>
  19. <http://www.codemesh.io/#timothy-baldridge>
-



# 2014

## 8.1 January

### 8.1.1 Sharp after all these years (2014-01-06 06:51)

The turning of a new year is traditionally the time to take stock of how things have gone, so it was interesting to see a number of articles on C # and its future were posted in December.

Firstly, [1]Mads Torgerson talked about [2]Roslyn on .NET rocks where he stressed that the Roslyn project, after five years of effort, is getting close to having a managed compiler ready for general release. Internally, after VS2013 was released, teams have now moved over to using the new managed compiler, so it is getting a lot of internal testing at Microsoft. Torgerson also did a talk on the coming features for C # 6 at the [3]London NDC, which is covered in [4]this post by Werner Moise.

Just before that post, Moise had also posted on the work that Joe Duffy and team had been doing on [5]isolated types, linking to some papers and patent applications for extensions to type systems which can be used to make code more easy to parallelise.

Just after Xmas, Joe Duffy himself posted the start of a discussion of how C # has been used as the [6]basis for a systems programming language which has led to some [7]interesting comments on lambda the ultimate and [8]Hacker News. There have been various suggestions that Duffy is really talking about the M # programming language underlying the [9]Midori operating system and that the movement of this group into the main OS division at Microsoft is going to mean that a lot more details are going to come out soon about the project. The work they are doing is really fascinating, and I hope that more details are forthcoming.

Sadly there were no fixed dates for when Roslyn is going to be more freely available, but given the length of time since the [10]last CTP, a release must be coming out soon.

And while we in the subject of C #, I thought this was an interesting explanation of the way the CLR uses the [11]GAC and [12]this post is an interesting exercise in getting into the fine print of the language specification.

1. <http://blogs.msdn.com/b/madst/>
2. <http://www.dotnetrocks.com/default.aspx?ShowNum=935>
3. <http://www.ndc-london.com/>
4. <http://wesnerm.blogs.com/net undocumented/2013/12/mads-on-c-60.html>
5. <http://wesnerm.blogs.com/net undocumented/2013/12/immutable-isolated-types-highly-likely-in-future-c.html>
6. <http://joeduffyblog.com/2013/12/27/csharp-for-systems-programming/>

7. <http://lambda-the-ultimate.org/node/4862#comment>
  8. <https://news.ycombinator.com/item?id=6974494>
  9. <http://www.zdnet.com/microsofts-midori-the-m-connection-7000024664/>
  10. <http://msdn.microsoft.com/en-gb/vstudio/roslyn.aspx>
  11. <http://blogs.msdn.com/b/abhinaba/archive/2013/12/11/net-loading-native-ngen-images-and-its-interaction-with-the-gac.aspx>
  12. <https://communities.coverity.com/blogs/development-testing-blog/2013/12/04/c-method-type-inference-with-multiple-interface-implementations>
- 

### 8.1.2 You don't have to be a compiler to manipulate a syntax tree (2014-01-13 06:52)

There was [1]an interesting post this week which talked about using macros with JavaScript to implement new language features. In the post, the author references a [2]GitHub project where he has been implementing some of the language features of ES6. Of course, JavaScript the language doesn't offer macros, so this work is all based on [3]sweet.js, a Mozilla project which takes JavaScript with macros and transcompiles it down to straight JavaScript. For a sense of the kind of transformations you can do, take a look at the documentation [4]here.

Having used macros a lot in my days as a Common Lisp programmer, I'm a fan. They are certainly a good way to allow you to author an [5]internal [6]DSL and then get the compiler to transform it down into the base language. Keeping everything in the same language means that you can use your usual debugging tools both to debug the transformation and to debug the code that is produced. In a Common Lisp system the compiler can often track how code in the original DSL is pushed into the final implementation code, and therefore can link them up when you are source code debugging.

The big advantage of languages like Common Lisp and its derivatives like Clojure, are that the syntax tree is presented to the macro function in a form that uses the base datatypes of the language and more specially in the actual datatypes that you, the user, use to express your program – atoms, lists and vectors. You don't get this in other languages, where parsing is a lot more complicated, and here you need to use accessors to navigate around the syntax tree which feels a lot less natural.

The thesis of the post is that macros might well be a better way of adding new language features, when compared to the current method of writing yet another transcompiler which takes a modified version of the language and converts it to JavaScript. Macros seem to be a good way to go, as long as the syntax of the derived language doesn't differ too much from standard JavaScript, which is certainly the case for the internal DSL use case. The post emphasises hygienic macros, but [7]as the documentation shows case macros allow you to take more control of the expansion allowing you to implement things like the classic anaphoric if example.

Macros started in early versions of Lisp, but have been making their way into other languages too. [8]Scala, for example, has had them for some time with a design that fits in well with an [9]emphasis on domain specific languages (though [10]that example doesn't actually use macros).

1. <http://jlongster.com/Stop-Writing-JavaScript-Compilers--Make-Macros-Instead>
  2. <https://github.com/jlongster/es6-macros>
  3. <https://github.com.mozilla/sweet.js>
  4. <http://sweetjs.org/>
  5. <http://philcalcado.com/research-on-dsds/domain-specific-languages-dsds/internal-dsds/>
  6. [http://en.wikipedia.org/wiki/Domain-specific\\_language](http://en.wikipedia.org/wiki/Domain-specific_language)
  7. <http://sweetjs.org/>
  8. <http://scalamacros.org/paperstalks/2013-04-22-LetOurPowersCombine.pdf>
  9. <http://www.scala-lang.org/old/node/1403>
  10. <https://github.com/fogus/baysick/blob/master/src/fogus/baysick/Baysick.scala>
- 

### 8.1.3 One at a time (2014-01-16 06:56)

[1]Akka Concurrency by Derek Wyatt

I thought this was a good introductory book on the [2]Akka library and its implementation of [3]Actors. The book has a chatty humorous style which remains just to the right side of being annoying. It starts simple with a brief discussion of the difference between concurrency and parallelism, and then walks through various features of the Akka library, demonstrating the various concepts by building up a simulation of an aircraft. In this simulation, the Actors range from the altimeter and the plane itself to the various people on the plane, all of which communicate using Actor based messaging. Usually such books demonstrate the ideas using small self contained examples, and it is refreshing to have a large example which is worked on as we progress through the book.

The book emphasises being able to test Actor systems, with one of the initial chapters explaining how to set up sbt and get unit/integration tests running. The chapter on Actor supervision is really good – it emphasises how, in this model, exceptions are sometimes best handled by tearing down the Actor and restarting it. This means we avoid having to try to fix up corrupted state, but just start again and replay events to the Actor if necessary so that it catches up with the rest of the world.

At the end of the book, there are also chapters on Akka programming patterns, which includes ideas such as the circuit breaker (if something keeps failing, stop doing it for a while) and event bus architectures, and better still a chapter on Akka programming antipatterns which includes notes on ideas such as synchronous blocking being a very bad thing.

The only Akka feature that I would have liked to the book to cover was clustering, but this is a recent addition to Akka and came after the book was written. Apart from that I'd summarise this book as a good easy read with plenty of example code.

As an aside, I recently came across [4]this free time port of some of the Akka ideas to .NET. The [5]code is available on GitHub and includes the SignalR library for doing the remote communication.

1. [http://www.amazon.co.uk/Akka-Concurrency-Derek-Wyatt/dp/0981531660/ref=sr\\_1\\_1?ie=UTF8&qid=1389389835&sr=8-1&keywords=akka+concurrency](http://www.amazon.co.uk/Akka-Concurrency-Derek-Wyatt/dp/0981531660/ref=sr_1_1?ie=UTF8&qid=1389389835&sr=8-1&keywords=akka+concurrency)
2. <http://akka.io/>

3. [http://en.wikipedia.org/wiki/Actor\\_model](http://en.wikipedia.org/wiki/Actor_model)
  4. <http://rogeralsing.com/2014/01/01/pigeon-akka-actors-for-net/>
  5. <https://github.com/rogeralsing/Pigeon>
- 

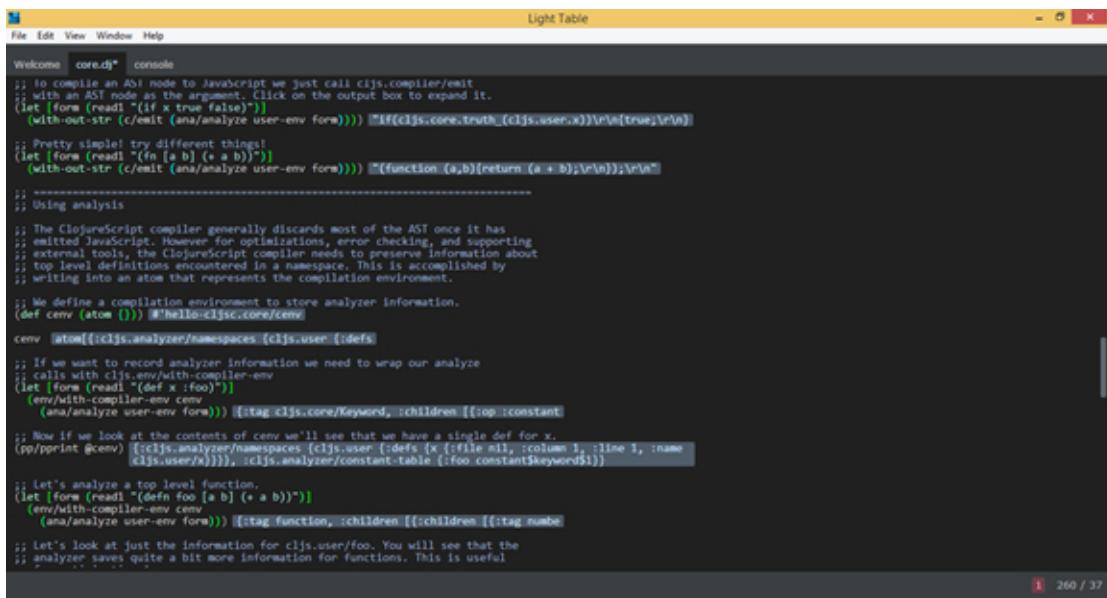
## 8.1.4 It's all about the script (2014-01-20 06:21)

Two JavaScript related articles caught my eye this week.

[1]ClojureScript seems to getting a lot of traction, with even some large applications like [2]LightTable being written using it. The advantages are that you can program using [3]Clojure, a LISP which makes the user highly productive, while at the same time not being forced to run the resulting code on the Java platform. Indeed, as the Clojure is compiled down into JavaScript, one can use the various packaging technologies that allow you to wrap this JavaScript into an application that hosts an HTML5 pane, and hence produce a rich portable GUI native “application”.

Of course it's interesting to know how the Clojure code is compiled into JavaScript, and so this GitHub project, [4]hello-cljsc, is a good introduction to the ClojureScript compiler. What makes this unique is that the overview is presented as a REPL session, where the comment explains what you are going to see, and then you can actually execute some code which demonstrates the idea. This seems like a great way of teaching which encourages the user to start with the example code, and they can try slight modifications to see what happens.

[5]



```

File Edit View Window Help
Welcome core.cljs*  console
;; To compile an Abi node to JavaScript we just call cljs.compiler/emit
;; with an AST node as the argument. Click on the output box to expand it.
(let [form (read! "(if x true false)")]
  (with-out-str (c/emit (ana/user-env form))) "((if(cljs.core.truth_(cljs.user/x))true)\n"
;; Pretty simple! Try different things!
(let [form (read! "(fn [a b] (+ a b))")]
  (with-out-str (c/emit (ana/analyze user-env form))) "((function (a,b){return (a+b);})\n\n"
;-----
;; Using analysis
;; The ClojureScript compiler generally discards most of the AST once it has
;; emitted JavaScript. However, for optimization, error checking, and supporting
;; external tools, the ClojureScript compiler needs to preserve information about
;; top level definitions encountered in a namespace. This is accomplished by
;; writing into an atom that represents the compilation environment.
;; We define a compilation environment to store analyzer information.
(def cenv (atom {})) #TheHello.cljs.core/cenv
cenv @atom[[cljs.analyzer/namespaces {cljs.user} {defs}
;; If we want to record analyzer information we need to wrap our analyze
;; calls with cljs.env/with-compiler-env
(let [form (read! "(def x :foo)")]
  (env/with-compiler-env cenv
    (ana/analyze user-env form))) [:tag cljs.core/Keyword, :children [{:op :constant
;; Now if we look at the contents of cenv we'll see that we have a single def for x.
(pp/pprint @cenv) [:cljs.analyzer/namespaces {cljs.user} {defs {x {:file nil, :column 1, :line 1, :name
  cljs.user/x}}}, cljs.analyzer/constant-table {:foo constant$keyword$1}]}
;; Let's analyze a top level function.
(let [form (read! "(defn foo [& b] (+ a b))")]
  (env/with-compiler-env cenv
    (ana/analyze user-env form))) [:tag function, :children [{:children [{:tag numbe
;; Let's look at just the information for cljs.user/foo. You will see that the
;; analyzer saves quite a bit more information for functions. This is useful
;-----
```

I also came across [6]esprima, a library for parsing and analysing JavaScript, and [7]Regenerator, a library that uses it to implement ES6 generators by converting them into standard JavaScript code.

Generators can be used to do all sort of interesting things, including rewriting code to pause at the yield points, hence allowing you to write a single stepping debugger by doing a source

to source transform. I found [8]this post that describes the technique, which is along the lines of CPS transform ideas that can also be used to implement languages which allow the hosting runtime control of the running code.

JavaScript to JavaScript transforms seem to be used all over the place these days. This is much easier to do than the traditional byte code manipulation, and source maps allow the old and new code to linked together when you are debugging, so this technique can lead to interesting places.

1. <http://clojure.org/clojurescript>
  2. <https://github.com/LightTable/LightTable>
  3. <http://clojure.org/>
  4. <https://github.com/swannodette/hello-cljsc>
  5. <http://clivetong.files.wordpress.com/2014/01/clojurescripttutorial.png>
  6. <https://github.com/ariya/esprima/tree/harmony>
  7. <http://facebook.github.io/regenerator/>
  8. <http://amasad.me/2014/01/06/building-an-in-browser-javascript-vm-and-debugger-using-generators/>
- 

### 8.1.5 Choose wisely (2014-01-20 06:57)

[1]Zermelo's Axiom Of Choice: Its Origins, Development and History by Gregory H Moore

The [2]axiom of choice is a fascinating subject which is used in most undergraduate mathematics classes as a means for justifying [3]Zorn's lemma which is in turn used to prove a number of theorems. I remember finding it really confusing when I first heard the witty remark (by Russell I think) "given an infinite set of pairs of shoes it's easy to choose a member of each pair, but not for an infinite set of pairs of socks".

This book covers the history of the axiom. [4]Zermelo formulated his well ordering principle around the time that mathematicians were investigating set theory and its application to the foundation of mathematics. It took a long time before people accepted that there was a need for infinite choices in a mathematical proof and there were many discussions about whether such a principle was needed in a given proof. Before the formulation of first order logic, and its use for formulating set theory, fairly famous mathematicians were on opposing sides during the debate, and the book covers this social aspect of mathematical discovery.

A fairly long read which is a little dry in places, but a very interesting read. As an appendix it lists loads of famous theorems and whether they are equivalent or are weaker with and without the use of the axiom of choice.

1. [http://www.amazon.co.uk/Zermelos-Axiom-Choice-Development-Mathematics/dp/0486488411/ref=sr\\_1\\_1?ie=UTF8&qid=1389427583&sr=8-1&keywords=zermelos+axiom+of+choice](http://www.amazon.co.uk/Zermelos-Axiom-Choice-Development-Mathematics/dp/0486488411/ref=sr_1_1?ie=UTF8&qid=1389427583&sr=8-1&keywords=zermelos+axiom+of+choice)
2. [http://en.wikipedia.org/wiki/Axiom\\_of\\_choice](http://en.wikipedia.org/wiki/Axiom_of_choice)
3. [http://en.wikipedia.org/wiki/Zorn%27s\\_lemma](http://en.wikipedia.org/wiki/Zorn%27s_lemma)
4. [http://en.wikipedia.org/wiki/Ernst\\_Zermelo](http://en.wikipedia.org/wiki/Ernst_Zermelo)

---

### **8.1.6 You GIT (2014-01-23 06:59)**

[1]Version Control With GIT: Powerful Tools and Techniques for Collaborative Software Development by Jon Loeliger and Matthew McCullough

I've started using GIT at work and needed a book to get up to speed with it. Moreover I wanted a book that both explained how GIT works under the covers, and also gave advice on using it for standard problems that I am going to encounter.

This book does all of these things. It explains how GIT actually works, encouraging you to work through some examples of repository creation, and then getting you to create and modify some files and then look at the file system to see how the directory structure and file contents are represented using hash named files. It covers how these are stored in the staging area. In later chapters it covers merging and the various techniques that GIT employs to do these merges, and also remote refs and how they are used to allow the sharing of repositories across machines.

The book is full of useful advice, often showing the commands to set up a scenario where the advice can be used. Before reading it I hadn't realised that GIT offers so many commands, or even that some of them are optional and depend on logging using the reflog.

Well worth a read if you are new to GIT.

1. [http://www.amazon.co.uk/Version-Control-Git-collaborative-development/dp/1449316387/ref=sr\\_1\\_1?ie=UTF8&qid=1389437715&sr=8-1&keywords=version+control+with+git](http://www.amazon.co.uk/Version-Control-Git-collaborative-development/dp/1449316387/ref=sr_1_1?ie=UTF8&qid=1389437715&sr=8-1&keywords=version+control+with+git)
- 

### **8.1.7 It will be my legacy (2014-01-28 06:01)**

[1]Working Effectively With Legacy Code by Michael C Feathers

This is a really interesting book, which essentially talks about how to use more modern techniques, such as unit testing and refactoring, in a code base that wasn't written that way, which the author terms legacy code.

The book has several sections. Part one covers the mechanics of change and was a fantastic read. It looks at the kinds of tooling that might be useful for working with such code, but for me the key ideas were sensing, separation and seams. Sensing is mocking plus the possible introduction of extra code and fields which allow a test to determine that something happened. Separation is the idea of being able to take some code and run it in a test harness, potentially faking or mocking the environment in which it usually runs. Seams are the places where you can introduce new code, perhaps by intercepting a method call and making it do something else in a test.

Part two of the book is a series of short chapters that focus on particular problems and their solutions - for example, "I don't have much time and I need to change it". The chapters are full of good practical advice, but whereas I found it easy to sit down and read part one, I found

it really hard to go through each of these individual recipes and absorb them. I presume they would be better read when you have a problem of the particular type to work on.

Part three covers some dependency breaking techniques and was also really interesting.

In summary, I thought this was a good book, though some parts of it are not really suitable as a sit down and read the whole book. Some neat ideas.

1. [http://www.amazon.co.uk/Working-Effectively-Legacy-Robert-Martin/dp/0131177052/ref=sr\\_1\\_1?ie=UTF8&qid=1389438365&sr=8-1&keywords=working+effectively+with+legacy](http://www.amazon.co.uk/Working-Effectively-Legacy-Robert-Martin/dp/0131177052/ref=sr_1_1?ie=UTF8&qid=1389438365&sr=8-1&keywords=working+effectively+with+legacy)
- 

Daniel S (2014-01-31 12:48:25)

Working Effectively With Legacy Code is THE best programming book I've ever read. It totally changed my perspective on what can be done. I think about the concepts there every single day at work.

### **8.1.8 Use the (Windows) source Luke (2014-01-30 06:24)**

We were discussing the Windows guard pages at work. To cut down on the amount of space that a stack uses, Windows typically reserves the full stack size of virtual memory, but only allocates a chunk of this space, putting a special PAGE\_GUARD flag on the last page. When the application touches this, the system allocates the page and tries to extend the stack to put a new guard page on the next page down.

The question is: what code actually does this extension, and under what circumstances does the behaviour happen? As a user, you can use the virtual memory allocation and protection functions, and allocate arbitrary pages with the GUARD flag, so what happens if you touch one of these pages.

After some searching, I found a copy of the Windows research version of the kernel [1]here. In that file, the function MiCheckForUserStackOverflow controls what happens - if the guard page lies within the stack of the current thread then the extension behaviour happens. If the stack cannot be extended, when the allocation of a following page fails (for example), a stack overflow exception is raised, and if the guard page doesn't lie inside the stack bounds then an exception is thrown. This behaviour agreed with a small C test program I had written, and which I was going to debug using a kernel debugger... much nicer to use some source code instead though. I hadn't come across the Windows research kernel before, but the code is nicely commented and looks like a useful read.

While searching for an answer, I found this [2]blog, by a Google kernel security engineer. The posts are full of really interesting kernel breaking ideas, and ideas aimed at allowing applications to break out of the sandbox. In particular, this [3]paper and [4]post on using the [5]Bochs emulator to find code where the same location is accessed twice in quick succession via a pointer - if the first time is a check, and the second is an action, then there is a window between the two in which the value can be changed after the check, allowing potential exploits. And also this [6]post on crashing Windows.

All very interesting stuff that I'm going to be exploring in the future.

1. [http://gate.upm.ro/os/LABs/Windows\\_OS\\_Internal\\_Curriculum\\_Resource\\_Kit-ACADEMIC/WindowsResearchKernel-WRK/v1.2/base/ntos/mm/acceschk.c](http://gate.upm.ro/os/LABs/Windows_OS_Internal_Curriculum_Resource_Kit-ACADEMIC/WindowsResearchKernel-WRK/v1.2/base/ntos/mm/acceschk.c)
  2. <http://j00ru.vexillium.org/?p=1594>
  3. <http://vexillium.org/dl.php?bochspwn.pdf>
  4. <http://j00ru.vexillium.org/?p=1880>
  5. <http://en.wikipedia.org/wiki/Bochs>
  6. <http://j00ru.vexillium.org/?p=1767>
- 

## 8.2 February

### 8.2.1 Cast the fun (2014-02-04 06:58)

Another podcast has recently come to my attention - the [1]HaskellCast podcast. The first four episodes are really good, and include discussion with [2]Don Stewart on the practical application of Haskell, [3]Simon Peyton Jones on GHC, the main Haskell implementation, and Simon Marlow on the work he is doing at Facebook.

In the first episode, with [4]Edward Kmett, there was plenty of discussion about [5]lenses, a pattern for making copies of immutable potentially deeply nested data structures while at the same time modifying some of the contents. This reminded me a lot of a generalised kind of [6]Zipper. There is a tutorial [7]here and an associated slide presentation [8]here.

Haskell is full of interesting ideas, and it is fascinating to see all of the practical uses of what once seemed a more research kind of language. Sometimes, you don't use the full set of facilities in the language - Don Stewart points out that often in practice, the laziness needs to be removed from the program by using strictness annotations, but that the language is great of implementing internal DSLs for the financial world. They've found it quick to train non-programmers for using Haskell to define mathematical models. The purity of the language seems to offer a lot of benefits - deterministic parallelism and "efficient" software transactional memory being two obvious benefits.

I'm looking forward to future editions of the podcast!

1. <http://www.haskellcast.com/>
  2. <http://donsbot.wordpress.com/>
  3. <http://research.microsoft.com/en-us/people/simonpj/>
  4. <https://github.com/ekmett>
  5. <http://lens.github.io/>
  6. [http://en.wikipedia.org/wiki/Zipper\\_\(data\\_structure\)](http://en.wikipedia.org/wiki/Zipper_(data_structure))
  7. <http://lens.github.io/tutorial.html>
  8. <http://comonad.com/haskell/Lenses-Folds-and-Traversals-NYC.pdf>
-

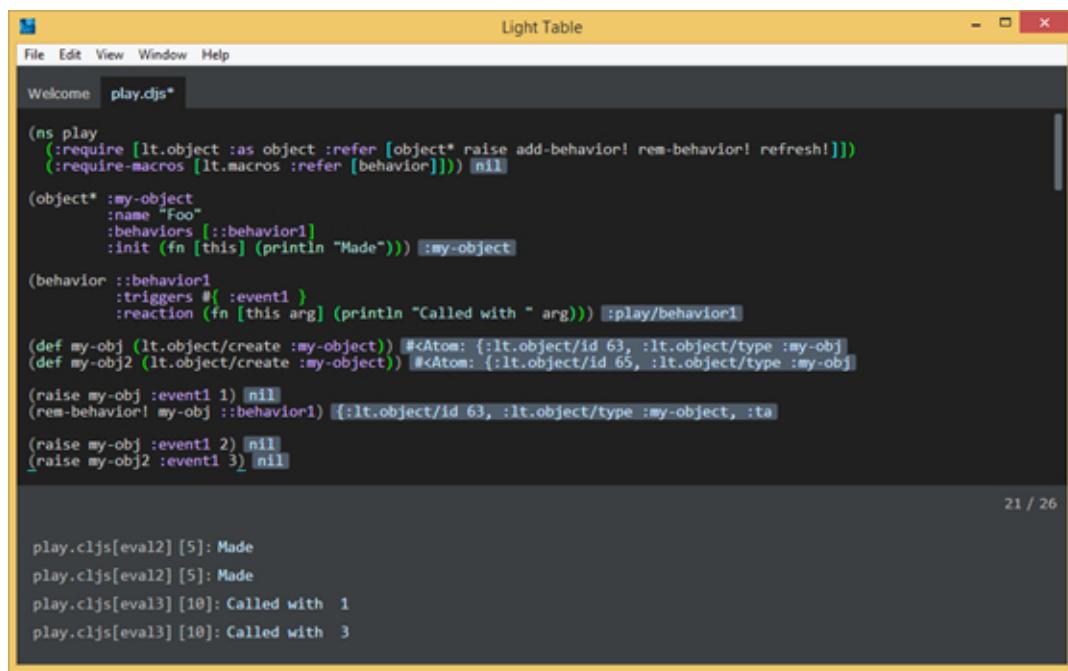
## 8.2.2 Plug table into light (2014-02-18 07:22)

We have a regular Clojure meetup one lunchtime a week at work, and for the last couple of weeks we've been looking at [1]LightTable. LightTable is a programming language IDE written entirely in [2]ClojureScript, a variant of [3]Clojure that can be compiled down to JavaScript. It is written as a hybrid application - the [4]node webkit browser is embedded into a simple native application which simply displays the browser pane and then runs the packaged JavaScript. There are lots of companies using this technology as listed on the wiki [5]here.

The ClojureScript code implements [6]a BOT model – behaviour-object-tag, which is a kind of prototype based object system, though in this system the central focus isn't the objects but is rather the behaviours and the objects to which they are attached. The code for the BOT system is contained in [7]object.cljs file which is included in the [8]LightTable source.

Using this code, we can easily create objects and behaviours and attach the behaviours to the objects. We can attach and remove behaviours dynamically. When an event is raised, the appropriate reaction is called with the passed arguments, though if no reaction is available then things just silently fail.

[9]



The screenshot shows the Light Table IDE interface. The top bar has tabs for 'File', 'Edit', 'View', 'Window', and 'Help'. The main window has a tab labeled 'play.cljs\*' and a status bar showing '21 / 26'. The code editor contains ClojureScript code defining an object 'Foo' with behaviors and reactions. The bottom part of the window shows a REPL history with three printed lines: 'play.cljs[eval2] [5]: Made', 'play.cljs[eval2] [5]: Made', and 'play.cljs[eval3] [10]: Called with 1' followed by 'play.cljs[eval3] [10]: Called with 3'.

```
(ns play
  (:require [lt.object :as object :refer [object* raise add-behavior! rem-behavior! refresh!]])
  (:require-macros [lt.macros :refer [behavior]])) nil

(object* :my-object
  :name "Foo"
  :behaviors [::behavior1]
  :init (fn [this] (println "Made")) :my-object

  (behavior ::behavior1
    :triggers #{:event1}
    :reaction (fn [this arg] (println "Called with " arg)) :play/behavior1

  (def my-obj (lt.object/create :my-object)) #<Atom: {:lt.object/id 63, :lt.object/type :my-obj}
  (def my-obj2 (lt.object/create :my-object)) #<Atom: {:lt.object/id 65, :lt.object/type :my-obj}

  (raise my-obj :event1 1) nil
  (rem-behavior! my-obj ::behavior1) #<Atom: {:lt.object/id 63, :lt.object/type :my-object, :tags #{}}
  (raise my-obj :event1 2) nil
  (raise my-obj2 :event1 3) nil)

play.cljs[eval2] [5]: Made
play.cljs[eval2] [5]: Made
play.cljs[eval3] [10]: Called with 1
play.cljs[eval3] [10]: Called with 3
```

What are these objects? Well, an object is just an atom containing a map that records the information about the behaviour mapping. For example, one of the objects we made above prints as follows in the REPL.

```
#<Atom: {:lt.object/id 65, :lt.object/type :my-object, :tags #{} :content nil, :triggers [], :args nil, :children {}, :listeners {:event1 [:play/behavior1]}, :destroy [:lt.objects.clients/on-destroy-remove-cb]}, :behaviors #{} :name Foo >
```

One thing to be aware of is that the object system is dynamic only to a limited degree. If you make changes you need to refresh! objects for the current definitions to take effect.

For example, here we redefine the behaviour, to trigger it with a new event. The change doesn't take effect until the object is refreshed.

[10]

The screenshot shows the Light Table IDE interface. The top menu bar includes File, Edit, View, Window, Help, and a tab labeled play.djs\*. Below the menu is a toolbar with Welcome and play.djs\* buttons. The main editor area contains the following Clojure code:

```
(behavior ::behavior1
  :triggers #{ :event2 }
  :reaction (fn [this arg] (println "Called new definition" arg))) :play/behavior1
(raise my-obj2 :event2 5) nil
(refresh! my-obj2) nil
(raise my-obj2 :event2 6) nil
```

The bottom terminal window displays the output of the code execution:

```
play.cljs[eval9] [26]: Called new definition 6
27 / 1
```

It's a very clever system through. Behaviours can encapsulate functionality which can be used across multiple unrelated object types.

[11]This tutorial shows how you can define a simple plugin. It uses the defui macro, which can be used to generate HTML elements for fairly complicated UI elements, and shows how you can use the tab manager to add these elements into the existing GUI.

We were more interested in modifying the behaviour of the existing GUI than adding new separated GUI elements, and the documentation points to the [12]DeClassifier plugin as a demonstration of how to go about this. [13]DeClassifier defines some commands for checking the current editor buffer and uses a [14]behaviour definition to hook them into the existing system.

```
{:+ {:app [(:lt.objs.plugins/load-js "declassifier _compiled.js" true)]
:editor [(:lt.objs.editor/on-change :declassify-behind-cursor)] }
```

The `:+` adds new behaviours, and there's a `:-` for removing them. The name (like `:editor`) finds the relevant objects using the `lt.object/by-tag` function.

`(lt.object/by-tag :app)` -> a sequence containing the single app instance in the system. This is the top level control object.

`(lt.object/by-tag :editor)` -> a sequence of the existing editors, one per loaded file

Looking at the listeners of one of the editor instances using

```
(println (:listeners (deref (first (lt.object/by-tag :editor)))))
```

we can see the triggers and their subscriptions.

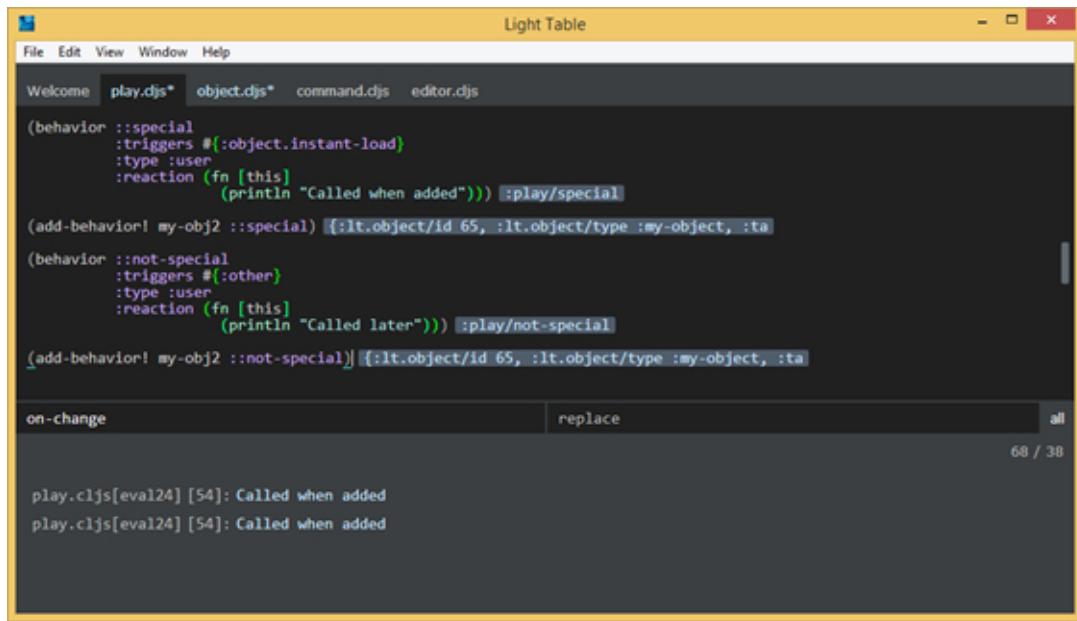
```
{:editor.eval.cljs.result.replace [(:lt.plugins.clojure/cljs-result.replace)],
:editor.exception [(:lt.objs.eval/inline-exceptions)],
:watch [(:lt.plugins.watches/eval-on-watch-or-unwatch)],
:editor.eval.clj.no-op [(:lt.plugins.clojure/no-op)],
:keymap.diffs.user+ [(:lt.objs.settings/user-keymap-diffs)],
:editor.eval.clj.print [(:lt.plugins.clojure/eval-print)],
:deactivated [(:lt.objs.style/remove-theme)],
:change [(:lt.objs.editor/file/dirty-on-change)
(:lt.objs.editor/on-change :declassify-behind-cursor)
(:lt.plugins.auto-complete/intra-buffer-string-hints)],
... }
```

Hence the change trigger on the editor will react using the in-change behaviour in [15]this file. This fires the command that is passed in as an argument as expected.

The more interesting behaviour is the first one, which seems to be responsible for loading the JavaScript file that is passed as the parameter. You might expect this to happen at the time that the plugin is loaded, but how does it get triggered.

It took a while to find the answer. There is a special trigger object.instant-load, which causes the reaction to happen when an object associated with it changes – for example, a behaviour is added or freed. In the following example we can add this tag on one of our own behaviours, and then see that the reaction fires when a behaviour is added. Contrast this with the ::not-special behaviour which is not specially tagged, and hence executes only when the trigger event happens.

[16]



```
(behavior ::special
  :triggers #{:object.instant-load}
  :type :user
  :reaction (fn [this]
    (println "Called when added")) :play/special)

(add-behavior! my-obj2 ::special) [:lt.object/id 65, :lt.object/type :my-object, :ta]

(behavior ::not-special
  :triggers #{:other}
  :type :user
  :reaction (fn [this]
    (println "Called later")) :play/not-special)

(add-behavior! my-obj2 ::not-special) [:lt.object/id 65, :lt.object/type :my-object, :ta]

on-change replace all
68 / 38

play.cljs[eval124] [54]: Called when added
play.cljs[eval124] [54]: Called when added
```

This is used by the load-js behaviour on the app which is defined as part of the plugin definition file ([17]as the last line).

[18]

```
(behavior ::load-js
  :triggers #{:object.instant-load}
  :desc "App: Load a javascript file"
  :params [{:label "path"}]
  :type :user
  :reaction (fn [this path]
```

It's really clever that the editor is customised by adding and removing behaviours, which is in turn by reading and executing the configuration files ([19]like the standard one). This data driven configuration looks very powerful and seems to work well inside LightTable. I guess the question is, how easy is it to use this mechanism if order starts being important when we are executing a number of reactions?

1. <https://github.com/LightTable/LightTable>
2. <https://github.com/clojure/clojurescript>
3. <http://clojure.org/>
4. <https://github.com/rogerwang/node-webkit/>

5. [https://github.com/rogerwang/node-webkit/wiki/List-of-apps-and-companies-using-node-webkit](https://github.com/rogerwang/node-webkit/wiki>List-of-apps-and-companies-using-node-webkit)  
6. <http://www.chris-granger.com/2013/01/24/the-ide-as-data/>  
7. <https://github.com/LightTable/LightTable/blob/master/src/lt/object.cljs>  
8. <https://github.com/LightTable/LightTable>  
9. <http://clivetong.files.wordpress.com/2014/02/objectandbehaviours.png>  
10. <http://clivetong.files.wordpress.com/2014/02/refreshneeded.png>  
11. <http://blog.jakubarnold.cz/light-table-plugin-tutorial>  
12. <https://github.com/LightTable/Declassifier>  
13. <https://github.com/LightTable/Declassifier/blob/master/src/lt/plugins/declassifier.cljs>  
14. <https://github.com/LightTable/Declassifier/blob/master/declassifier.behaviors>  
15. <https://github.com/LightTable/LightTable/blob/master/src/lt/objs/editor.cljs>  
16. <http://clivetong.files.wordpress.com/2014/02/specialtriggers.png>  
17. <https://github.com/LightTable/LightTable/blob/master/src/lt/objs/plugins.cljs>  
18. <http://clivetong.files.wordpress.com/2014/02/loadjs.png>  
19. <https://github.com/LightTable/LightTable/blob/master/deploy/settings/default/default.behaviors>

---

### 8.2.3 Should I do them concurrently or in parallel? (2014-02-24 07:06)

[1]Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming by Simon Marlow

This was a fascinating read. Haskell itself has a reputation for being very high level and abstract, and, of course, this book uses some very high level concepts to achieve multithreaded execution and deterministic parallelism. However the author was one of the main implementers for the runtime system of GHC, and this comes across in the writing – from the use of [2]ThreadScope to see exactly where the threads and runtime are spending their time to the brilliant explanation of weak-head normal form, [3]deepseq and the execution semantics using sparks.

[4]GHC has a great number of ways of using multiple threads. The book starts with parallel Haskell, covering rpar, rseq and the [5]Eval monad which allows an algorithm to be parameterised with different evaluation strategies. All the time, the author demonstrates the facility using an example, and often demonstrates how well things are working by measuring the speedup with number of cores that the runtime is allowed to use. Sometimes these measurements point to bottlenecks in the runtime itself and he isn't afraid to point this out.

We move on to dataflow parallelism using the [6]Par monad and then array style parallelism using [7]Repa and Accelerate, a means for converting some Haskell code into [8]CUDA which will run on the GPU.

Next the book moves on to concurrent Haskell.

The author starts by covering threads communicating via [9]MVars (which are like data values protected by a semaphore). He then moves on to discussing [10]Exceptions, a facility of Haskell that I had largely ignored in the past. At this point, things start getting little dirty as we need to start thinking about asynchronous exceptions, and that means that we need to be able to guard regions in order to prevent asynchronous exceptions happening in things like finally blocks which would make it impossible to preserve invariants.

This is the thing that I always found distasteful in C# where thread abort exceptions, which are asynchronously delivered as a result of the Thread.Abort, can be caught but require special code to prevent them propagating after the catch handler has finished.

[11]

```
static void Main(string[] args)
{
    var t = new Thread(Worker);
    t.Start();
    Thread.Sleep(2000);
    t.Abort();
    Console.ReadLine();
}

static void Worker()
{
    try
    {
        Thread.Sleep(10000);
    }
    catch (ThreadAbortException)
    {
        Console.WriteLine("Thread being aborted.");
        //Thread.ResetAbort();
    }

    // This line is only reached if you reset abort above
    Console.WriteLine("Outside the abort");
}
```

There's also the need to prevent the delivery inside finally blocks.

```
static void Main(string[] args)
{
    var t = new Thread(Worker);
    t.Start();
    Thread.Sleep(2000);
    Console.WriteLine("Aborting");
    t.Abort();
    Console.ReadLine();
}

static void Worker()
{
    try
    {
    }
    finally
    {
        Console.WriteLine("Finally started");
        Thread.Sleep(10000);
        Console.WriteLine("At end of finally");
    }

    Console.WriteLine("Outside the finally");
}
```

[12]

The GHC runtime requires the programmer to duplicate these semantics themselves by exposing primitives for controlling asynchronous delivery.

The book then moves on to software transactional memory. It shows how beautiful code can become using this feature. Using retry, for example, allows the STM system to wait until the global state changes before restarting failed computations. Better still, the author points out the possible problems with STM for large transactions where the transaction log access and multiple potential retries can make performance really bad.

The rest of the book is filled with some examples, including a chat server. These examples demonstrate a number of techniques, including a function which can be used as a combinator that races two threads and returns the answer from the first to complete, aborting the other.

The book is a great read, and I think everyone will learn something from it.

1. [http://www.amazon.co.uk/Parallel-Concurrent-Programming-Haskell-Multithreaded/dp/1449335942/ref=sr\\_1\\_1?ie=UTF8&qid=1393146310&sr=8-1&keywords=parallel+and+concu](http://www.amazon.co.uk/Parallel-Concurrent-Programming-Haskell-Multithreaded/dp/1449335942/ref=sr_1_1?ie=UTF8&qid=1393146310&sr=8-1&keywords=parallel+and+concu)
  2. <http://www.haskell.org/haskellwiki/ThreadScope>
  3. <http://hackage.haskell.org/package/deepseq>
  4. [http://en.wikipedia.org/wiki/Glasgow\\_Haskell\\_Compiler](http://en.wikipedia.org/wiki/Glasgow_Haskell_Compiler)
  5. <http://hackage.haskell.org/package/parallel-3.1.0.1/docs/Control-Parallel-Strategies.html>
  6. <https://hackage.haskell.org/package/monad-par>
  7. <http://hackage.haskell.org/package/repa>
  8. <http://en.wikipedia.org/wiki/CUDA>
  9. <http://hackage.haskell.org/package/base-4.6.0.1/docs/Control-Concurrent-MVar.html>
  10. <http://www.haskell.org/haskellwiki/Exception>
  11. <http://clivetong.files.wordpress.com/2014/02/abort.png>
  12. <http://clivetong.files.wordpress.com/2014/02/finally.png>
- 

#### 8.2.4 To infinity from far away (2014-02-27 07:07)

[1]The Man Who new Infinity: Life of the genius Ramanujan by Robert Kanigel

Very little technical detail about his various theorems and contributions, but a great biography of the man and his life, with loads of historical information about India and the [2]British Raj at the end of the 19th century. As a resident of Cambridge, it was great to hear the inter-linking story of [3]G.H Hardy, his time at [4]Trinity College, Cambridge and his reforms of the [5]mathematical Tripos.

The author gets into [6]Ramanujan's character, seeking to explain his motivations and frustrations at not being able to do full time mathematical research, and also seeks to explain Ramanujan's bodily breakdown when he eventually reached Cambridge. In many ways a sad story, but a very interesting read.

1. [http://www.amazon.co.uk/The-Man-Who-Knew-Infinity/dp/0349104522/ref=sr\\_1\\_1?ie=UTF8&qid=1393082024&sr=8-1&keywords=the+man+who+knew+infinity](http://www.amazon.co.uk/The-Man-Who-Knew-Infinity/dp/0349104522/ref=sr_1_1?ie=UTF8&qid=1393082024&sr=8-1&keywords=the+man+who+knew+infinity)
  2. [http://en.wikipedia.org/wiki/British\\_Raj](http://en.wikipedia.org/wiki/British_Raj)
  3. [http://en.wikipedia.org/wiki/G.\\_H.\\_Hardy](http://en.wikipedia.org/wiki/G._H._Hardy)
  4. [http://en.wikipedia.org/wiki/Trinity\\_College,\\_Cambridge](http://en.wikipedia.org/wiki/Trinity_College,_Cambridge)
  5. <http://en.wikipedia.org/wiki/Tripos>
  6. [http://en.wikipedia.org/wiki/Srinivasa\\_Ramanujan](http://en.wikipedia.org/wiki/Srinivasa_Ramanujan)
-

### **8.2.5 Where next? (2014-02-28 07:08)**

[1]In Pursuit of the Travelling Salesman: Mathematics at the Limits of Computation by William J Cook

This is one of those maths books that describes the problem, and then tells the story of the various historical approaches to solving it. The story is fascinating, and in the pre-computer days involved massive amounts of hand calculation using clever tricks to prove lower bounds for various instances of the problem.

Along the way we learn about the P!=NP conjecture, and also learn about the simplex method and integer programming which can be used as a way to get strong lower bounds for various for travelling salesman problems (using so called control zones).

A good interesting read.

- 
1. [http://www.amazon.co.uk/Pursuit-Traveling-Salesman-Mathematics-Computation/dp/0691152705/ref=sr\\_1\\_1?ie=UTF8&qid=1393088766&sr=8-1&keywords=in+pursuit+of+the+tra](http://www.amazon.co.uk/Pursuit-Traveling-Salesman-Mathematics-Computation/dp/0691152705/ref=sr_1_1?ie=UTF8&qid=1393088766&sr=8-1&keywords=in+pursuit+of+the+tra)

## **8.3 March**

### **8.3.1 It's off to war we go (2014-03-31 06:17)**

[1]DogFight: How Apple and Google went to war and started a revolution by Fred Vogelstein

A fairly lightweight book which gives details about the development of the iPhone and the Android platform, telling the story from the perspective of both the Apple and Google teams. For some of the book we do a chapter in turn from each company, though the stories start to intertwine for the most part. Few technical details, but some coverage of the patent battles between the various players including Samsung, and if you haven't followed the story from the beginning, this book is a fairly good way to catch up.

- 
1. [http://www.amazon.co.uk/Dogfight-Apple-Google-Started-Revolution/dp/0007448406/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1395000835&sr=1-1&keywords=dogfight](http://www.amazon.co.uk/Dogfight-Apple-Google-Started-Revolution/dp/0007448406/ref=sr_1_1?s=books&ie=UTF8&qid=1395000835&sr=1-1&keywords=dogfight)

## **8.4 April**

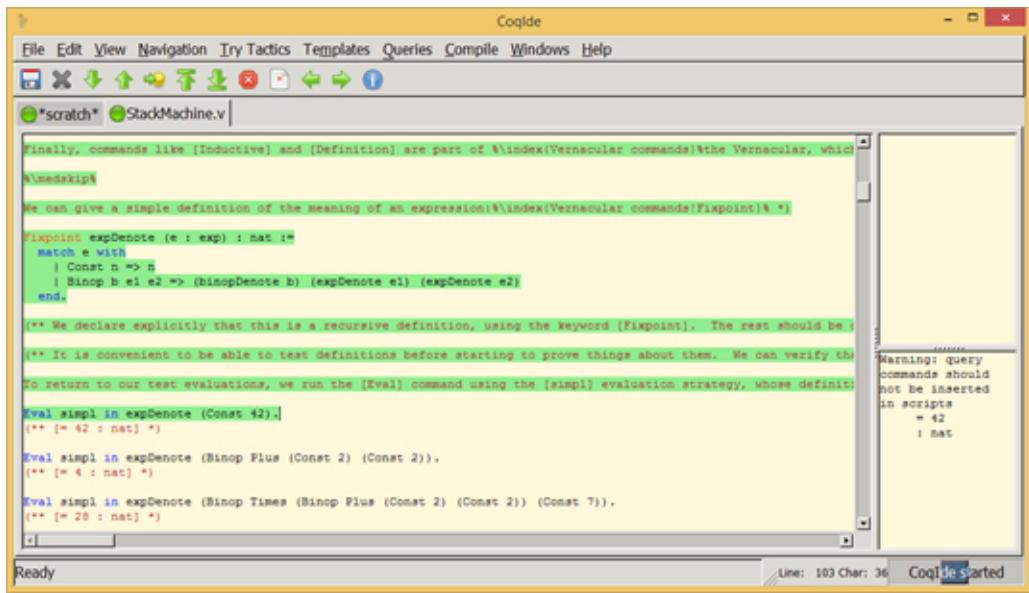
### **8.4.1 That's for sure (2014-04-04 05:29)**

[1]Certified Programming with Dependent Types: A pragmatic introduction to the COQ proof assistant by Adam Chlipala

I really loved this introduction to [2]COQ, a formal proof management system which can be used to formalise and prove mathematical theorems. COQ is based on a constructive type theory which allows you to also write dependently typed programs in its formal language, which can be executed by the COQ system itself. COQ's most famous moment was probably when it was used to [3]formalise the four colour theorem.

This text is available online [4]here, as a series of COQ script files which can be executed and which contain the text of the book as embedded comments. The book itself is really practical, starting quickly with a definition of the semantics of a simple language of expressions over the natural numbers (including times, plus and later comparison), and then proves a the correctness of a compiler that compiles an expression to run on top of a simple stack machine.

[5]



The book then rapidly moves on to give an overview of the whole COQ system. It starts with a description of very basic programming and proving, all the while working through a set of examples. It covers infinite data types and the equivalence of streams. It then moves on to issues around programming with dependent types and more general forms of recursion, covering the tricky constructive concepts of equality and universes.

The final section of the book discusses more proof engineering concepts such as using reflection in proofs, and one chapter covers the Ltac language, allowing a user to invent new proof methods (whilst keeping the core logic consistent).

A brilliant book that covers both the theoretical concepts and also explains them with a stream of useful practical examples. Loved it!

1. [http://www.amazon.co.uk/Certified-Programming-Dependent-Types-Introduction/dp/0262026651/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1395000911&sr=1-1&keywords=certified+pro](http://www.amazon.co.uk/Certified-Programming-Dependent-Types-Introduction/dp/0262026651/ref=sr_1_1?s=books&ie=UTF8&qid=1395000911&sr=1-1&keywords=certified+pro)
2. <https://github.com/coq/coq>
3. <http://research.microsoft.com/en-us/um/people/gonthier/4colproof.pdf>
4. <http://adam.chlipala.net/cpdt/>
5. <http://clivetong.files.wordpress.com/2014/04/coq.png>

---

## 8.4.2 Some odds and ends (2014-04-28 05:52)

I haven't blogged for a while, mainly because I've been busy doing some rather interesting [1]Coursera courses. In fact at the moment I have [2]1, [3]2, [4]3, [5]4 on the go. Bart de Smet's recent comments on Cortana and the underlying use of IQbservable mentioned in the talk linked from this [6]post has got me doing some research using [7]Rx on remoting observable (Rx) queries in preparation for a lightning talk at work.

However, what with a mass of //BUILD videos to get through, and the usual stream of interesting articles, I thought it might be good to list some interesting finds.

Moving data off heap in a managed application is a technique I have read about before. [8]This InfoQ article gives some details about a Java approach.

With the [9]HeartBleed problem, it is instructive to look at the [10]metasploit code which uses the exploit to retrieve the key information. Essentially the code looks into the information that has bled out of the server and tries to use the contents of memory as factors of the key information.

There has been a good Going Deep episode on Channel 9 about [11]the implementation of .NET Native, the recently announced native compiler for .NET (currently in early release and working only for 64 bit windows store applications).

Of the //BUILD talks that have been published, the keynotes are obviously worth watching, and despite the lack on interesting talks in the .NET space, I have enjoyed the following: [12]a discussion of architecture of hybrid cloud approaches, [13]some lessons learned by Microsoft's Azure failures, [14]present and future .NET, [15]C # and Xamarin, [16]Azure service bus and the internet of things plus some of the panel discussions and the overview of universal applications for phone and tablet.

1. <https://www.coursera.org/>
2. <https://class.coursera.org/linearopt-002>
3. <https://class.coursera.org/mathphil-002>
4. <https://class.coursera.org/progfun-004>
5. <https://www.coursera.org/course/malsoftware>
6. <http://channel9.msdn.com/Forums/Coffeehouse/so-Bart-de-Smets-new-project-is-powering-Cortana-with-Rx>
7. <http://rxx.codeplex.com/>
8. <http://www.infoq.com/articles/Open-JDK-and-HashMap-Off-Heap>
9. <http://arstechnica.com/security/2014/04/how-heartbleed-transformed-https-security-into-the-stuff-of-absurd-theater/>
10. [https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/ssl/openssl\\_heartbleed.rb](https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/ssl/openssl_heartbleed.rb)
11. <http://channel9.msdn.com/Shows/Going+Deep/Inside-.NET-Native>
12. <http://channel9.msdn.com/Events/Build/2014/3-632>
13. <http://channel9.msdn.com/Events/Build/2014/3-615>
14. <http://channel9.msdn.com/Events/Build/2014/2-588>
15. <http://channel9.msdn.com/Events/Build/2014/3-653>
16. <http://channel9.msdn.com/Events/Build/2014/3-635>

---

## 8.5 May

### 8.5.1 Hello source, meet filter (2014-05-07 05:31)

I did a [1]lightning talk at work on the subject of IQbservable, the IObservable equivalent of IQueryable. The code I demoed was a standard chat demo from from the [2]Rx project. The author of this project has a great blog post [3]describing the project here.

1. <https://onedrive.live.com/embed?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%213333&authkey=ABXuzzw-IcMf0RY&em=2>
2. <http://rx.codeplex.com/>
3. <http://davesexton.com/blog/post/LINQ-to-Cloud-IQbservable-Over-the-Wire.aspx>

---

Reactive extensions in action | Thirty years tapping a keyboard (2018-04-18 05:46:48)

[...] also noticed that the pre-release System.Reactive Nuget package contains code around the IQbservable interface. It will be interesting to see where that goes in the [...]

### 8.5.2 It's certainly an oK runtime (2014-05-15 05:04)

After a slight delay, [1]the deep dive on the K runtime was published yesterday. What's rather cool about this is that the source code [2]is all available on github as well as a set of [3]starter examples and a more complicated [4]playground of examples. The introduction talk on this technology is [5]available here. There's a good introductory diagram [6]here.

1. <http://channel9.msdn.com/Events/TechEd/NorthAmerica/2014/DEV-B411#fbid=>
2. <https://github.com/aspnet/KRuntime>
3. <https://github.com/aspnet/Home>
4. <https://github.com/aspnet/Entropy>
5. <http://channel9.msdn.com/Events/TechEd/NorthAmerica/2014/DEV-B385#fbid=>
6. <https://github.com/aspnet/Home/wiki/KRuntime-structure>

---

### 8.5.3 A nugget of information (2014-05-19 05:39)

[1]Pro NuGet by Maarten Balliauw and Xavier Decoster

I recently had to write a NuGet package at work, and was fairly impressed with how easy it is to take an assembly and push it into a local package store. Admittedly this was made easier by a number of tasks that were available on the build server, but even so, the reverse process of consuming by pulling packages into a Visual Studio project is also really easy. I had spent a little time looking at the implementation of [2]NuGet, but got interested in this again after seeing how fundamental [3]NuGet is as part of the [4]Katana runtime which Microsoft recently announced.

So it was time to find a book on the subject, and I found this book on Amazon – it's rather expensive but I managed to get a copy of it second hand.

The first thing to point out is that the book is several years old. With something like NuGet which is rapidly evolving, this might mean that it doesn't cover some of the implementation. However, the book itself doesn't go that deep into any particular area, but covers the surface area of what NuGet offers, so I don't think the age matters. There are numerous examples and both the versions of Visual Studio and MVC that are used seem a tiny bit old now.

There's a quick getting started on how to download the required bits and pieces, and then a walkthrough of consuming some packages inside a Visual Studio project. The book then moves on to creating packages, publishing packages (on the Microsoft Nuget gallery) and then covers how to set up your own NuGet server. It is worth pointing out that the Nuget tooling supports the repository being a simple file share, which makes it trivial to start using NuGet in a small deployment.

This coverage includes some examples of simple PowerShell scripts which can be triggered when a package is installed into a solution and when a package is installed into individual projects in the solution, as well as how you add other assets to a project when a NuGet package is associated with it.

The book then moves to a slightly higher level, discussing continuous deployment using packages as the mechanism for pushing changes. This is followed by a chapter on automated deployment which covers Octopus and Chocolatey, tools that use NuGet packages as a means for pushing entire applications (rather than pushing constituent parts of an application).

The last two chapters talk about extending NuGet itself, and about using NuGet as a protocol. In the latter chapter, the worked example is a technique for deploying plugins using NuGet.

At all times the book goes into just enough detail, letting you know what is available and setting you up to go to other sources to get the fine details. As a bonus it is an easy read.

1. [http://www.amazon.co.uk/Pro-NuGet-Experts-Voice-Microsoft/dp/1430241918/ref=sr\\_1\\_2?ie=UTF8&qid=1400395925&sr=8-2&keywords=pro+nuget](http://www.amazon.co.uk/Pro-NuGet-Experts-Voice-Microsoft/dp/1430241918/ref=sr_1_2?ie=UTF8&qid=1400395925&sr=8-2&keywords=pro+nuget)
2. <http://nuget.codeplex.com/>
3. <http://www.hanselman.com/blog/IntroducingNuGetPackageManagementForNETAnotherPieceOfTheWebStack.aspx>
4. <http://www.asp.net/vnext>

## 8.6 June

### 8.6.1 You need a load of extra functionality to support that (2014-06-02 05:18)

One of [1]the interesting things the new K runtime offers the programmer, is the rather interesting notion of an [2]Assembly Neutral interface, which allows separately compiled components to target the same (from the CLR perspective) type, without that type having been provided to them at compile time. David Fowler talks a little about the implementation [3]here, and after reading it I could understand how things looked from the assembly point of view, but I didn't see any explanation of how the assembly loading happens. Fortunately all of [4]the code is available on GitHub so it is possible to do a little bit of investigation.

The idea of these neutral types is that they are pulled into separate assemblies (named in a way that matches across separate compilations) and these separate assemblies are actually saved as a resource in the assembly that includes them. The AssemblyNeutral definitions themselves are also removed from the original assembly, by removing them from the compilation unit that produces it.

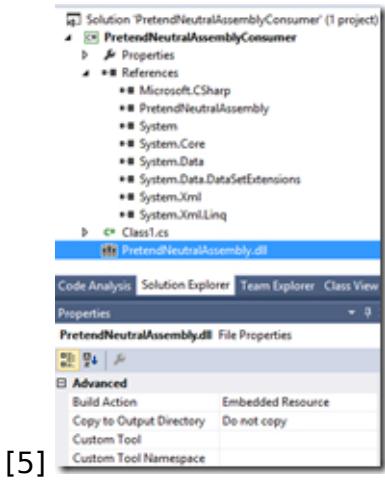
We can simulate the look of the eventual assembly by generating an assembly, PretendNeutralAssembly from this code:

```
namespace PretendNeutralAssembly
{
    public class PretendNeutralInterface
    {
        public void TestMethod()
        {
            Console.WriteLine("Called");
        }
    }
}
```

And then simulate the Roslyn produced main assembly by compiling the following code. Remember that we are pretending that the above code and the code that follows start in the same compilation unit, and the above code is initially marked with the AssemblyNeutral attribute. Roslyn would be used to extract the above code into a separate assembly, which would be referenced by that shown below.

```
namespace PretendNeutralAssemblyConsumer
{
    public class PretendNeutralConsumer
    {
        public void Worker()
        {
            var instance = new PretendNeutralAssembly.PretendNeutralInterface();
            instance.TestMethod();
        }
    }
}
```

When we compile the above, we both reference the PretendNeutralAssembly and also include it as a resource in the final output.



The idea is that multiple assemblies that contain a particular AssemblyNeutral type will all end up with a resource containing the compiled definition of that type. The trick to consuming these interfaces is that we will be loading them inside a host, and this host will be able to hook the assembly loading mechanics of the CLR in order to extract the embedded types. If multiple assemblies define the same neutral type, then they will share the first embedded assembly that is loaded, keeping the CLR happy with its stricter notion of type equality.

The host will override the `Resolve` method on the `AppDomain`, giving it a chance to help the CLR find missing assemblies. In particular, the assemblies that are held in the resources which the CLR's loader knows nothing about. We could wait until the assembly is needed and then go back and search for embedded assemblies, or we can pre-emptively scan for embedded assemblies whenever we load an assembly. The implementation does the latter, adding them to `Dictionary` so that they can be found later.

We can get our demonstration assembly working by hosting it in an assembly that has the following code.

```
static void Main(string[] args)
{
    AppDomain.CurrentDomain.AssemblyLoad += CurrentDomain_AssemblyLoad;
    AppDomain.CurrentDomain.AssemblyResolve += CurrentDomain_AssemblyResolve;
    var loadedAssembly = Assembly.LoadFrom("PretendNeutralAssemblyConsumer.dll");
    // We need to exercise the loaded type, to get the JIT to load the referenced types.
    // We'll just call the Worker method on a created instance.

    var loadedType = loadedAssembly.DefinedTypes.First();
    var instance = Activator.CreateInstance(loadedType);
    var method = instance.GetType().GetMethods().First();
    method.Invoke(instance, null);
}

private static Dictionary<string, Assembly> s_cache = new Dictionary<string, Assembly>();
// When an assembly has been loaded deal with any embedded resource assemblies
private static void CurrentDomain_AssemblyLoad(object sender, AssemblyLoadEventArgs args)
{
```

```

var assembly = args.LoadedAssembly;
// We just look for the specific assembly - in the real code we'd need to look at
// all of the names to find the assemblies.
var stream = assembly.GetManifestResourceStream("PretendNeutralAssemblyConsumer.PretendN
eutralAssembly.dll");
if (stream != null)
{
    var assemblyAsBytes = new byte[stream.Length];
    stream.Read(assemblyAsBytes, 0, (int)stream.Length);
    var loadedAssembly = Assembly.Load(assemblyAsBytes);
    s_cache.Add(loadedAssembly.FullName, loadedAssembly);
}
}

// If the CLR can't find the assembly, see if it a neutral type assembly that it is looking for
private static Assembly CurrentDomain _AssemblyResolve(object sender, ResolveEventArgs
args)
{
    var target = args.Name;
    return s_cache[target];
}

```

AssemblyNeutral types seem like a useful idea. In the real implementation, they will probably only allow interfaces to be marked as AssemblyNeutral, and we are going to be in the usual world of keeping an interface type unchanged once it is published to the world –otherwise there will be a range of interesting loader exceptions if the definition for a type which the CLR loads is different from the neutral definition that a subsequently loaded assembly was compiled against. In some ways it is shame that this requires a special host, but the current implementation at least gives us a way to experiment with the idea without changing the CLR in any way.

You can see the real KRuntime implementation of this in the Github sources on [6]line 121.

1. <http://www.davidfowl.com/asp-net-vnext/>
  2. <http://www.davidfowl.com/assembly-neutral-interfaces/>
  3. <http://www.davidfowl.com/assembly-neutral-interfaces-implementation/>
  4. <https://github.com/aspnet/KRuntime/>
  5. <https://clivetong.files.wordpress.com/2014/05/embedd.png>
  6. <https://github.com/aspnet/KRuntime/blob/64f34f35575ff75c9190c387a9cd52fd29832794/src/klr.hosting.shared/Ru
ntimeBootstrapper.cs>
- 

## 8.7 July

### 8.7.1 Deep, but friendly (2014-07-08 05:49)

Linux system programming: Talking directly to the kernel and the C library  
by Robert Love

A very readable book on the APIs and C libraries that sit just above the Linux kernel. The book has chapters on topics such as files, process management, memory management, signals and time. The author gives a brief overview of what the API means to the kernel and then covers the API as the user sees it, mainly at a raw C callable API level, but occasionally discussing the wrapping that the C library does to ease the burden on the user, for example, when efficiently using buffered I/O (when the most efficient strategy is to load blocks that are a multiple of the device's native block size which might not match the size of blocks that the user code wishes to write).

I enjoyed this book because it is very well written, flowing nicely between the chapters, and because the author offers insights into the kernel implementation as we go along, often comparing the feature as it is implemented in different LINUX variants, and often discussing the history behind the feature. Some parts of the UNIX philosophy are just beautiful – unifying communication with devices behind the file model seems really tidy (though admittedly there's always the escape to using ioctl calls for the extra parts to the communication).

The book has a useful appendices, one on the extensions to the C language implemented in the GCC compiler, and another containing a list of recommended reads. This list contains a number of interesting books on device drivers that I hope to read soon.

Well worth a read if you're going to dive into the kernel and want to see how many of its facilities are exposed to the user space.

1. [http://www.amazon.co.uk/Linux-System-Programming-Talking-Directly-ebook/dp/B00CS94J8U/ref=sr\\_1\\_1?ie=UTF8&qid=1404576985&sr=8-1&keywords=linux+system+programming](http://www.amazon.co.uk/Linux-System-Programming-Talking-Directly-ebook/dp/B00CS94J8U/ref=sr_1_1?ie=UTF8&qid=1404576985&sr=8-1&keywords=linux+system+programming)
- 

### **8.7.2 Evolution is the key (2014-07-10 05:57)**

[1]

Designing Evolvable Web APIs with ASP.NET

by Glenn Block, Pablo Cibraro, Pedro Felix, Howard Dierking, Darrel Miller

I think that [2]

Web API

was an interesting addition to the Microsoft Web stack. We started with Web forms as a way of allowing people to (try to) write their web applications in the style of a desktop windows form application. Some of the weaknesses of this were patched when ASP.NET MVC came along, but over time the focus moved to writing the client side of web applications entirely in JavaScript and there was a need for a better framework for writing non-GUI server code that was there simply to serve data for the clients to consume. The book demonstrates the use of Web API in this wider setting, but also covers a lot more, pushing the reader to implement HTTP as it was originally conceived, with its notions of discoverability (via links), content negotiation

(using accept to get different renderings of the same data depending on the clients needs), cache friendliness (by making sure that GET really is idempotent, and by using Etags to make it possible to verify if data has changed) and using the proper HTTP verbs for updating data.

You can tell the book was written by a group of authors, as the flow isn't the same as a single author book, but given the range of diverse topics that are covered, the need for multiple authors isn't at all surprising.

The first two chapters give a very good overview of HTTP, covering topics ranging from the HTTP methods (verbs) to a great summary of all the various headers, status codes, caching and authentication. The book then covers the difference between using HTTP as a means for REST (representational state transfer) compared with using it for RPC style SOAP messages. The authors give a great case for supporting the former style.

The book then covers ASP.NET Web API, covering the hosting layer, the message handler pipeline and controller handling.

We then go into chapters that discuss things at a higher level – choices for mapping URLs to resources, what it means for an API to be evolvable and the design of media types. These ideas are then used to implement an API for an issue tracking system. We get a whole chapter on improving the API and a chapter on designing the client. The whole time the authors focus on testability, emphasising how the Web API implementation allows us to unit test the implementation by offering simple instantiable implementations that can be used from tests, with loads of C # to demonstrate the implementation.

Later chapters get low level again, covering the hosting model, controllers and routing in detail, formatter and model binding which allow your code to work at the domain level instead of the byte level, and HttpClient, the new way of accessing resources via the web. Chapters on security, OAUTH and testability finish the book.

This book is a great way to learn about modern HTTP APIs such as those being developed by GitHub. The coverage of Web API is really good, describing in some detail the design ideas behind the implementation. I zoned out a little while reading the chapters on media types, but now realise how important they are and so will definitely go back and read them. Well worth a read, as Web API is definitely the future given its support for SPAs.

1. [http://www.amazon.co.uk/Designing-Evolvable-Web-APIs-ASP-NET/dp/1449337716/ref=sr\\_1\\_fkmr0\\_2?ie=UTF8&qid=1404577831&sr=8-2-fkmr0&keywords=extensible+web+api+glen](http://www.amazon.co.uk/Designing-Evolvable-Web-APIs-ASP-NET/dp/1449337716/ref=sr_1_fkmr0_2?ie=UTF8&qid=1404577831&sr=8-2-fkmr0&keywords=extensible+web+api+glen)
2. <http://www.asp.net/web-api>

---

### 8.7.3 I've told you several times, but all at the same time (2014-07-15 05:06)

Programming on Parallel Machines by Norm Matloff

This is a pdf document on the author's web page which can be downloaded [1] here

. It is a very good read, covering the general issues around why we should use parallel programming and common performance pitfalls, but also goes on to demonstrate a number of different

frameworks for writing parallel programs. The last section covers some common problems like matrix multiplication and looks at algorithms for doing them in parallel.

By way of a discussion followed by the implementation of a number of algorithms, the author demonstrates Open MP (a shared memory framework written as an extension to C), GPU programming via CUDA (a cut down C with no recursion but with multi-SIMD threads called warps scheduled by the hardware), Thrust (a framework that allows a higher level expression of programs which can then map to Open MP or CUDA), message passing systems as implemented in Open MPI, and cloud computing via the map-reduce as implemented in Hadoop. I have read whole books on Open MP before, but here the author picks just the key concepts which are demonstrated with their benefits and problems discussed.

This was a great read to get a good feel of the available technologies.

1. <http://heather.cs.ucdavis.edu/~matloff/158/PLN/ParProcBook.pdf>

---

#### 8.7.4 Odds but not ends (2014-07-17 06:04)

The CLR has been around for a while now, and there seem to be fewer interesting developments happening at the virtual machine level. However, there's still a lot of stuff happening at the library and user framework levels, and I was pleased to recently come across a couple of interesting talks.

[1]The Roslyn project, which rewrote the C # and VB compilers in managed code lead to the team getting a load of experience of writing a large managed application, and [2]this talk covers some of the lessons they learned. The worst user experiences are those when there is loads of jitter in the responsiveness of the GUI, and the talk discusses places where there is unexpected allocation - the allocation is indeed cheap, but each garbage collection can pause the user experience. I remember from 20 years ago, the team implementing Dylan had an incremental garbage collector, and frequently pointed out that although overall performance was less, the lack of long pauses when a major collection happened made it a good platform for development environments.

Environments like Windows RT have also been pushing the need for responsive GUIs, and this led to the inclusion of Async in C # 5. There are a lot of subtle things happening under the covers with Async though, and [3]this talk covers what is going on with respect to synchronisation and execution contexts, and how libraries should be designed to avoid unexpected thread transitions when offering async API methods. It was very interesting, and contains many performance demonstrations of real code, giving you a feel for the overheads involved in using Async (and in summary the overhead is generally so low that the extra code clarity more or less demands you use async).

[4]This post on [5].NET native is interesting as it describes a way to see the kind of code that is generated when you use P/Invoke to call into native libraries. When using .NET native you can get to see managed wrappers which will give you a feel for what overheads are involved (as well as allowing you to see if you actually managed to get the P/Invoke signature

right). I would very much like to look at the code that is generated for COM Interop, so intend to try these techniques out in the near future.

I've had problems in the past working with X509 certificates (and the local certificate store). [6]This post covers some of the pitfalls and explains many of the "problems" I've had in the past.

1. <http://msdn.microsoft.com/en-gb/vstudio/roslyn.aspx>
  2. <http://channel9.msdn.com/Events/TechEd/NorthAmerica/2013/DEV-B333#fbid=?hashlink=fbid>
  3. <http://channel9.msdn.com/Events/TechEd/Europe/2013/DEV-B318#fbid=>
  4. <http://blogs.msdn.com/b/dotnet/archive/2014/06/13/net-native-deep-dive-debugging-into-interop-code.aspx>
  5. [http://msdn.microsoft.com/en-us/library/dn584397\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/dn584397(v=vs.110).aspx)
  6. <http://paulstovell.com/blog/x509certificate2>
- 

### 8.7.5 It's not the parsing, but a question of semantics (2014-07-22 06:10)

One very neat feature of Lisp based languages is that you don't really need a parser. Lisp systems provide a facility called the Reader which is responsible for taking a string and converting it into a "parse tree". The neat thing is that the form of this parsed representation is trivial to express in the datatypes that you use for doing your normal programming in the language - this language idea is called [1]homoiconicity. Hence the Reader itself can be very simple, and it certainly makes it easy to do some forms of metaprogramming such as macros for syntax tree transformations. However, when it comes down to actually compiling some source code, systems like Clojure still need a tree representation that is annotated with semantic information about the various forms.

I've recently been having a play with a couple of GitHub Clojure projects that take care of this expansion, and these have been used in a number of recent language analysis tools. Such tools typically need some form of tree that better expresses the dataflow of the resulting code.

The projects are a [2]base analyser and separate extensions for [3]JVM and [4]JavaScript (ClojureScript) backends.

A call to analyse macroexpands the supplied form, and returns a tree representation of the semantic tree.

```
(def form (ana.jvm/analyze '(fn [x] (+ x 1))))
```

We can convert the tree back into source code using the emit-form function

```
user=> (clojure.tools.analyzer.passes.jvm.emit-form/emit-form form)
(fn* ([x] (clojure.lang.Numbers/add x 1)))
```

The top level form that we processed above is tagged as a function definition.

```
user=> (-> form :tag)
```

```
clojure.lang.AFunction
```

The function has a number of overloaded methods associated with it. We can take the first (of one) and look to see what its body looks like.

```
user=> (clojure.tools.analyzer.passes.jvm.emit-form/emit-form (-> form :methods
first :body))
```

```
(clojure.lang.Numbers/add x 1)
```

The body is a do (as each method body is contained inside an implicit do)

```
user=> (-> form :methods first :body :op)
:do
user=> (-> form :methods first :body keys)
(:o-tag :tag :body? :op :env :form :statements :ret :children)
```

[5]This post lists the various tree types that turn up in the annotated tree and the unit tests show the kind of items you expect in maps that represent the nodes, and [6]this video by Timothy Baldridge covers some of the potential uses. It is very useful to have a library that will take care of macroexpanding a given form, and then express its meaning as a tree of a limited node types. To actually compile things we simply need to be able to code generate these node types, knowing that more complicated expressions define their semantics in terms of these simple nodes. Sometimes, the beauty of the ideas behind Lisp astound me.

1. <http://en.wikipedia.org/wiki/Homoiconicity>
2. <https://github.com/clojure/tools.analyzer/>
3. <https://github.com/clojure/tools.analyzer.jvm/>
4. <https://github.com/clojure/tools.analyzer.js>
5. <http://mkremins.github.io/clojure-ast-ref/>
6. [https://www.youtube.com/watch?v=KhRQmT22SSg&list=PLZdCLR02grLp\\_\\_wRg50TavVj4wefg69hM&index=10](https://www.youtube.com/watch?v=KhRQmT22SSg&list=PLZdCLR02grLp__wRg50TavVj4wefg69hM&index=10)

## **8.7.6 To the bottom of the sea (2014-07-24 06:19)**

[1]Pearls of Functional Algorithm Design by Richard Bird

This book has 30 chapters covering various computer science problems such as Sudoku, Rush Hour and some string problems like longest suffix prefix pair, and for each problem the author derives an efficient solution. The book emphasises functional programming, using Haskell as the implementation language.

Each problem is specified via a straightforward Haskell program, which is then transformed into a more efficient solution, typically using tricks like fusion (of fold and map), generalisation or localised use of arrays. At each stage, the author uses calculation to ensure that the derived program matches the specification of the initial program.

The book covers ways of calculating the complexity of the algorithms that are derived, and gives insights into how the derivations are produced. For some of the algorithms, the author gives speed measurements of various test cases to show practically how well the functional algorithm runs. As well as straightforward derivations, there are a couple of chapters that discuss algorithms via the transcript of students at a tutorial, showing how different approaches lead to better solutions ,and how flashes of inspiration about the problem domain can be converted into better solutions.

To be honest, the book is a bit dry, making it best to read it in a number of sittings. However, it is good to see functional techniques and how they can efficiently solve many complex problems. I also used reading it as an opportunity to remind myself about the Haskell language.

1. [http://www.amazon.co.uk/Pearls-Functional-Algorithm-Design-Richard/dp/0521513383/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1405955745&sr=1-1&keywords=pearls+of+functional](http://www.amazon.co.uk/Pearls-Functional-Algorithm-Design-Richard/dp/0521513383/ref=sr_1_1?s=books&ie=UTF8&qid=1405955745&sr=1-1&keywords=pearls+of+functional)
- 

## **8.8 August**

### **8.8.1 Device drivers step by step (2014-08-05 05:32)**

[1]Linux Device Drivers by Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman

In my continuing attempts to get to grips with operating system kernels, I've been playing around with Linux. This all started out when I took a [2]Coursera course on malware, which had a section on Android malware, which included lectures on the virtual machine and covered how it sits on top of a Linux operating system. Interprocess communication happens at a high level in an Android application via Intents, but this maps down to calls via the [3]Binder at the kernel level, where the Binder is driven using a range of IOCTL calls on the respective device.

Wanting to learn a bit more, I found the [4]free section of this course which takes you  
782

through the steps of building a Linux kernel and booting into it. With my copy of VmWare it was easy to download the latest copy of [5]Ubuntu, apt-get the relevant tools (like git), clone the kernel sources and then build a new version of the kernel - though the compilation of the kernel inside a virtual machine on my laptop took several hours. It was however easy to change the banner message for the new kernel and boot it, and then check that we were running my new kernel.

The free part of the tutorial gets to the point of writing a trivial kernel module, and so I started looking for more resources around the subject of device drivers. I was lucky enough to come across this book.

Let me start by saying that the content is outstanding!

It takes you through device drivers - character, block and network. It discusses the parts of the kernel that a device driver author needs to understand, such as interrupt handling, memory mapping and timers. It discusses how modern hardware interacts with devices - I had no idea how PCI and USB drivers worked before reading this, and discusses why the traditional Unix model of character and block devices doesn't abstract network devices correctly. The explanations are clear, and the text always seems to answer the questions that you have while reading.

That is all really interesting material, but the book is also practical. From chapter two, we start writing device drivers, initially for devices which simply remember and return the data that was written to them, but later things become more complicated and we even implement a fake network adapter. The examples have been kept up to date by the authors (as the Linux kernel has changed) and are available on [6]GitHub here.

I would have been happy just reading about all of this, but it has been interesting actually building and running their example code. If you want to understand operating systems at a low level, then this book is a must read!

1. <http://lwn.net/Kernel/LDD3/>
2. <https://class.coursera.org/malsoftware-002>
3. <https://www.nds.rub.de/media/attachments/files/2011/10/main.pdf>
4. <http://www.crashcourse.ca/introduction-linux-kernel-programming-2nd-edition/lesson-1-setting-your-development-host-and-general>
5. <http://www.ubuntu.com/download/desktop>
6. <https://github.com/duxing2007/ldd3-examples-3.x>

---

Will Smith (2014-08-05 14:15:12)

I have always been interested in some of the small RTOS's. QNX is one I find interesting and is used by CISCO in its monster routers and in automobiles. Blackberry move to it as well. [http://www.qnx.com/developers/docs/6.3.2/neutrino/sys\\_arch/kernel.html](http://www.qnx.com/developers/docs/6.3.2/neutrino/sys_arch/kernel.html) In fact my crazy idea is that Microsoft should buy the QNX folks and build a new OS on top of that.

## 8.8.2 See more sharp things (2014-08-19 06:30)

I found a few interesting C # related things over the weekend.

On the open [1]Roslyn codeplex repository there is now a link to a specification for pattern matching in C #. This boils down to a new operator, is, which can be called to determine if a pattern (named via the class on which it is defined) matches, and this operator can also provide a set of out variables that reflect any destructuring in the binding. The standard C # switch statement has been extended to allow cases to be defined that use patterns. The specification can be found [2]here and there is already a mass of discussion [3]here. Unfortunately there isn't yet an implementation available (though one is promised in a few weeks), and it is always a lot easier to digest these things when you can play with examples.

Delegate and Func always appear similar but disconnected in the .NET framework. Your own delegate type is code generated to IL as a subclass of the inbuilt System.MulticastDelegate type, and the compiler goes out of its way to make the construction of such objects transparent - at its root, a MultiCast delegate is just an instance object (of type object, which can be null if you are referencing a static method) together with a token representing the method you wish to call.

However, you cannot simply convert from one to the other, so the xx3 line in the following code will not compile.

```
delegate string TestCode(int x);

static void Main()
{
    TestCode xx = x => x.ToString();
    Func<int, string> xx2 = x => x.ToString();
    Func<int, string> xx3 = xx;
}
```

The trick as outlined in this [4]post, is to use the Invoke method on the delegate instance.

```
Func<int, string> xx4 = xx.Invoke;
```

As you can see from the IL, this generates a new delegate instance that references the function token of the Invoke method.

```
L_0039: ldloc.0
L_003a: ldftn instance string ConsoleApplication6.Program/TestCode::Invoke(int32)
L_0040: newobj instance void [mscorlib]System.Func`2<int32, string>::ctor(object, native int)
L_0045: stloc.2
```

I'm a big fan of [5]Scala, mainly because the language contains a mass of rather cool language features such as call by name and multiple inheritance via Traits. Recently I did the online Coursera Scala course where there was a module on the for comprehension syntax and [6]how it gets translated into a sequence of maps and filters. It was therefore interesting to see a blog post by Eric Lippert that discusses the more complicated translation that [7]C #

specifies for its translation of LINQ expressions. It is rather interesting that overload resolution in C# has a number of issues when it comes to [8]resolving nested lambda expressions which he references in the article.

As usual, when rereading bits of the Joe Duffy's excellent [9]Concurrent Programming in Windows, I noticed some features of the [10]TPL that I hadn't thought about before. [And also noticed some features mentioned that didn't make it through to the release version of the library such as Task.Current where there is now only an internal InternalCurrent, and mention of a Blocking status in Task.Status]

Tasks that are created inside other tasks are by default independent, but there are configuration options to allow you to attach the children to the parent, so that the parent doesn't finish until all of the children have completed.

```
var task = Task.Factory.StartNew(delegate {
    var nestedTask = new Task(delegate
    {
        Thread.Sleep(20000);
        Console.WriteLine("Finishing 2");
    }, TaskCreationOptions.AttachedToParent);
    nestedTask.Start();
    Thread.Sleep(10000);
    Console.WriteLine("Finishing 1");
});

Task.WaitAll(task);
Console.WriteLine("Finished main");
```

There is also the .NET version of a promise, a TaskCompletionSource. An instance of this type has a Task property that returns a Task, and the state of this task can be set using methods on the TaskCompletionSource instance. For example, you can set a result, an exception and also set the task as cancelled using the Try... methods.

```
var tcs = new TaskCompletionSource<int>();

Task.Factory.StartNew(delegate
{
    Thread.Sleep(10000);
    tcs.TrySetResult(20);
});

Task.WaitAll(tcs.Task);
Console.WriteLine("Finished main");
```

Duffy's book brings together and explains a mass of interesting concurrency observations and is well worth many reads.

1. [roslyn.codeplex.com](http://roslyn.codeplex.com)

2. <https://onedrive.live.com/view.aspx?resid=4558A04E77D0CF5!5396&app=Word>

3. <https://roslyn.codeplex.com/discussions/560339>  
4. <http://blog.marcgravell.com/2009/11/solving-delegate-variance.html>  
5. <http://www.scala-lang.org/what-is-scala.html>  
6. <http://www.artima.com/pinsied/for-expressions-revisited.html>  
7. <http://ericlippert.com/2014/08/01/transparent-identifiers-part-two/>  
8. <http://blogs.msdn.com/b/ericlippert/archive/2007/03/26/lambda-expressions-vs-anonymous-methods-part-four.aspx>  
9. [http://www.amazon.co.uk/Concurrent-Programming-Windows-Architecture-Development/dp/032143482X/ref=sr\\_1\\_1?ie=UTF8&qid=1408198041&sr=8-1&keywords=windows+concurrent](http://www.amazon.co.uk/Concurrent-Programming-Windows-Architecture-Development/dp/032143482X/ref=sr_1_1?ie=UTF8&qid=1408198041&sr=8-1&keywords=windows+concurrent)  
10. [http://msdn.microsoft.com/en-us/library/dd460717\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/dd460717(v=vs.110).aspx)

---

### 8.8.3 Stop interrupting me (2014-08-21 06:31)

We've been using the [1]Quartz work scheduler in a service that I'm maintaining at work. This is a .NET port of a Java application, and while looking over the source code I noticed a pattern that I hadn't seen used before - the catching of [2]ThreadInterruptedException and the subsequent use of the Interrupt method on a thread, as [3]for example in this file.

Threads have a load of interesting state information associated with them. Previously I'd researched [4]ThreadAbortException, which is associated with the [5]Thread.Abort method. This causes an asynchronous exception to be thrown on the target thread, which is potentially queued if the thread isn't in a suitable state to throw the exception, and which can use the notion of a pending abort to make sure that the abort is re-thrown when the code exits any catch or finally blocks. This flag can be reset using the [6]Thread.ResetAbort method. Asynchronous exceptions seem to have fallen out of favour in most .NET code, as it is very hard to ensure invariants over your objects if an exception can be thrown at any point in the code - it is impossible to guarantee atomic regions if the presence of asynchronous exceptions unless you start using features like [7]constrained execution regions.

I wasn't aware before that a Thread can have an associated interrupt pending flag, which the CLR checks before it goes into blocking waits. So this code throws the exception on the line with the Sleep.

```
Thread.CurrentThread.Interrupt();
Thread.Sleep(10000);
```

So then, how does this all work under the covers? Time to get out the [8]SSCLI source code and have a quick look.

The cpp code in the file sscli20/clr/src/vm/threads.cpp handles the native control of Threads. Each thread object has a field that records whether a user interrupt is pending called m\_UserInterrupt. The method HandleThreadInterrupt checks this flag and if we are in a suitable state

then it throws the ThreadInterruptedException. This method is called in three other methods - UserSleep, DoAppropriateWaitWorker and DoAppropriateWaitWorkerAlertableHelper (where this last method is called from the non helper method). The first method is called if the user had asked the thread to sleep, and it is DoAppropriateWaitWorker which does the clever logic when the current thread needs to wait for a HANDLE at the OS level. Waiting is quite a complicated business in Windows - if we want an alertable wait, then queued [9]APC messages can hijack our thread, releasing the wait and this may require us to wait again for a smaller time interval if we are trying to wait with a timeout. These APC messages could come from IO completions for example.

Also in this file is the interesting ReadyForAsyncException which contains the code to see if an asynchronous abort can be delivered to the thread. This involves walking the stack to see if we are inside catch or finally methods. There is also the code in UserAbort which does the logic of a thread abort and which is very complicated - we need to do different things depending on whether we are in managed code or are in unmanaged code for the target thread. We may also need to wait until the thread hits a safe point which will cause it to call CommonTripThread.

The real CLR has the ability to hijack code, by changing the return address on stack frame to regain control. Some of the code to handle this can be seen in excep.cpp around the method IsThreadHijackedForThreadStop.

1. <https://github.com/quartznet/quartznet/tree/master/src>
  2. [http://msdn.microsoft.com/en-us/library/system.threading.threadinterruptedException\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.threading.threadinterruptedException(v=vs.110).aspx)
  3. <https://github.com/quartznet/quartznet/blob/master/src/Quartz/Core/QuartzSchedulerThread.cs>
  4. [http://msdn.microsoft.com/en-us/library/system.threading.threadabortexception\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.threading.threadabortexception(v=vs.110).aspx)
  5. [http://msdn.microsoft.com/en-us/library/system.threading.thread.abort\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.threading.thread.abort(v=vs.110).aspx)
  6. [http://msdn.microsoft.com/en-us/library/system.threading.thread.resetabort\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.threading.thread.resetabort(v=vs.110).aspx)
  7. [http://msdn.microsoft.com/en-us/library/ms228973\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/ms228973(v=vs.110).aspx)
  8. [http://en.wikipedia.org/wiki/Shared\\_Source\\_Common\\_Language\\_Infrastructure](http://en.wikipedia.org/wiki/Shared_Source_Common_Language_Infrastructure)
  9. [http://msdn.microsoft.com/en-gb/library/windows/desktop/ms681951\(v=vs.85\).aspx](http://msdn.microsoft.com/en-gb/library/windows/desktop/ms681951(v=vs.85).aspx)
- 

#### **8.8.4 A marvellous miscellany (2014-08-26 07:30)**

Three different books and articles that I read this weekend across a range of different subjects.

[1]The Little Book of Semaphores by Allen B. Downey

This book looks at various concurrency problems and shows how they can be solved using semaphores. At first I found it slightly strange that the book insisted on using global variables in the solutions (rather than thread locals) which led to a number of strange implementations, but the problems are really interesting and the book explains its solutions well. Semaphore patterns like turnstiles and lightswitch are described and used in the solutions. The book covers fairness and starvation, and mentions a solution by J.P.Morris who showed that it was possible to solve the reader/writers problem without starvation in the presence of weak semaphores

using the trick of batching the waiting threads and ensuring that the batches complete in order (which proved incorrect a conjecture by Dijkstra). A quick read, but worthwhile.

## [2]Building Cloud Apps with Microsoft Azure

This is a recently released free eBook which covers the various parts of Azure as well as a number of patterns and concepts for Cloud development. Each short chapter has a long list of additional resources which you can easily click and visit. The book gives a concise introduction to large parts of Azure and is well worth a read if you are thinking about moving to the cloud - a world where transient failure and scalability make application development a little different from in-house desktop applications.

## [3]OAuth tutorial

I've been doing a [4]Coursera course on [5]developing cloud services for Android handheld systems and the application we had to develop used [6]OAuth for the authentication and authorisation. Using OAuth means that the user credentials do not need to be stored on the phone, while at the same time each application can identify itself to the OAuth provider which can prompt for the relevant credentials. In the actual application we wrote for the course, the Spring framework provides the actual OAuth implementation, but as it is always good to understand what is happening under the covers, someone posted a link to this tutorial on the user forum.

It is brief and informative, giving a good overview of the various parts of the protocol.

1. <http://greenteapress.com/seaphores/>
  2. [http://blogs.msdn.com/b/microsoft\\_press/archive/2014/07/23/free-ebook-building-cloud-apps-with-microsoft-azure.aspx](http://blogs.msdn.com/b/microsoft_press/archive/2014/07/23/free-ebook-building-cloud-apps-with-microsoft-azure.aspx)
  3. <http://tutorials.jenkov.com/oauth2/index.html>
  4. <http://www.coursera.org/>
  5. <https://class.coursera.org/mobilecloud-001>
  6. <http://en.wikipedia.org/wiki/OAuth>
- 

## 8.9 September

### 8.9.1 Sometimes your worker role needs to run server GC (2014-09-02 07:18)

We were debugging a dump of a Windows Azure worker process at work when it suddenly struck me that the process was using workstation GC and not server GC - I noticed this when looking at the unmanaged stack and noticing that the GC was running C++ functions in the WKS class and not the SRV class.

Right, I said, I know how to deal with this and went to the worker role's configuration and changed the setting of the gcServer configuration entry. When I deployed this, the role went into a continuous reboot cycle.

Luckily I came across [1]this post, which discusses how you can write a quick post-deployment

script which tweaks the configuration file of the process that hosts your Azure worker.

I must admit that I'd never thought about the hosting of the worker role before, so I logged into the instance and had a quick look. The worker role is deployed to the approot directory on the F drive of the virtual machine.

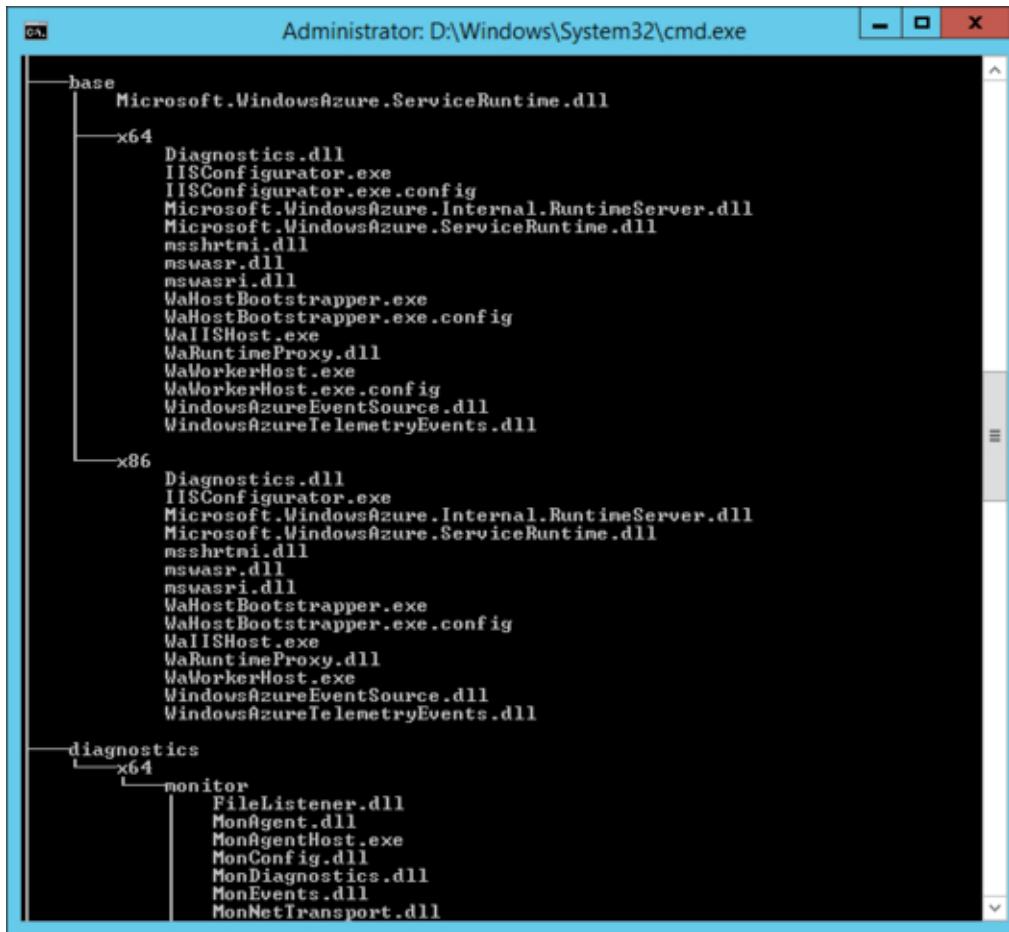
[2]

```
F:\>tree /F
Folder PATH listing
Volume serial number is 0000000D CA71:2DD4
F:.
    1d2093a9d5aa487598771dc65fd152fe.0.27.cssx.tag
    787d934a-1738-41e1-86d6-52bfd89b61cb.csman
    Cloud.uar.csman
    RoleModel.xml
    RuntimeSetup.Manifest
    [Content_Types].xml
    __entrypoint.txt

    approot
        AWSSDK.dll
        AWSSDK.pdb
        CS.dll
        Common.Logging.dll
        Common.Logging.pdb
        diagnostics.wadcfg
        EntityFramework.dll
        GCSettingsManagement.ps1
        Local.Quartz.dll
        Local.Quartz.pdb
        Microsoft.Data.Edm.dll
        Microsoft.Data.OData.dll
        Microsoft.Data.Services.Client.dll
        Microsoft.WindowsAzure.Configuration.dll
        Microsoft.WindowsAzure.Diagnostics.dll
        Microsoft.WindowsAzure.Diagnostics.StorageUtility.dll
        Microsoft.WindowsAzure.ServiceRuntime.dll
        Microsoft.WindowsAzure.Storage.dll
        Moq.dll
        Newtonsoft.Json.dll
        Ninject.dll
        RedGate.Cloud.BackupEngine.dll
        RedGate.Cloud.BackupEngine.pdb
        RedGate.Cloud.Scheduler.dll
        RedGate.Cloud.Scheduler.dll.config
        RedGate.Cloud.Scheduler.pdb
        RedGate.Shared.Utils.dll
        RedGate.Shared.Utils.pdb
        ServerGC.cmd
        System.Data.SQLite.dll
        System.Spatial.dll
```

And the F: also has a mass of other code installed on to it.

[3]



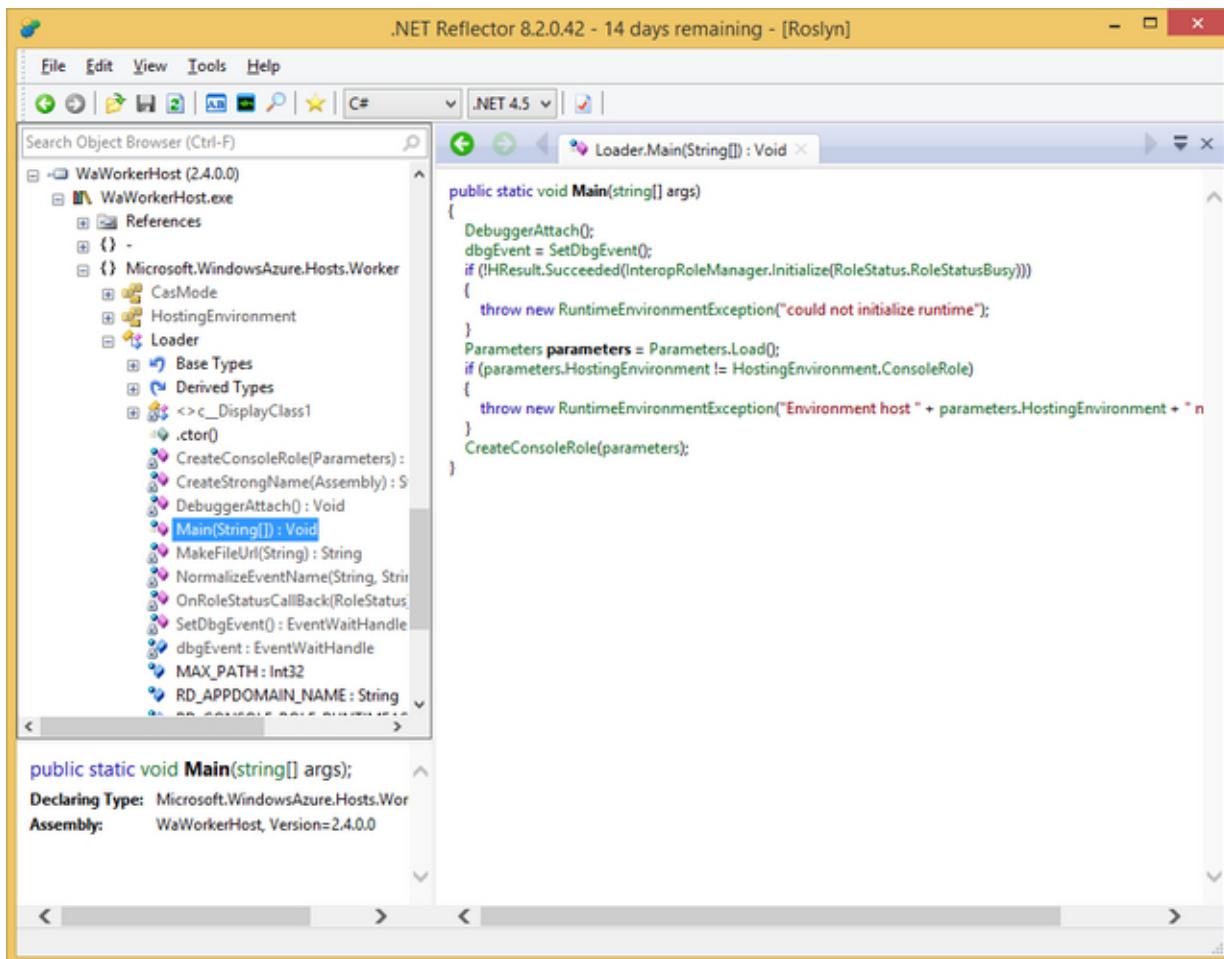
This includes the WaWorkerHost process that acts as a host process for your worker's code (as you can see from processor explorer).

[4]

RemoteForwarderService.exe	0.01	30,612 K	40,936 K	1272	Windows Azure Remote Des...	Microsoft Cor	
WindowsAzureGuestAgent.exe	0.10	36,740 K	53,892 K	1384	Windows.AzureGuestAgent	Microsoft Cor	
__ WaHostBootstrapper.exe		18,092 K	20,920 K	4124	Microsoft Windows Azure Ru...	Microsoft Cor	
__ DiagnosticsAgent.exe		28,740 K	36,420 K	4712	Windows Azure Diagnostics ...	Microsoft Cor	
__ conhost.exe		624 K	2,684 K	2132	Console Window Host	Microsoft Cor	
__ MonAgentHost.exe	< 0.01	15,640 K	25,068 K	3928	Monitoring Agent Host	Microsoft Cor	
__ conhost.exe		648 K	2,688 K	3760	Console Window Host	Microsoft Cor	
__ RemoteAccessAgent.exe		23,932 K	27,804 K	2920	Windows Azure Remote Des...	Microsoft Cor	
__ conhost.exe		568 K	2,612 K	1696	Console Window Host	Microsoft Cor	
__ Microsoft.VisualStudio.WindowsAzure.Re...		33,212 K	33,328 K	4296	Microsoft.VisualStudio.Wind...	Microsoft Cor	
__ conhost.exe		560 K	2,584 K	3124	Console Window Host	Microsoft Cor	
__ WaWorkerHost.exe	0.02	310,328 K	123,812 K	4804	Microsoft Windows Azure W...	Microsoft Cor	
WindowsAzureTelemetryService.exe		42,704 K	53,136 K	1424	TelemetryService	Microsoft Cor	
WaAppAgent.exe		29,576 K	39,328 K	1520	Microsoft Windows Azure M...	Microsoft Cor	

This hosting process is actually managed.

[5]



If you want to change properties of the hosting process then you need to change the app.config for this worker process as the PowerShell script from the blog post does.

1. <http://blogs.msdn.com/b/cie/archive/2013/11/14/enable-server-gc-mode-for-your-worker-role.aspx>
2. <https://clivetong.files.wordpress.com/2014/08/dir1.png>
3. <https://clivetong.files.wordpress.com/2014/08/dir2.png>
4. <https://clivetong.files.wordpress.com/2014/08/processes.png>
5. <https://clivetong.files.wordpress.com/2014/08/managed.png>

## 8.9.2 What does better mean for overloads? (2014-09-03 06:21)

This example came up at work.

```

class A : I1 { }
interface I1 { }
interface I2 : I1 { }

```

```
static void Method(I1 x) { }
static void Method(I2 x) { }

static void Main(string[] args)
{
    Method(new A());
}
```

It seems reasonable that the invocation calls the overload defined on I1, but where in the actual specification is this defined - what does it mean for one overload to be better than another?

The overloads are [1]certainly applicable and so we need to consider [2]which methods are better. To see which methods are better in a particular parameter position, we need to check whether a conversion is better and this behaviour is [3]defined here.

1. [http://msdn.microsoft.com/en-us/library/aa691337\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/aa691337(v=vs.71).aspx)
  2. [http://msdn.microsoft.com/en-us/library/aa691338\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/aa691338(v=vs.71).aspx)
  3. [http://msdn.microsoft.com/en-us/library/aa691339\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/aa691339(v=vs.71).aspx)
- 

### 8.9.3 A good read on concurrency in general (and Java in particular) (2014-09-04 06:33)

[1]Java Concurrency in Practice by Brian Goetz et al

I'd been hearing people mention this book for a long time, and took quite a while to actually get hold of a copy to read. It's a very good book. Its strength is that it covers the Java concurrency related libraries in good detail, but more than this, there is a lot of supporting material on concurrency in general.

The second chapter of the book covers what it means for something to be thread safe. This term is thrown around a lot by developers, but the chapter makes it clear that there are lots of properties that you may require of a thread safe class, and it is surprisingly difficult to pin down exactly what thread safety is. The next two chapters of the book talk about the sharing and composing of objects, and how this affects their concurrent use. The last chapter of the introductory section covers the basic building blocks, including synchronised and concurrent collections, blocking queues, blocking and interruptible methods, and synchronisers. All of the topics are covered in depth, and I think most people would learn something from reading them.

The second section of the book is about structuring concurrent applications. It covers the notion of tasks, and the connected notions of cancellation and shutdown. The Java mechanism for running thread pools is covered, and this is followed by a discussion of the interplay between GUI applications and concurrency. This has a very well written section on why GUIs are single threaded, answering well the question of why do I always need to move onto the GUI thread before changing GUI elements.

Section three is on liveness, performance and testing. This covers livelocks, deadlocks, performance, scalability, Amdahl's law and also has a good discussion of testing concurrent programs.

Section four, advanced topics, covers locks and moves on to a discussion of when you might want to use atomic variables and non-blocking synchronisation. There is also a section on the Java memory model and the subtle guarantees of the platform such as how it needs to ensure that publication happens correctly after a constructor runs. There is also a chapter on [2]AbstractQueuedSynchronizer a class which acts a superclass for many of the Java library's synchronization constructs.

This is a thoroughly interesting book, both explaining the Java library very well and providing a lot of general advice on the topic of concurrency, and is well worth a read.

1. [http://www.amazon.co.uk/Java-Concurrency-Practice-Brian-Goetz/dp/0321349601/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1409470581&sr=1-1&keywords=java+concurrency+in+practi](http://www.amazon.co.uk/Java-Concurrency-Practice-Brian-Goetz/dp/0321349601/ref=sr_1_1?s=books&ie=UTF8&qid=1409470581&sr=1-1&keywords=java+concurrency+in+practi)
  2. <http://www.docjar.com/html/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.java.html%20>
- 

#### **8.9.4 Making your C# more effective (2014-09-09 06:43)**

[1]Effective C #: 50 Specific Ways To Improve Your C # by Bill Wagner

This is one of those books that lists 50 different issues and pieces of advice for using the programming language - there are variants for C # and Java and many other languages. The items are divided into 6 different chapters.

Chapter one, "Language Idioms" discusses some language level issues - avoiding user defined conversion operators, conditional attributes instead of #if and using "is" and "as" instead of casting. All of the advice seems very reasonable. The next chapter is on "Resource Management", and goes into detail about class initialization and then covers the standard Dispose pattern. Immutability and when to use a value type instead of a reference type are also covered.

Chapter three, "Expressing Designs in C #" is very good and full of lots of good advice. Limiting visibility and not returning references to internal objects are covered, as are using interfaces instead of inheritance and the difference between interface methods and virtual methods. There are items on defining callbacks using delegates and using the event pattern for notifications. In this mixed bag, there is a discussion about making chunky rather than chatty calls, and also a discussion of co- and contra-variance.

Chapter four is entitled "Working With The Framework". This covers ordering relationships with IComparer<T> and IComparable<T> and then moves onto writing parallel algorithms using PLINQ.

Chapter five covers the dynamic type, in a chapter with the rather misleading title of "Dynamic Programming in C #". There's a lot of discussion about how dynamic works, and a good explanation of expression trees.

The last chapter, "Miscellaneous", throws in some advice about boxing and structuring

applications as sets of small assemblies.

There were some items that I found very useful. How to use `IFormattable` to define better string representations for types, minimizing duplicated constructor logic and some of the PLINQ notes in particular. Not a bad book, but some of the advice is either well known or potentially just a matter of opinion.

The author appears on a recent [2].NET Rocks where [3]he discusses C # 6.0.

1. [http://www.amazon.co.uk/Effective-covers-4-0-Specific-Development/dp/0321658701/ref=sr\\_1\\_1?ie=UTF8&qid=1409496829&sr=8-1&keywords=effective+c%23](http://www.amazon.co.uk/Effective-covers-4-0-Specific-Development/dp/0321658701/ref=sr_1_1?ie=UTF8&qid=1409496829&sr=8-1&keywords=effective+c%23)
  2. <http://www.dotnetrocks.com/>
  3. <http://www.dotnetrocks.com/default.aspx?showNum=1029>
- 

### 8.9.5 csc generates code as well you know (2014-09-19 06:53)

I wondered the other day how the C # compiler, `csc.exe`, decides which version of the runtime to target. Some C # language features are effectively syntactic sugar that is logically translated into other C # which is compiled. But when you target a certain version of the framework, how does the compiler change this code generation? The answer appears to be that it looks at the references that you compile against.

Take this small C # class definition in a file `enumerable.cs`

```
using System.Collections.Generic;
```

```
class Test
{
    IEnumerable<int> DoGeneration()
    {
        yield return 10;
    }
}
```

If you compile from the command line using

```
csc /t:library enumerable.cs
```

you'll see that the generated class's constructor uses `CurrentManagedThreadId`

The screenshot shows a debugger window displaying assembly code for a constructor. The title bar reads '<DoGeneration>d\_0::ctor : void(int32)'. The assembly code is as follows:

```
.method public hidebysig specialname rtspecialname
    instance void .ctor(int32 '<>1__state') cil managed
{
    .custom instance void [mscorlib]System.Diagnostics.DebuggerHiddenAttribute::.ctor() = { 01 00 00 00 }
    // Code size      25 (0x19)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: call    instance void [mscorlib]System.Object::.ctor()
    IL_0006: ldarg.0
    IL_0007: ldarg.1
    IL_0008: stfld   int32 Test/'<DoGeneration>d_0'.'<>1__state'
    IL_000d: ldarg.0
    IL_000e: call    int32 [mscorlib]System.Environment::get_CurrentManagedThreadId()
    IL_0013: stfld   int32 Test/'<DoGeneration>d_0'.'<>1_initialThreadId'
    IL_0018: ret
} // end of method '<DoGeneration>d_0'::ctor
```

If you compile the same file referencing the .NET 4 version of mscorelib.dll (via a reference Assembly on my machine which has 4.5 installed)

```
csc /t:library /nostdlib /r:"c:/Program Files (x86)/Reference Assemblies/Microsoft/Framework/.NETFramework/v4.0/mscorlib.dll" enumerable.cs
```

you'll see that it references ManagedThreadId

[2] [3]

The screenshot shows a debugger window displaying assembly code for a constructor, identical to the one above but with different assembly instructions due to the different framework version. The title bar reads '<DoGeneration>d\_0::ctor : void(int32)'. The assembly code is as follows:

```
.method public hidebysig specialname rtspecialname
    instance void .ctor(int32 '<>1__state') cil managed
{
    .custom instance void [mscorlib]System.Diagnostics.DebuggerHiddenAttribute::.ctor() = { 01 00 00 00 }
    // Code size      30 (0x1e)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: call    instance void [mscorlib]System.Object::.ctor()
    IL_0006: ldarg.0
    IL_0007: ldarg.1
    IL_0008: stfld   int32 Test/'<DoGeneration>d_0'.'<>1__state'
    IL_000d: ldarg.0
    IL_000e: call    class [mscorlib]System.Threading.Thread [mscorlib]System.Threading.Thread::get_CurrentThread()
    IL_0013: callvirt instance int32 [mscorlib]System.Threading.Thread::get_ManagedThreadId()
    IL_0018: stfld   int32 Test/'<DoGeneration>d_0'.'<>1_initialThreadId'
    IL_001d: ret
} // end of method '<DoGeneration>d_0'::ctor
```

I think it's rather cool that the C# translation can use different methods from the framework depending on what it expects to find on the target. It can lead to a few [4]interesting and unexpected problems though.

1. <https://clivetong.files.wordpress.com/2014/09/pic13.png>
2. <https://clivetong.files.wordpress.com/2014/09/pic1.png>
3. <https://clivetong.files.wordpress.com/2014/09/pic21.png>
4. <http://blog.marcgravell.com/2012/09/iterator-blocks-missing-methods-and-net.html>

---

### **8.9.6 The best explanation of cache coherency I've seen (2014-09-30 06:48)**

[1]A Primer on Memory Consistency and Memory Coherence by Daniel Sorin, Mark Hill and David Wood

This is, by far, the best explanation on cache coherency and memory consistency that I have read. There are lots of books that give the subject some coverage - from programming manuals on various languages to works on parallel programming, but none of them match the standard of this text. It starts out by covering serial consistency and then gives a great explanation why you may want to weaken this model in order to gain performance. It then has a chapter on the predominant TSO/x86 model (which Intel's documentation hints that they support) and then follows this with a great chapter on even weaker memory models. The text is introductory, but covers things in a way that made what I had read in the past consistent. The following chapters cover cache coherency protocols, from the shared bus MOESI protocol to the more scalable directory based protocols. I liked the way that the authors explain some of the gaps in the usual protocols, when the usual state machines fail to explain that often we need to get a response to a message before the transition can be fully made for the state of a particular cache line.

This introduction is fairly short and therefore quick to read, but explains a subject that is often covered in far less detail. Simply brilliant, answering all sorts of unanswered questions from past reads.

1. A\_Primer\_on\_Memory\_Consistency\_and\_Coherence

---

## **8.10 October**

### **8.10.1 Some interesting bits and pieces (2014-10-02 06:51)**

[1]This series which implements a simplified browser is really interesting. I've tried to find my way around the HTML and CSS specifications in the past, and they are a fairly dry read, whilst seeing code and hence being able to get a grip on the algorithms is a great way of improving one's understanding.

[2]Lenses seems to be a common pattern to be making its way through the functional programming world, and this [3]F # version looks interesting.

The [4]type theory podcast is shaping up nicely. The initial podcast talks about the connection between testing and type theory, and the second episode covers [5]Idris whose focus on dependent types seems to be quite the rage at the moment. The first episode mentions [6]Px, a system for deriving programs from proofs. I still have the book on the bookshelf and

it was the reason that I wanted to study for a Phd.

[7]Mirage, a functional operating system is also looking interesting, as does [8]Docker, an application container framework that runs on Linux. Linux seems to have a rich [9]history of such containers. The background on how this is implemented is covered [10]here.

1. <http://limpet.net/mbrubeck/2014/09/17/toy-layout-engine-6-block.html>
  2. <https://www.fpcomplete.com/school/to-infinity-and-beyond/pick-of-the-week/basic-lensing>
  3. <https://github.com/yncro/aether/blob/master/src/Aether/Aether.fs>
  4. <http://typetheorypodcast.com/2014/08/episode-1-peter-dybjer-on-type-theory-and-testing/>
  5. [http://en.wikipedia.org/wiki/Idris\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Idris_(programming_language))
  6. <https://archive.org/details/somephilosophic00untegoog>
  7. <http://openmirage.org/wiki/install>
  8. <https://github.com/docker/>
  9. [http://media.wix.com/ugd/295986\\_d73d8d6087ed430c34c21f90b0b607fd.pdf](http://media.wix.com/ugd/295986_d73d8d6087ed430c34c21f90b0b607fd.pdf)
  10. [http://media.wix.com/ugd/295986\\_d73d8d6087ed430c34c21f90b0b607fd.pdf](http://media.wix.com/ugd/295986_d73d8d6087ed430c34c21f90b0b607fd.pdf)
- 

## 8.10.2 TPL Dataflow still has its uses (2014-10-14 06:27)

[1]TPL Dataflow by Example: Dataflow and Reactive programming in .NET by Matt Carkci

We've been doing some coding at work on a distributed system which does some processing that seems to match the streaming approach of the dataflow library. There are loads of fairly interesting posts on the TDF (TPL Dataflow library), both on [2]MSDN and on the [3]PFX team blog, but sometimes a book is good as it gives a more rounded picture of the technology, and perhaps tells you about problems that users commonly have.

This book is very short, coming in at 52 pages, though there are numerous pages with full code listings (of which the actual code you are interested in forms a tiny percentage), so it is a fairly quick read. There's a quick introduction to the dataflow library, and then the vast majority of the book is filled with the author going through the block types that come with the library - the execution blocks like ActionBlock, the buffering blocks like BufferBlock and the grouping blocks like BatchedJoinBlock. Each block type is accompanied with a simple code sample that takes many pages. This kind of material is available elsewhere.

The unique parts of the book are really the last 8 or so pages which list some ideas and gotchas when designing a dataflow program. These were an interesting read, and many of the items I hadn't come across in a single source.

To be honest though, if you want to understand the implementation of Dataflow, this [4]Channel 9 video interview with Stephen Toub gives a lot of very useful implementation detail about the interfaces that the TDF library defines. If you are considering writing your own blocks, then [5]This document discusses many of the implementation issues that you will face if you want to write your own dataflow blocks - along the way, it gives more detail about

the expected protocols behind the interfaces.

1. [http://www.amazon.co.uk/TPL-Dataflow-Example-Reactive-Programming/dp/1499149352/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1413055605&sr=1-1&keywords=tpl+dataflow](http://www.amazon.co.uk/TPL-Dataflow-Example-Reactive-Programming/dp/1499149352/ref=sr_1_1?s=books&ie=UTF8&qid=1413055605&sr=1-1&keywords=tpl+dataflow)
  2. [http://msdn.microsoft.com/en-us/library/hh228603\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/hh228603(v=vs.110).aspx)
  3. <http://blogs.msdn.com/b/pfxteam/>
  4. <http://channel9.msdn.com/Shows/Going+Deep/Stephen-Toub-Inside-TPL-Dataflow>
  5. <http://download.microsoft.com/download/1/6/1/1615555D-287C-4159-8491-8E5644C43CBA/Guide%20to%20Implementing%20Custom%20TPL%20Dataflow%20Blocks.pdf>
- 

### **8.10.3 If you use R# to code generate, make sure to regenerate if things change (2014-10-21 06:42)**

At work the other day when I was looking through a pull request. I noticed what looked like a very strange equality method on a C # struct, shown here as if it were named Foo.

```
public override bool Equals(object obj)
{
    if (ReferenceEquals(null, obj)) return false;
    if (ReferenceEquals(this, obj)) return true;
    if (obj.GetType() != this.GetType()) return false;
    return Equals((Foo) obj);
}
```

The code looks very wasteful - you cannot inherit from structs, so rather than the GetType malarkey you'd surely expect just an 'is'. In the end it turned out that the Equals method had been generated by R #, but at the time it was generated, the containing type had been a class and this was later converted to a struct.

For a struct, R # would have generated the much more reasonable looking

```
public override bool Equals(object obj)
{
    if (ReferenceEquals(null, obj)) return false;
    return obj is Foo && Equals((Foo) obj);
}
```

While I was checking my understanding of what was going on, I spent a little time looking at the x86 code that was generated by the various methods.

A cut down version of the code using is,

```
public bool Test2(object obj)
{
    if (!(obj is A)) return false;
```

```
return true;  
}
```

generates the code

```
0:004> u 002f00f8 002f0119  
002f00f8 55 push ebp  
002f00f9 8bec mov ebp,esp  
002f00fb 85d2 test edx,edx <<<< check null  
002f00fd 740c je 002f010b  
002f00ff 813a2c381200 cmp dword ptr [edx],12382Ch <<<< check type handle  
002f0105 7502 jne 002f0109  
002f0107 eb02 jmp 002f010b  
002f0109 33d2 xor edx,edx <<<< generate less than optimal code  
002f010b 85d2 test edx,edx  
002f010d 7504 jne 002f0113  
002f010f 33c0 xor eax,eax  
002f0111 5d pop ebp  
002f0112 c3 ret  
002f0113 b801000000 mov eax,1  
002f0118 5d pop ebp  
002f0119 c3 ret
```

The C # 'is' can simply generate check that the type handle (vtable) for the object is the desired type.

The longer form code, simplified to

```
public bool Test1(object obj)  
{  
if (ReferenceEquals(null, obj)) return false;  
if (obj.GetType() != GetType()) return false;  
return true;  
}
```

is a little more complicated.

```
0:004> u 002f0098 002f00e7  
002f0098 55 push ebp  
002f0099 8bec mov ebp,esp  
002f009b 57 push edi  
002f009c 56 push esi  
002f009d 50 push eax  
002f009e 8bf9 mov edi,ecx  
002f00a0 8bf2 mov esi,edx  
002f00a2 85f6 test esi,esi <<<<<< null check  
002f00a4 7507 jne 002f00ad  
002f00a6 33c0 xor eax,eax  
002f00a8 59 pop ecx  
002f00a9 5e pop esi
```

```

002f00aa 5f pop edi
002f00ab 5d pop ebp
002f00ac c3 ret
002f00ad b92c381200 mov ecx,12382Ch <<< type handle for the structure type
002f00b2 e84920e2ff call 00112100 <<<< ***** See below for this!!!!!!!
002f00b7 8945f4 mov dword ptr [ebp-0Ch],eax
002f00ba 8bce mov ecx,esi
002f00bc e86bad2472 call msclib _ni+0x27ae2c (7253ae2c) <<<< GetType
002f00c1 8bf0 mov esi,eax
002f00c3 0fbe07 movsx eax,byte ptr [edi]
002f00c6 8b55f4 mov edx,dword ptr [ebp-0Ch]
002f00c9 884204 mov byte ptr [edx+4],al
002f00cc 8bca mov ecx,edx
002f00ce e859ad2472 call msclib _ni+0x27ae2c (7253ae2c) <<<<< GetType
002f00d3 3bf0 cmp esi,eax
002f00d5 7407 je 002f00de
002f00d7 33c0 xor eax,eax
002f00d9 59 pop ecx
002f00da 5e pop esi
002f00db 5f pop edi
002f00dc 5d pop ebp
002f00dd c3 ret
002f00de b801000000 mov eax,1
002f00e3 59 pop ecx
002f00e4 5e pop esi
002f00e5 5f pop edi
002f00e6 5d pop ebp
002f00e7 c3 ret

```

In the \*\*\* line we allocate a boxed instance of Foo. The first line gets the type handle of the struct type, and this is passed as the first argument to the called method.

```

mov ecx,12382Ch
call 00112100

```

Addresses differ as this is a different run, but the called method is doing allocation of a new boxed instance.

```

0:000> u 00162100 0016211b
00162100 8b4104 mov eax,dword ptr [ecx+4]
00162103 648b15380e0000 mov edx,dword ptr fs:[0E38h] <<< Use fast allocation buffer
0016210a 034240 add eax,dword ptr [edx+40h]
0016210d 3b4244 cmp eax,dword ptr [edx+44h]
00162110 7709 ja 0016211b
00162112 894240 mov dword ptr [edx+40h],eax
00162115 2b4104 sub eax,dword ptr [ecx+4]
00162118 8908 mov dword ptr [eax],ecx <<<< Set the object header
0016211a c3 ret
0016211b e915788b73 jmp clr!JIT_New (73a19935) <<<< fast path not available so punt

```

I thought it was quite interesting to see the boxing required to call GetType on 'this'

800

when 'this' is a struct instance, and it is also interesting to see the bump allocation that happens on the fast path allocation.

The key point however, is that code generation is all very well, but it is useful to record the assumptions behind the generation so that you can regenerate if things change.

---

## 8.11 December

### 8.11.1 Homomorphic encryption could make it easier to use cloud computing power (in the future) (2014-12-08 09:13)

I recently had the opportunity to do a lightning talk at work on [1]Homomorphic Encryption. The slides are available [2]here, though, as it was a five minute lightning talk, the details are a little sparse. [3]This paper showing an encryption scheme over the integers is a good detailed explanation of taking an encryption scheme and converting it into something that is fully homomorphic, and also demonstrates the costs that make such schemes currently impractical.

[4]Craig Gentry's thesis is also a very good read on what is a very clever technology.

1. [http://en.wikipedia.org/wiki/Homomorphic\\_encryption](http://en.wikipedia.org/wiki/Homomorphic_encryption)
  2. <https://onedrive.live.com/embed?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%214996&authkey=ANVjPQr5ZT1UMb8&em=2>
  3. <http://eprint.iacr.org/2009/616>
  4. <http://crypto.stanford.edu/craig/>
- 

### 8.11.2 Distributed computing is another world! (2014-12-11 07:42)

I'm been writing Azure tooling for a while, but have only just finished writing my first Azure based [1]Microservice. Along the way though, I've been caught a few times by the difference between distributed applications and the more usual desktop applications that we have all been writing for many years. Things like disciplined exception handling become more important, as taking an exception without doing anything about it can leave you in a state where some part of a chain of applications just stops responding, and we enter a world where methods can simply timeout leaving you with the issue of whether to wait and for how long.

There are various good sources of information on the Internet. [2]This brief introduction to the fallacies is very good, but for a bigger picture overview [3]this pdf book is a very good

read. It offers a good introduction as to why we are moving to a more distributed world, and then discusses many of the issues you are going to hit when writing such applications. It also has a very good series of links to additional material such as a [4]book on the new datacentre architectures, [5]the problems you encounter when implementing distributed algorithms for real, [6]other consistency models, and a [7]discussion of the CAP theorem. The book also mentions [8]commutative datatypes which look like a very interesting idea that I would like to pursue.

Talking of datacenters, this [9]recent post gives an insight into the Amazon datacentres.

I think I did the book a disservice by reading it quickly in one sitting. I need to go back and ponder the many points that it raises.

1. <http://en.wikipedia.org/wiki/Microservices>
  2. <http://www.somethingsimilar.com/2013/01/14/notes-on-distributed-systems-for-young-bloods/>
  3. <http://book.mixu.net/distsys/index.html>
  4. <http://www.morganclaypool.com/doi/pdf/10.2200/s00193ed1v01y200905cac006>
  5. [http://static.googleusercontent.com/media/research.google.com/en//archive/paxos\\_made\\_live.pdf](http://static.googleusercontent.com/media/research.google.com/en//archive/paxos_made_live.pdf)
  6. <https://cs.brown.edu/courses/cs227/archives/2012/papers/weaker/cidr07p15.pdf>
  7. <http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed>
  8. <https://hal.inria.fr/file/index/docid/555588/filename/techreport.pdf>
  9. <http://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws/>
- 

### **8.11.3 The source (or a variant) is often available (2014-12-12 08:20)**

I was helping someone out with a win32 GUI issue the other day. Whenever I do this, I always end up wanting slightly more details of how the various messages that you see in Spy++ are being sent and used by the system... for example, your WndProc gets a chance to handle many messages, and then is supposed to call the default window procedure - how much work does this default window procedure do and what would happen if I didn't call it?

I realised that a good way to get a feel about what was happening would be to go and look at Wine - an implementation of windows that runs on top of other operating systems. This is implemented to make it possible to run Windows code on other systems and so respects the messaging that a normal Windows system is going to do.

My original question was about the WM\_PARENTNOTIFY message, and sure enough I could find the implementation in the send\_parent\_notify function in [1]message.c file. One can also get a feel for how the builtin controls are implemented, say by looking at the implementation of buttons in [2]button.c. Moreover, the implementation of the default window procedure as DEFWND\_DefWinProc in [3]defwnd.c cleared up some questions around the context menu.

This open source is great for reading in order to understand things. Less so for actually  
802

building and running - I spent five hours at the weekend trying to build Chrome only to hit C++ compilations errors (concerning a method that was getting an extra parameter), and then today tried to [4]build F # from source only to end up with a stream of compilation problems.

1. <http://source.winehq.org/git/wine.git/blob/HEAD:/dlls/user32/message.c>
  2. <http://source.winehq.org/git/wine.git/blob/HEAD:/dlls/user32/button.c>
  3. <http://source.winehq.org/git/wine.git/blob/HEAD:/dlls/user32/defwnd.c>
  4. <https://visualfsharp.codeplex.com/>
- 

#### **8.11.4 It's the season to watch loads of videos (2014-12-22 06:46)**

Loads of conferences seem to have recently published videos to various streams - [1]Strange Loop, [2]Clojure Conj, [3]React 2014, [4]ICFP, [5]ML WorkShop so here is a list of a few I found particularly interesting.

[6]Inside Transducers and a talk from the previous year, [7]Transducers, both by Rich Hickey discuss a new construct that has been added to the latest version of [8]Clojure. [9]Transducers offer a means for doing algorithmic transformations as part of a pipeline and integrate nicely with [10]Clojure's async coding infrastructure. Yet again Clojure seems to have taken ideas from other languages and made them into something slightly different but very practical.

I've been a fan of Reactive Functional Programming ever since I came across the [11]FRAN library in Haskell many years ago, but have never had the chance to use such ideas in a real commercial application. I found [12]this talk by Paul Betts very interesting where he discusses the use of such ideas in implementing the GitHub client. This discussion of the [13]different formulations of FRP is also very good, There is also [14]a talk here on the [15]React framework which uses ideas from functional programming to control mutation of the DOM for browser applications.

Memory management has always been an interest of mine and [16]this talk on [17]Shenan-dooah is a good introduction to some of the issues of multicore garbage collection and its trade offs. There was also a talk on why [18]deterministic memory management could be useful on the JVM.

I very much enjoyed these talks on JavaScript. The [19]first discusses the benefits of implementing JavaScript in JavaScript (as you get the many optimisations applied on the core library which you also want to apply to user code), and the second discusses the[20] implementation of the Chakra, IE's JavaScript engine.

[21]This discussion of the merits and problems of type systems was also very pragmatic and led into some of the interesting talks at ICFP [22]on dependent types and the use of O'CAML for cloud programming on top of the [23]Mirage operating system. There were also plenty of interesting talks at the associated Haskell workshop such as a talk on [24]Core, GHC's intermediate language. [25]Lenses are an idea that also seem to be making their way

into loads of functional languages.

1. <https://thestrangeloop.com/>
  2. <http://clojure-conj.org/>
  3. <https://www.youtube.com/user/reactconf>
  4. <http://icfpconference.org/icfp2014/>
  5. <http://www.mlworkshop.org/>
  6. <https://www.youtube.com/watch?v=4KqUvG8HPYo&list=WL&index=13>
  7. <https://www.youtube.com/watch?v=6mTbuzafcII>
  8. <http://clojure.org/>
  9. <http://clojure.org/transducers>
  10. <https://clojure.github.io/core.async/>
  11. <http://conal.net/fran/>
  12. <https://www.youtube.com/watch?v=1XNATGjqM6U&list=WL&index=19>
  13. <https://www.youtube.com/watch?v=Agu6jipKfYw&list=WL&index=16>
  14. <https://www.youtube.com/watch?v=IVvHPPc12TM&list=WL&index=12>
  15. <http://facebook.github.io/react/>
  16. <https://www.youtube.com/watch?v=QcwyKLlmXeY>
  17. <https://christineflood.wordpress.com/>
  18. <https://www.youtube.com/watch?v=cp-Tr0ixCRM&list=WL&index=9>
  19. <https://www.youtube.com/watch?v=YFFvNQWf3Iw&list=WL&index=21>
  20. <https://www.youtube.com/watch?v=0qnMmoF2GqM&list=WL&index=22>
  21. <https://www.youtube.com/watch?v=SWTwkYbcWU0&list=WL&index=17>
  22. <https://www.youtube.com/watch?v=rhWMhTjQzsU>
  23. <https://www.youtube.com/watch?v=oKJ8Sxqr4r8>
  24. <https://www.youtube.com/watch?v=kBm1sW3hNck&index=8&list=WL>
  25. <https://www.youtube.com/watch?v=efv0SQNde5Q&index=14&list=WL>
-

# 2015

## 9.1 January

### 9.1.1 Some useful distributed algorithms (2015-01-12 06:44)

[1]Distributed Algorithms: An Intuitive Approach by Wan Fokkink

I kept coming across descriptions of various algorithms from distributed computing during my reading, such as [2]this one on Paxos, but felt that I'd like to read a book that gives an overview of the many clever algorithms that exist out there. This book is excellent, covering lots of algorithms, often offering an intuitive explanation for the correctness of the algorithm and with a good set of examples and exercises to get you familiar with the algorithms.

It is divided into two sections - message passing and shared memory. In the message passing model the processes communicate via messages that pass across a network which can have various topologies in various algorithms from ring to directed to undirected and which can be FIFO or random order across the various links. The shared memory section looks at algorithms that depend on variants of the atomic test-and-set operations.

The first section breaks the algorithms into the various groupings. Snapshots - how do we capture the state of the processes and the messages in transit at a given moment, so that we could potentially debug or restart the distributed system. Waves - algorithms for visiting every node in the network, where each node only knows about various neighbours. Deadlock detection - getting a global wait graph from the network which can be used to detect circular wait patterns (and hence deadlock). Termination detection - discovering that a launched activity has finished. Garbage collection in the presence of inter-node references, and an explanation of the close relationship between GC and some of the earlier algorithms. Routing - how the network can adapt to knowledge about best paths.

The first section then continues with the more typical distributed system algorithms I have seen mentioned before. Election of a leader by a group of nodes, anonymous networks and how this affects the election process. Synchronous networks, where each process needs to take a step before all processes take another step. Handling crash and byzantine failures - in the former the process just stops communicating, in the latter it can continue behaving but in an illegal manner. The last chapter of the first part covers mutual exclusion, various types of locking in a distributed system and ways to ensure fairness.

At times one can get a little lost in the details of the algorithms which are just listed

one after another in the chapters, but I found the exercises a good way to step back and think about the differences between the algorithms.

Section two has chapters of algorithms for processes running in shared memory. Chapter one covers the usual Peterson's and Bakery algorithms, but then goes into a bit more details about memory coherence and avoids many flying cache lines by using test-and-test-and-set. Chapter two covers barriers and the next chapter covers self-stabilization. The last chapter covers online scheduling, considering the various scheduling policies that a task may have as a hard deadline.

The book is a great introduction to vast range of clever algorithms for doing operations in a distributed setting. It is well written and you are guaranteed to learn something from it.

1. <http://www.amazon.co.uk/gp/product/0262026775>
  2. <http://harry.me/blog/2014/12/27/neat-algorithms-paxos/?hn=1>
- 

### **9.1.2 Writing high-performance .NET code can be hard (2015-01-14 06:25)**

[1]Writing High-Performance .NET Code by Ben Watson

A really good book from someone who has clearly worked hard in the past to maximize the performance of .NET applications. The book has a strong emphasis on using [2]Event-tracing for windows, ETW, for getting information about the behaviour of various components of the CLR (such as [3]jitting or [4]garbage collection behavior), and also makes a lot of good observations on measurement, opining that averages often not the best measurement and one should instead aim at a particular percentile (which is something that Gil Tene discusses in [5]his talk on understanding latency)

Chapter one covers performance measurement and tools. Here the author describes ETW (by way of example), perfmon and performance counters, Windbg and its associated SOS extension for CLR debugging, decompilation using ILSpy/Reflector, the sysinternals tools such as VMMap, and also mentions profilers such as the one that comes with Visual Studio. The author seems to shy away from using profilers because of their invasiveness, and he is often more concerned with measuring the performance on systems that don't need the applications to be restarted in order to get measurements. Certainly it would be hard work to profile a production system using rewriting .NET profilers, and ETW seems to be positioned as a low overhead logging framework while still offering great detail on the application being logged.

Chapter two covers garbage collection and it's a very good overview. It has clear explanations on the various modes for the garbage collection, server/workstation with and without background garbage collection (for generation two) and explains how the mode maps down to the number of threads that the system is going to be using. When to force full garbage collections (hardly ever), when to compact the large object heap, when to handle the garbage collection pending event and how to track down fragmentation (using Windbg) are covered, as well as how to monitor the performance of the memory system.

Chapter three covers the JIT. When to NGEN and when to use the 4.5 background compilation mode are covered. The author also discusses the C # language constructs that generate masses of code - LINQ and dynamic code in particular, and again covers the way to monitor what the JIT is doing.

Chapter four covers asynchronous programming and is a collection of items that the author has considered when writing his own systems. There is an emphasis on using [6]Tasks as the abstraction for chunks of work, given that these support cancellation and various combinatorics for combining them in interesting ways. PLINQ, timers, and await are also covered in some of the observations.

Chapter five on general coding and design looks at various things. Class versus struct and the massive savings in memory usage that structs can offer, sealing and virtual dispatch contrasted with interface dispatch at a polymorphic call site, avoiding boxing and casting, and using exceptions as a general control flow operation. There's also an interesting note on dynamic code generation.

Chapter six looks at parts of the .NET Framework, and points out that you should understand the cost of every API call you make - particularly as the framework often provides many different ways to do the same thing (for example parsing XML). This chapter looks at some miscellaneous items that are commonly misused.

The next two chapters look at performance counters in more detail, and then look at writing your own trace events into the ETW logs.

The final two chapters discuss how you can make your team more performance focused and how to avoid the various performance traps.

I liked this book because of its practical application. The explanations of the various technologies are really good, and the text is peppered with stories of where the observations actually helped in practice. ETW is being pushed hard as the technology to use for logging and it is really good to find a book that focusses on the use of this.

1. [http://www.amazon.co.uk/Writing-High-Performance-NET-Code-Watson/dp/0990583430/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1420710347&sr=1-1&keywords=writing+high+performance](http://www.amazon.co.uk/Writing-High-Performance-NET-Code-Watson/dp/0990583430/ref=sr_1_1?s=books&ie=UTF8&qid=1420710347&sr=1-1&keywords=writing+high+performance)
2. <http://msdn.microsoft.com/en-us/magazine/cc163437.aspx>
3. [http://msdn.microsoft.com/en-us/library/vstudio/ff356158\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/ff356158(v=vs.100).aspx)
4. <http://blogs.msdn.com/b/maoni/archive/2014/12/30/gc-etw-events-4.aspx>
5. <https://www.youtube.com/watch?v=9MKY4KypBzg>
6. [http://msdn.microsoft.com/en-us/library/system.threading.tasks.task\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx)

---

### 9.1.3 Category theory seems to pop up everywhere (2015-01-16 06:21)

[1]Basic Category Theory For Computer Scientists by Benjamin C. Pierce

I have looked at this book before, but decided to try to give it another read over the holiday. The content is rather dry, but for each of the concepts the author provides a mass of examples that make it easier to understand the concepts.

The book is fairly short with only four chapters, and the fourth chapter is just descriptions of additional material with a paragraph describing each items relevance. However, it does what it says on the cover - providing a way to understand terms that appear to crop up all over the place in Computer Science these days. Chapter three on applications covers some domain theory and Cartesian Closed Categories in a few pages.

I'm not sure I'd recommend this as an end to end read, but more as something to dip into when terms need to be understood.

1. [http://www.amazon.co.uk/Category-Computer-Scientists-Foundations-Computing/dp/0262660717/ref=sr\\_1\\_1?ie=UTF8&qid=1420888800&sr=8-1&keywords=basic+category+theory](http://www.amazon.co.uk/Category-Computer-Scientists-Foundations-Computing/dp/0262660717/ref=sr_1_1?ie=UTF8&qid=1420888800&sr=8-1&keywords=basic+category+theory)
- 

#### 9.1.4 Understanding the CLR via its assembly code (2015-01-19 06:39)

[1]Expert .NET 2.0 IL Assembler by Serge Lidin

I have read large parts of this book multiple times, but have never sat down and read it cover to cover before. It's a great book for understanding the CLR. The book isn't just about IL assembly language as a programming language. The book covers this but also covers the translated form of the IL, discussing the format and sections of a PE file including the various metadata tables of the various types (think assembly or module or class) and heaps. It also gives a great explanation of the various IL instructions and discusses the semantics of generics and exceptions. All in all, I think it is a great book for understanding the CLR and I always (re)learn something every time I reread it.

For example, when you want to call the methods on a valuetype via an interface, you typically need to box it. If the struct is not mutable this is usually not the semantics you want. When you instantiate a generic class using a structure type, the user would like the interface method to be invoked without any boxing, and the CLR version 2 got a new IL instruction, [2]constrained to deal with this.

class

Holder

<T>

where

T :

808

```
IFoo

{
    T field;

    public
    Holder(T x)
```

```
public
void
DoInc()
```

```
{
    field.Increment();
}
}
```

leads to the following code for the DoInc method

```
IL_0000: nop
IL_0001: ldarg.0
IL_0002: ldflda !0 class ConsoleApplication10.Holder`1<!T>::'field'
IL_0007: constrained. !T
IL_000d: callvirt instance void ConsoleApplication10.IFoo::Increment()
IL_0012: nop
IL_0013: ret
```

The book also warns you of some of the times when C# isn't the assembly language of .NET... if you satisfy an interface using a non-virtual method, the C# compiler silently makes it sealed virtual, or if you nest a non-generic class inside a generic class the compiler silently makes the nested class generic so that you have access to the type parameter.

The book teaches you loads of miscellaneous facts about the CLR - debug information and various security attributes for example, and how unmanaged mixed-mode assemblies work. The emphasis on using ILDASM/ILASM to round trip code with modification is also very informative.

A thoroughly great book!

1. [http://www.amazon.co.uk/Expert-NET-2-0-IL-Assembler-ebook/dp/B001DUAFKG/ref=sr\\_1\\_1?ie=UTF8&qid=1420298870&sr=8-1&keywords=expert+il+2.0+assembler](http://www.amazon.co.uk/Expert-NET-2-0-IL-Assembler-ebook/dp/B001DUAFKG/ref=sr_1_1?ie=UTF8&qid=1420298870&sr=8-1&keywords=expert+il+2.0+assembler)
2. [http://msdn.microsoft.com/en-us/library/system.reflection.emit.opcodes.constrained\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.reflection.emit.opcodes.constrained(v=vs.110).aspx)

---

### 9.1.5 JavaScript Ninja Secrets (2015-01-21 06:43)

[1]Secrets of the JavaScript Ninja by John Resig and Bear Bibeault

This is a very good book which is so much more than a simple JavaScript language text. Sure, it covers some of the subtle points of the JavaScript language, and you are expected to have knowledge of JavaScript before reading the book, but it also offers a lot of information about some of the practical problems you will meet as a JavaScript programmer.

It starts in a slightly odd manner. There is a chapter discussing the benefits of cross browser development, followed by a chapter in which the author writes a simple testing framework. This framework allows you to embed tests inside an HTML page, and displays the results of the assertions inside the tests in an easy to read manner. Lots of the explanations that follow in the book will be expressed as assertions in the language of this testing framework. This way of working works very well.

The next section of the book, Apprentice Training, consisting of six chapters, has five chapters that focus on the subtle parts of the JavaScript language. The book focuses on the functional nature of JavaScript, discusses why it is important and then gives a very good explanation of scope and the four ways that you can invoke a function object - as a function, as a method, as a constructor and via apply and call. There is a good discussion of recursion followed by an example of memorisation, showing the big benefits of pure functional code. This is followed by a chapter on closures, which describes what they are and then shows how they can be used to implement partial application and temporary scopes with private local variables. We then go object-oriented in a discussion of prototypes and how they can be used to get features like those of standard OO languages. The author defines a JavaScript mini-framework that defines inheritance hierarchies that support a super operation to access base types. There is then a chapter on regular expressions, showing their great power (though you'll need to be responsible to use them).

The last chapter of this section talks about the single threaded nature of browsers, the typical JavaScript host, and discusses how timers can be set and cleared. There is also lots of cross browser detail in how these things are implemented, and the book covers some of the common gotchas.

The next section, Ninja Training, first talks about runtime code evaluation, at first using eval and Function to make new function objects, but then moving on to describe the inbuilt decompilation (as you can ask a Function for its source code). There is then an interesting section showing some of the uses of these techniques.

The next chapter looks at with statements, a feature of JavaScript that you either love or hate. There is a [2]small concise example that shows a powerful micro-template engine that is written in a tiny amount of code, which shows the power of the features we have been shown before.

There are then two chapters on cross-browser strategies for using the DOM. Lots of practical discussion of the differences between browsers which taught me a lot about the DOM and how it should be used.

The last section, Master Training, also looks at browser differences, looking at the differences in the event model and CSS selectors.

The book is a really good read. It explains the parts of JavaScript that it covers really well, and you'll learn loads about browsers and the DOM which is very interesting and useful if you do cross-browser work. The browsers that it discusses are a little behind the times, but that is probably the only complaint I have.

1. [http://www.amazon.co.uk/Secrets-JavaScript-Ninja-John-Resig/dp/193398869X/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1421091417&sr=1-1&keywords=secrets+of+the+javascript+ni](http://www.amazon.co.uk/Secrets-JavaScript-Ninja-John-Resig/dp/193398869X/ref=sr_1_1?s=books&ie=UTF8&qid=1421091417&sr=1-1&keywords=secrets+of+the+javascript+ni)
  2. <http://ejohn.org/blog/javascript-micro-template/>
- 

## 9.2 February

### 9.2.1 You can have your own CLR too (2015-02-16 08:04)

I was surprised when the [1]source to the CoreCLR was published on GitHub a few weeks ago. Over the weekend I thought I'd start to build it and begin looking through the code in detail. I was amazed at how easy it is to build.

First you check out the sources from GitHub. Next you download the latest version of [2]CMake and put it on your path. Then you run the build.cmd script. Your PC gets a bit busy for 30 minutes and then you have your own version of the CLR and the associated mscorel ib.dll. [There is one gotcha concerning the DIA SDK that the script will warn you about - the fix is a copy from an older VS installation to the newest]

You can then use your existing C # compiler to compile against this version of the CLR.

```
c:\Windows\Microsoft.NET\Framework          \v4.0.30319\csc.exe      /nostdlib  
/r:D:\git\coreclr\binaries\Product\x64\debug\mscorl ib.dll test.cs
```

and then run it using CoreRun

```
D:\git\coreclr\binaries\ash\Product\x64\debug\CoreRun.e xe test.exe
```

Brilliantly painless.

1. <https://github.com/dotnet/coreclr>
2. <http://www.cmake.org/download/>

---

## 9.3 April

### 9.3.1 Time Reborn (2015-04-14 06:59)

[1]Time Reborn: From the Crisis in Physics to the Future of the Universe by Lee Smolin

This was a very interesting read in the popular science category. It centres on the position on time in the laws of physics, first describing how time lost its place as a central focus of many theories, and then discusses the problems that this causes when physicists try to apply their theories to the universe as a whole. Smolin discusses how the reductionist approach of deriving laws by observing a system from the outside, is inapplicable to the discovery of the laws that govern the universe as a whole. The book is a mash up of interesting ideas from new theories and discussions around existing ideas and theories, all pitched at the popular science level with understandable explanations.

1. [http://www.amazon.co.uk/Time-Reborn-Crisis-Physics-Universe/dp/014104652X/ref=sr\\_1\\_1?ie=UTF8&qid=1428828491&sr=8-1&keywords=time+reborn+lee+smolin](http://www.amazon.co.uk/Time-Reborn-Crisis-Physics-Universe/dp/014104652X/ref=sr_1_1?ie=UTF8&qid=1428828491&sr=8-1&keywords=time+reborn+lee+smolin)
- 

### 9.3.2 F# Deep Dives (2015-04-14 07:46)

[1]F # Deep Dives by various

I was really looking forward to reading this book, and it was well worth waiting for. The book has many contributing authors, each of whom provides one of the 12 chapters. Of course, with such a diverse set of contributors, some of the chapters are more interesting than others, but many emphasise the benefits of F # as a functional first programming language.

There is an interesting opening chapter on how functional first programming languages can be effectively used in industry, but there were five chapters that I found particularly interesting. Parsing text-based languages looks at the task of writing a parser in F # and uses some interesting techniques such as active patterns to make it easy to read the resulting code. Chapter 6, integrating stock data into the F # language, offers a brief introduction to type providers, and then shows most of the steps to implementing type provider for CSV and YQL.

In the following section on developing complete systems, there are chapters on developing rich user interfaces using the MVC pattern which shows how F # can be used to write an application that uses WPF for its UI, a great chapter on asynchronous and agent-based programming that looks at ETL (extract, transform and load) models, and a chapter on writing

games using XNA which emphasises the async workflow which can help keep the state machines of a game understandable.

You'll learn something from all of the chapters and the book is well worth a read.

1. [http://www.amazon.co.uk/F-Deep-Dives-Tomas-Petricek/dp/1617291323/ref=sr\\_1\\_1?ie=UTF8&qid=1428949089&sr=8-1&keywords=f%23+deep+dives](http://www.amazon.co.uk/F-Deep-Dives-Tomas-Petricek/dp/1617291323/ref=sr_1_1?ie=UTF8&qid=1428949089&sr=8-1&keywords=f%23+deep+dives)
- 

### 9.3.3 So how do you patch a deployed Nuget package (2015-04-16 07:06)

Quite some time ago, [1]this blog post went past on the usual Microsoft RSS feed, discussing how Microsoft are going to make it possible to patch NuGet libraries. Using the standard mechanism of putting an assembly in the GAC and redirecting clients to use it via a publisher policy is a way to deploy a patch, but the article hints at an attribute that signals to Microsoft Update that such an updated assembly needs to be installed. I thought that it was probably time to see what was happening.

Adding a NuGet reference to SignalR and then using Reflector to look at the assembly attributes, there is now a use of  
[assembly: [2]AssemblyMetadata("Serviceable", "True")]  
to flag the assembly to the runtime.

Looking in the [3]CoreClr source code on github, in the [4]assembly.cpp file there's a GenerateBreadCrumbForServicing method which writes details about the assembly under c:\programData\Microsoft\NetFramework\BreadcrumbStore\ and sure enough when I look at that folder in explorer I see a mass of NuGet packages that my applications have loaded. These take the form of empty files with names such as

EntityFramework, Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561 934e089

It will be interesting to see how well this mechanism works in practice - if I xcopy deploy something, there is going to be a delay between the first time I load the assembly (leading to the breadcrumb being set) and Microsoft Update running and fixing it up. I'm also interested in what other AssemblyMetadata attribute values there are being used, but investigating that is going to have to wait.

1. <http://blogs.msdn.com/b/dotnet/archive/2014/01/22/net-4-5-1-supports-microsoft-security-updates-for-net-nuget-libraries.aspx>
2. [http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Reflection.AssemblyMetadataAttribute..ctor\(String, String\)](http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Reflection.AssemblyMetadataAttribute..ctor(String, String))
3. <https://github.com/dotnet/coreclr>
4. <https://github.com/dotnet/coreclr/blob/master/src/vm/assembly.cpp>

---

## 9.4 May

### 9.4.1 Effective Modern C++ (2015-05-06 07:08)

[1]Effective Modern C++ by Scott Meyers

C++ has changed a lot over the years and this book covers loads of the subtle points around the newer additions to the language. As usual with the Scott Meyer books, the author takes a number (42 in this case) of items grouped into 8 categories and discusses the issues around the particular topic.

The topics covered include [2]perfect forwarding and so-called universal references, smart pointers which after C++ 14 cover most of the memory management scenarios, lambda expressions and the related issues around variable capture (and whether to capture the variables by reference or by value), and a very interesting section on concurrency in the new C++.

As in his other books, some of the sections are a little overwhelming in technical detail, but you always come away with a better understanding of a topic than when you started reading the issue. A good read and a good introduction to modern effective C++.

1. [http://www.amazon.co.uk/Effective-Modern-Specific-Ways-Improve/dp/1491903996/ref=sr\\_1\\_1?ie=UTF8&qid=1430845697&sr=8-1&keywords=effective+modern+c%2B%2B](http://www.amazon.co.uk/Effective-Modern-Specific-Ways-Improve/dp/1491903996/ref=sr_1_1?ie=UTF8&qid=1430845697&sr=8-1&keywords=effective+modern+c%2B%2B)
  2. <http://eli.thegreenplace.net/2014/perfect-forwarding-and-universal-references-in-c/>
- 

### 9.4.2 Some low level locking papers (2015-05-19 07:23)

It's been a bit quiet around here, as I have been making my way through 50 or so videos of talks from [1]//build/. I'll do a summary post when I get through them all.

In the meantime, I've also been reading a few low level papers. It's quite interesting that hardware transactional memory is becoming more available, and so [2]this paper on malloc placement is worth a read - the transactional memory uses the cache lines in the L1 and L2 caches to see if a value has been modified by another party while the transaction has been happening. This means that caches can be affected by memory patterns that place many values into locations which map to the same cache line.

On the same blog there were also good posts on the liveness benefits of [3]LIFO lock patterns and implementations of [4]spin-then-park locks.

This [5]gitbook on the Linux kernel is also worth reading if you are interested on how the system gets from a reboot to a running kernel, and [6]this paper describes the many

techniques for x86 virtualization.

1. <http://www.buildwindows.com/>
  2. <https://blogs.oracle.com/dave/resource/arxiv-1504.04640v2.pdf>
  3. [https://blogs.oracle.com/dave/entry/locks\\_with\\_lifo\\_admission\\_order](https://blogs.oracle.com/dave/entry/locks_with_lifo_admission_order)
  4. [https://blogs.oracle.com/dave/entry/waiting\\_policies\\_for\\_locks\\_spin](https://blogs.oracle.com/dave/entry/waiting_policies_for_locks_spin)
  5. <http://0xax.gitbooks.io/linux-insides/content/Booting/linux-bootstrap-1.html>
  6. [http://www.vmware.com/pdf/asplos235\\_adams.pdf](http://www.vmware.com/pdf/asplos235_adams.pdf)
- 

## 9.5 June

### 9.5.1 So what happened at //build (2015-06-03 07:14)

[1]//build happened at the beginning of May, and I've been spending a fair amount of time lately working my way through the session videos that have been appearing on [2]Channel 9. I'll try to summarise the various videos that I have watched.

By far the most important video is the [3]interview with Don Box. After his prominence in the COM and .NET communities he seemed to disappear off the radar some time ago. In this brief interview he tells us what he has been up to in the XBox team.

Roslyn makes it easy for users to write their own compiler extensions, and [4]this talk pushes the idea of getting library writers producing compiler helpers to aid users with integration of their library into a product. There's general discussion about Roslyn [5]here. There's a general [6]state of the union talk on the .NET ecosystem, which shows how much innovation is happening in the .NET world at the moment. This talk discusses [7]building cross-platform .NET.

[8]The chat with Anders is also worth a watch

Visual Studio talks include an [9]introduction to writing extensions, integrated [10]Git in VS2015, some debugger tips and [11]improvements in VS2015, and some methods for [12]debugging performance problems. It's not quite Visual Studio, but there's a deep dive into the [13]cross platform code editor here.

There are some good videos on ASP.NET, [14]introducing ASP.NET 5 and [15]another on improving performance.

Moving on to Azure, the new reliable services are introduced [16]here with a deep dive into reliable actors [17]here. There are some new web site facilities on Azure covered [18]here. There's also a talk on some of the [19]practical issues in developing efficient cloud applications.

I very much enjoyed [20]this talk on the improved compiler technology Microsoft have

been developing. They can use the same compiler backend to compile Objective-C as a means to extend a bridge to iPhone applications to bring them into the Windows platform. They have also got a [21]bridge for Android applications.

JavaScript related talks include two on TypeScript, the [22]first is an introduction followed by a [23]discussion of the future direction. There is also a [24]talk on WinJS changes. The new Microsoft browser is covered [25]here and [26]here, and this talk [27]discusses its new development tools.

With the success of Docker in the Linux world, Microsoft are keen on bringing it to Windows. The advantages of containers are [28]covered in this talk. [29]A walk through of producing a Docker application is interesting. These containers [30]may well run on the new nano-server.

There one of two odds and ends. [31]Continuum, using your phone as a normal computer was really interesting. Using other [32]means of authentication was interesting and [33]support of HTTP/2 were worth watching.

The mass of talks that I watched were on Universal Windows Apps, the progression of Metro style applications from the Windows 8 days. The idea now is that these applications are portable across devices that range from the phone to the large screen computer. The general platform is [34]covered here, the [35]app model is covered here and [36]this covers their deployment. Microsoft is going to provide yet another [37]bridge for covering installer based windows applications into windows universal applications. There is detail about [38]the new lifecycle, [39]the navigation model and [40]app to app communication in this new world.

[41]XAML plays a key role in [42]implementing these applications, so there are [43]many extensions to features like [44]data binding as well as [45]general XAML optimisation techniques. This talk [46]discusses universal app development and this shows [47]how they can be developed in VS2015. [48]Publishing is also an important aspect. [49]The design patterns for such apps are supposed to make it possible to [50]design an app that works on the small and large scales [51]while remaining responsive. There are [52]many talks on these issues including [53]accessibility. [54]Migrating an existing application was also [55]covered.

All in all it was an interesting build. It was great to see more work on .NET and Roslyn, and the universal windows platform may turn out to be really interesting. Only time will tell.

1. <http://www.buildwindows.com/>
2. <https://channel9.msdn.com/Events/Build/2015>
3. <https://channel9.msdn.com/Events/Build/2015/C9-22>
4. <https://channel9.msdn.com/Events/Build/2015/3-725>
5. <https://channel9.msdn.com/Events/Build/2015/C9-06>
6. <https://channel9.msdn.com/Events/Build/2015/2-614>
7. <https://channel9.msdn.com/Events/Build/2015/3-670>
8. <https://channel9.msdn.com/Events/Build/2015/C9-11>
9. <https://channel9.msdn.com/Events/Build/2015/3-99>
10. <https://channel9.msdn.com/Events/Build/2015/3-746>
11. <https://channel9.msdn.com/Events/Build/2015/3-677>
12. <https://channel9.msdn.com/Events/Build/2015/3-731>

13. <https://channel9.msdn.com/Events/Build/2015/3-680>
14. <https://channel9.msdn.com/Events/Build/2015/2-687>
15. <https://channel9.msdn.com/Events/Build/2014/3-605>
16. <https://channel9.msdn.com/Events/Build/2015/2-700>
17. <https://channel9.msdn.com/Events/Build/2015/2-66>
18. <https://channel9.msdn.com/Events/Build/2014/3-625>
19. <https://channel9.msdn.com/Events/Build/2015/2-667>
20. <https://channel9.msdn.com/Events/Build/2015/3-610>
21. <https://channel9.msdn.com/Events/Build/2015/2-702>
22. <https://channel9.msdn.com/Events/Build/2014/3-576>
23. <https://channel9.msdn.com/Events/Build/2015/3-644>
24. <https://channel9.msdn.com/Events/Build/2015/2-720>
25. <https://channel9.msdn.com/Events/Build/2015/3-682>
26. <https://channel9.msdn.com/Events/Build/2015/2-656>
27. <https://channel9.msdn.com/Events/Build/2015/3-638>
28. <https://channel9.msdn.com/Events/Build/2015/2-704>
29. <https://channel9.msdn.com/Events/Build/2015/2-683>
30. <https://channel9.msdn.com/Events/Build/2015/2-755>
31. <https://channel9.msdn.com/Events/Build/2015/2-703>
32. <https://channel9.msdn.com/Events/Build/2015/2-639>
33. <https://channel9.msdn.com/Events/Build/2015/3-88>
34. <https://channel9.msdn.com/Events/Build/2015/C9-04>
35. <https://channel9.msdn.com/Events/Build/2015/2-617>
36. <https://channel9.msdn.com/Events/Build/2015/2-695>
37. <https://channel9.msdn.com/Events/Build/2015/2-692>
38. <https://channel9.msdn.com/Events/Build/2015/3-626>
39. <https://channel9.msdn.com/Events/Build/2015/3-779>
40. <https://channel9.msdn.com/Events/Build/2015/3-765>
41. <https://channel9.msdn.com/Events/Build/2015/C9-05A>
42. <https://channel9.msdn.com/Events/Build/2014/2-507>
43. <https://channel9.msdn.com/Events/Build/2015/2-629>
44. <https://channel9.msdn.com/Events/Build/2015/3-635>
45. <https://channel9.msdn.com/Events/Build/2015/3-698>
46. <https://channel9.msdn.com/Events/Build/2015/2-790>
47. <https://channel9.msdn.com/Events/Build/2015/2-650>
48. <https://channel9.msdn.com/Events/Build/2015/3-710>
49. <https://channel9.msdn.com/Events/Build/2015/2-658>
50. <https://channel9.msdn.com/Events/Build/2015/2-679>
51. <https://channel9.msdn.com/Events/Build/2015/2-672>
52. <https://channel9.msdn.com/Events/Build/2015/2-658>
53. <https://channel9.msdn.com/Events/Build/2015/2-612>
54. <https://channel9.msdn.com/Events/Build/2015/3-741>
55. <https://channel9.msdn.com/Events/Build/2014/9-002>

## 9.5.2 /dev/summer had some good talks as usual (2015-06-30 07:00)

It was [1]/dev/summer at the weekend, where there were a few very interesting talks. Rather than going to the Clojure or Haskell tracks, this time I attended the double session on [2]Go and a couple of sessions by [3]Gleb Bahmutov .

The Go session took a little while to get going, with the first 45 minutes hardly touching the language after some questions around the directory structure of a Go project on the disc. This meant that we didn't get to see any large examples, thought the witty presentation and insightful answers to the questions led me to come away with basic Go knowledge that I can now try to put into practice.

The presentations by Gleb were full of good interesting ideas. [4]The first presentation looked at the issues around npm modules and their dependencies (though the ideas applied more generally), and how [5]semantic versioning isn't typically supported by the modules. The clever idea of the presentation [6]was a tool that tests semantic compatibility by running unit tests against the old and new versions of the module. Failures in the tests imply incompatibility and by having users send their results to a central server it is possible to check how safe a module upgrade is going to be. [7]The second presentation looked at using code snippets inside Chrome to get various metrics on the page such as load time and render time, and also showed us how to combine this with the various profiling tools. Having not used these tools before, this was a great introduction to speeding up a web application.

I also attended the lightning talks which were a useful and informative set on general open source and browser testing.

It was a little strange to come across these talks. I'd just been reading about module dependencies in the .Net Nuget world after the [8]publication of this post which talks about some changes to Nuget to make it support the new CoreClr world where the number of profiles is too great to simply name profiles using an integer value, and package authors will instead need to declare their dependencies. It also links to the [9]bait and switch technique. I was also doing some interesting reading on the new [10]WebAssembly technology and came across this [11]great interview with Brendan Eich that explains things well.

Two other interesting blog posts. A talk on [12]cloud scale event processing using Rx, which mentions some of the changes required to scale from the desktop version to a server based version - features needed included a way to checkpoint a query. This post [13]on Rust's type system is also interesting.

1. <http://devcycles.net/2015/summer/programme/>
2. <http://golang.org/>
3. <http://glebbahmutov.com/blog/>
4. <http://slides.com/bahmutov/self-improving-software#/>
5. <http://semver.org/>
6. <https://github.com/bahmutov/next-update>
7. <http://devcycles.net/2015/summer/sessions/index.php?session=33>
8. <http://blogs.msdn.com/b/mvpawardprogram/archive/2015/06/18/demystifying-pcls-net-core-dnx-and-uwp-redux.aspx>
9. <http://log.paulbetts.org/the-bait-and-switch-pcl-trick/>

10. <https://brendaneich.com/2015/06/from-asm-js-to-webassembly/>
  11. <https://medium.com/javascript-scene/why-we-need-webassembly-an-interview-with-brendan-eich-7fb2a60b0723>
  12. <http://www.infoq.com/presentations/cloud-rx>
  13. <http://theburningmonk.com/2015/05/rust-memory-safety-without-gc/>
- 

## 9.6 July

### 9.6.1 Advanced Topics In Types and Programming Languages (2015-07-09 06:47)

[1]Advanced Topics in Types and Programming Languages edited by Benjamin Pierce

I'd been meaning to read this book for a while, and managed to buy it with various present money after Xmas. It consists of ten chapters by different authors on ten topics ranging from type systems to proof-carrying code.

The first three chapters discuss various type systems - substructural type systems which control the use of variables in the type context and which lead to linear typing in one of the variants, dependent types where you are allowed to do calculations at the type level and which is being popularised by languages such as [2]Idris, and a very interesting chapter on effect systems which discusses the region-based memory management work on Tofte and Talpin. In the latter work, the type of a program contains information about the allocation contexts which can allow the runtime to manage dynamic memory using a stack based approach.

The next two chapters cover the low level use of types. Typed assembly language would allow assembly language to be typed to check properties such as memory safety, and proof carrying code allows an executable to contain proof of its own safety and enough information for the safety to be checked by the target that is going to execute the code.

The next two chapters cover methods for reasoning about programs using their types, followed by a fantastic section on types for programming in the large. This section contains a chapter on the design considerations of the ML type system - this was very interesting. This section also contains a chapter on type definitions.

The last chapter is on type inference and discusses the typing of ML programs using constraint solving, rather than the usual unification based Hindley-Milner approach.

The book contains loads of interesting idea, but some of the theory is perhaps a little off-putting. If you really want to grips with the details, there are lots of exercises, or you can choose to simply skim these and still keep up with the exposition.

1. [http://www.amazon.co.uk/Advanced-Topics-Types-Programming-Languages/dp/0262162288/ref=sr\\_1\\_fkmr1\\_2?s=book&ie=UTF8&qid=1436108305&sr=1-2-fkmr1&keywords=types+be](http://www.amazon.co.uk/Advanced-Topics-Types-Programming-Languages/dp/0262162288/ref=sr_1_fkmr1_2?s=book&ie=UTF8&qid=1436108305&sr=1-2-fkmr1&keywords=types+be)
2. <http://www.idris-lang.org/>

---

## **9.6.2 C# await inside catch and finally leads to some interesting semantics (2015-07-22 06:30)**

I remember when await was introduced into C # - the feature took us away from the callback hell that was developing, but it also required some understanding of the underlying code generation to really see what was going on in simple looking code. The abstraction led to a few rather confusing effects, such as always throwing the [1]first exception of an AggregateException in the await.

At the time people were interested in why you couldn't use await inside catch or finally blocks, and [2]the answer always came down to confusing semantics. In the recently released C # 6, await is now available in catch and finally blocks, so the question is how this has been achieved without breaking the existing semantics. And I think the answer is that the semantics of some C # forms has now been changed in a way that again requires knowledge of the underlying transforms to understand.

Let's take the async version of the standard thread abort construct. The call to Abort() notionally throws a ThreadAbortException which can be caught and processed by catch and finally blocks, but which has the interesting semantic of being re-raised when the processing block finishes (unless you reset the abort).

```
static
async
Task
AsyncVersion()

{
try
{
    Thread
    .CurrentThread.Abort();
}
finally
```

```
{  
  
Console  
.WriteLine(  
"Can't stop me!!"  
);  
  
}  
  
Console  
.WriteLine(  
"After abort"  
);  
  
}
```

We can explain the behavior of this code fairly easily. The Abort throws the ThreadAbortException, which is caught by the finally block. This finally block prints "Can't stop me!!" and then the exception is rethrown when the finally block finishes. Hence the "After abort" is never printed.

Now change the finally block to

```
finally  
{  
  
Console  
.WriteLine(  
"Can't stop me!!"  
);  
  
await  
Task  
.Run(() =>  
Console  
.WriteLine(  
"Next"  
));
```

```
}
```

When you run this version of the code nothing is printed, and it's hard to understand why without looking at the code generation. [3]The async method is translated to a class that implements a state machine and, crucially, the finally is no longer a finally in the generated IL code, but is instead translated to a catch block that handles all exceptions.

Reflector shows us:

```
try
{
    Thread.CurrentThread.Abort();
}
catch (object obj1)
{
    obj2 = obj1;
    this.<>s__1 = obj2;
}
```

ie the finally isn't processed as a CLR finally, but is instead translated to a catch which re-enters the state machine processing. Of course, this means that the ThreadAbort exception which is rethrown at the end of any catch block is going to be rethrown too early, avoiding the exception of the code in the finally block.

I can see that it is a benefit to be able to await in catch and finally blocks, and I guess we've had to accept in the past that C # is not the assembly language of .NET. However, the code generation approach to implementing these high level constructs means that sometimes we can see through the abstraction as we can here. I'm not quite sure how important that is.

1. <http://blogs.msdn.com/b/pfxteam/archive/2011/09/28/task-exception-handling-in-net-4-5.aspx>
2. <http://stackoverflow.com/questions/12879926/why-compiler-does-not-allow-using-await-inside-catch-block?rq=1>
3. <https://www.simple-talk.com/dotnet/.net-tools/c-async-what-is-it,-and-how-does-it-work/>

---

## 9.7 August

### 9.7.1 Some UWP and a little ClojureScript (2015-08-03 06:17)

I've been following [1]Lucian Wischik's interesting series of posts on writing Windows 10 applications in .NET. While I was on holiday he posted an article on writing [2]NuGet packages for UWP but unfortunately the article was removed from the web site when I was only half a way through it. The place holder mentions that the NuGet team were going to talk about the issues and, sure enough, [3]a post has now appeared on NuGet support for UWP in NuGet 3.1. In summary you will now be able to declare project references using a project.json (in much

the same way [4]as ASP.NET 5 projects). This will also use the transitive references idea, where you list only the top level projects that you require, and NuGet will take care of fetching their dependencies ([5]and also resolve the appropriate versions when there are conflicts), in contrast to the current model where dependencies of dependencies make their way into your packages.config.

In other news, [6]David Nolen has just announced on his blog that [7]ClojureScript, the variant of [8]Clojure that uses JavaScript as its target language, can now compile itself. Having been through the bootstrapping process myself for both ML and Common Lisp compilers, its always very satisfying when you can get rid of the other implementation, and finally get the compiler to work on itself. His post is rather cool, as it embeds some of the implementation of the translation process into the actual post, and you can run these in the browser that you are using to read the post. Some of this work sits on top of reader conditions, one of the two key features that were recently added to [9]Clojure 1.7 (the other being [10]transducers).

Finally, if Windows Update fails to install Windows 10 repeatedly), then t[11]his post has the answer, or rather it works for some people. For me, the relaunched Windows Update failed again, so in the end I needed to manually download by following this [12]link.

1. <http://blogs.msdn.com/b/lucian/archive/2015/05/18/win10-development-in-net-getting-started.aspx>
2. <http://blogs.msdn.com/b/lucian/archive/2015/07/27/writing-a-nuget-package-for-vs2015-rtm-and-uwp.aspx>
3. <http://blog.nuget.org/20150729/Introducing-nuget-uwp.html>
4. <https://www.simple-talk.com/dotnet/.net-framework/what-is-dnx/>
5. <http://docs.nuget.org/consume/ProjectJson-Dependency>
6. <http://swannodette.github.io/2015/07/29/clojurescript-17/>
7. <https://github.com/clojure/clojurescript>
8. <https://github.com/clojure/clojure>
9. <http://blog.cognitect.com/blog/2015/6/30/clojure-17>
10. <http://ignaciiothayer.com/post/Transducers-Are-Fundamental/>
11. [http://answers.microsoft.com/en-us/windows/forum/windows\\_10-win\\_upgrade/windows-10-failed-to-install-error-code-80240020/9d69a99b-1851-4464-b10b-e69946993e78](http://answers.microsoft.com/en-us/windows/forum/windows_10-win_upgrade/windows-10-failed-to-install-error-code-80240020/9d69a99b-1851-4464-b10b-e69946993e78)
12. <https://www.microsoft.com/en-us/software-download/windows10>

---

### 9.7.2 CLR Load contexts can make things confusing (2015-08-05 06:17)

I've never really spent a lot of time thinking about CLR binding contexts, but a colleague at work had a problem related to them that took a bit of thought to figure out. These posts, one on [1]stackoverflow and one related to [2]why adding to the private path of an AppDomain has been deprecated, help explain part of the issue and also why there is (maybe) a need for different contexts.

My colleague was writing a Visual Studio extension. The extension was loaded by VS, and his code created a new AppDomain and used remoting to create an application object in the AppDomain. He tried to cast the result of [3]CreateInstanceOfAndUnwrap into the type of his remote class and got an InvalidCastException.

After some investigation it turned out that this was a problem with type loading. The returned transparent proxy needs to validate the cast that is being made, which it does in the following methods (which were part of the stacktrace I grabbed using WinDbg and SOS).

```
System.RuntimeTypeHandle.GetTypeByName(...)System.RuntimeTypeHandle.GetTypeByName(...)System.Runtime.Remoting.RemotingServices.LoadClrTypeWithPartialBindFallback(...)System.Runtime.Remoting.RemotingServices.Internal GetTypeFromQualifiedTypeName  
System.Runtime.Remoting.TypeInfo.CanCastTo(...)  
System.Runtime.Remoting.Proxies.RemotingProxy.CanCastTo(...)System.Runtime.Remoting.RemotingServices.CheckCast(...)
```

The issue was that the initial assembly was loaded using LoadFrom, but when remoting returns a type it is going to be loaded into the Load context as remoting will use the name of the type to try to find it. This is essentially the same effect that you get if you run the following code:

```
var  
assembly1 =  
Assembly  
.LoadFrom(  
  
@"C:\Users\clive\documentskslashvisual studio 2013\Projects\ClassLibrary3\ClassLibrary3\bin\debug\classlibrary3.dll"  
);  
  
var  
fullName = assembly1.FullName;  
  
var  
assembliesLoaded =  
new  
List  
<  
Assembly  
>(  
AppDomain  
.CurrentDomain.GetAssemblies());
```

```
Debug  
.Assert(assembliesLoaded.Any(assembly => assembly.FullName == fullName));
```

```
var  
assembly2 =  
Assembly  
.Load(fullName);
```

The assembly is part of the AppDomain's assembly collection, but you cannot load it by name using Assembly.Load. In the above code we get a FileNotFoundException, which is the underlying exception that causing the cast to fail in the remoting example.

What's the fix?

Well we have the assembly object in the collection of Domain assemblies, so we should just return it if the assembly resolution fails. If we add the following code in front of the Assembly.Load, then the assembly is loaded as expected.

```
AppDomain  
.CurrentDomain.AssemblyResolve += (o, eventArgs) =>  
{  
  
    var  
    assembliesInDomain =  
    new  
    List  
<  
    Assembly  
>(br/>    AppDomain  
.CurrentDomain.GetAssemblies());  
  
    return  
    assembliesInDomain.FirstOrDefault(  
    assembly => assembly.FullName == eventArgs.Name);  
};
```

Essentially, we take control of the load resolution in the case where the CLR is trying to protect us by using the different contexts.

1. <http://stackoverflow.com/questions/2493786/what-are-3-kinds-of-binding-contexts-for>
  2. <http://blogs.msdn.com/b/dotnet/archive/2009/05/14/why-is-appdomain-appendprivatepath-obsolete.aspx>
  3. [https://msdn.microsoft.com/en-us/library/system.appdomain.createinstanceandunwrap\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.appdomain.createinstanceandunwrap(v=vs.110).aspx)
- 

### 9.7.3 Modern C++ Design: magic with templates (2015-08-07 06:05)

[1]Modern C++ Design: Generic Programming and Design Patterns Applied by Andrei Alexandrescu

I've done some C++ programming in my time which has required me to write some small simple templates. However, just before going on holiday I noticed this book on the work bookshelf and thought it deserved a read. In the past, I'd come across posts implementing simple functions such as Fibonacci in C++ using templates in a way that gets them evaluated at compile time, but this book takes this much further and shows the tremendous power of type level computation during the compilation phase.

On reading the book, one important message came through - there's [2]a pattern matching functional language embedded into the C++ compiler which can be used to do some truly impressive feats, including many instances of compile time code generation of highly optimized code. The astounding creativity of the author, and the edge semantics of C++ that are needed to understand some of the constructs, makes this a fascinating read. Though I'm not sure I'll ever get to use some of the items in the applications that I write.

Chapter one is a great read. The author discusses where features like multiple inheritance and templates fail when they are used on their own. He then goes on to explain where their combination can be very powerful - after all, the template expansion process can generate new classes and can determine how they are going to be mixed together in the multiple inheritance hierarchy. He calls this "policy-based class design", and it's a very clever technique. Chapters two and three introduce the main techniques that will be used later. Chapter two goes through some advanced features of C++ templates including partial template specialization and local classes, as well as ways for detecting convertibility and inheritance at compile time. Chapter three describes a `TypeList` datatype which is used in many of the techniques to allow template expansion to work through the items of a list of types generating things as it goes. These two chapters are truly fascinating, and I wish I could fully understood all of the concepts that they cover.

The other 8 chapters of the book use some of these techniques to implement a number of patterns - small-object allocation, generalised Functors (the command pattern on steroids), implementation techniques for singletons emphasizing how policy selection allows you to pick implementation trade offs, smart pointers, object factories, abstract factories, the visitor pattern and multi-methods. In each case the author describes the problem, works thorough some of the various design decisions, and then offers several solutions. Many of these

solutions build on the concepts in the first few chapters.

I learned loads about C++ in general, lots more about templates and got to grips with ideas around template metaprogramming. Some of the design discussions are also really interesting. Brilliant!

1. [http://www.amazon.co.uk/gp/product/0201704315?keywords=modern%20C%2B%2Bdesign&qid=1438527875&ref\\_=sr\\_1\\_1&s=books&sr=1-1](http://www.amazon.co.uk/gp/product/0201704315?keywords=modern%20C%2B%2Bdesign&qid=1438527875&ref_=sr_1_1&s=books&sr=1-1)
  2. <http://www.vandenoever.info/blog/2015/07/12/translating-haskell-to-c++.html>
- 

#### **9.7.4 Adaptive Code Via C#: Agile coding with design patterns and SOLID principles (2015-08-31 07:17)**

[1]Adaptive Code Via C #: Agile coding with design patterns and SOLID principles by Gary McLean Hall

This is very much a book of three parts - a section named "An Agile Foundation" that covers basic scrum and some of the basic principles of modern software engineering, a section going into detail about each of the SOLID principles, and a section called "Adaptive sample" that tries to demonstrate how the techniques in the book might be used across a number of sprints, by describing two sprints worth of development on a chat client.

The first section is an introduction to agile practices. There is an introduction to Scrum which covers the roles and phases of a scrum project, emphasising how interaction with the customer drives the development process. This is followed by a chapter on dependencies and layering which goes into detail about how you should manage your project dependencies and even gives some brief coverage of topics such as aspect-oriented programming. Chapter three goes through interfaces and design pattern, showing how useful interfaces can be in the .NET world. Chapter four covers unit testing and refactoring - these two items are closely related as the tests give you confidence that refactoring have not broken anything, allowing you to tidy working code without risk. All interesting material which includes some good observations.

The second section of the book, and by far the majority of the book, covers writing SOLID code. The author takes each of the solid principles in turn with a chapter on each. The author discusses the meaning of the principle and then demonstrates its use. On the way there's often lots of additional material. The chapter on the single responsibility principle, for example, has a lot of extra material on the decorator pattern, showing how they can be used for logging and the introduction of asynchrony, and the chapter on the Liskov substitution principle has some material on pre- and post-conditions. This section of the book is really very good.

I was less convinced by the adaptive sample. We follow conversations between team members as they write a simple chat application. This covers the initial planning for a sample application, and then the team carries out two sprints. The code is available on github so that you can follow along, but it felt like a lot of effort to understand the requirements and the way

that the team was working.

The first two sections of the book are really good though, and I certainly learned something from each chapter.

1. [http://www.amazon.co.uk/Adaptive-Code-via-Interface-Principles/dp/0735683204/ref=sr\\_1\\_1?ie=UTF8&qid=1438532757&sr=8-1&keywords=adaptive+code+via+C%23](http://www.amazon.co.uk/Adaptive-Code-via-Interface-Principles/dp/0735683204/ref=sr_1_1?ie=UTF8&qid=1438532757&sr=8-1&keywords=adaptive+code+via+C%23)
- 

## 9.8 October

### 9.8.1 Some interesting .NET videos (2015-10-19 08:01)

Channel 9 have just published [1]a great video with Mads Torgersen. In it he covers some of the features of C # 6, mentioning the [2]new FormattableString class to which an interpolated string expression can be cast to in order to access additional functionality. Most of the video is about some potential C # 7 features, which include pattern matching and valuetype tuples whose type includes the name of the item in the tuple. There will be work on improving the performance of structs by allowing the return of structures by reference. There is also talk of more research into reference types that do not include the null value. Torgersen also touches on [3]the expression problem when discussing the emphasis on including more functional features into the language.

On a related.NET note, [4]there's this recording of David Fowler talking about the internals of the new ASP.NET. He follows a request as it makes its way through the new platform, including a dive into the internals of the new [5]Kestrel web server, which uses [6]libuv to allow the same code to work on Windows and Linux. The leastprivilege blog had [7]a recent post that looks at authorisation in this new framework.

Its also interesting that people are starting to submit pull requests for language and CLR changes - examples include pull requests like [8]this and [9]this.

1. <https://channel9.msdn.com/Blogs/Seth-Juarez/Looking-Ahead-to-C-7-with-Mads-Torgersen>
  2. <http://blog.somewhatabstract.com/tag/formattablestring/>
  3. [https://en.wikipedia.org/wiki/Expression\\_problem](https://en.wikipedia.org/wiki/Expression_problem)
  4. <https://vimeo.com/142347212>
  5. <https://github.com/aspnet/KestrelHttpServer>
  6. <https://en.wikipedia.org/wiki/Libuv>
  7. <http://leastprivilege.com/2015/10/12/the-state-of-security-in-asp-net-5-and-mvc-6-authorization/>
  8. <https://github.com/dotnet/coreclr/pull/1684>
  9. <http://xoofx.com/blog/2015/10/08/stackalloc-for-class-with-roslyn-and-coreclr/>
-

## 9.9 November

### 9.9.1 Some good reads on Angular 1.x (2015-11-24 07:31)

I decided it was time to learn one of the many JavaScript SPA frameworks, and figured that it would be a good idea to have a look at [1]Angular.

Of course there have been many posts in the past detailing problems in Angular such as [2]this one, [3]this one and [4]this one. There are also a number of articles discussing the good parts of the framework such as [5]this one and [6]this one. There is also the rewrite as [7]Angular 2.0 going on which [8]seems to be progressing well with its emphasis on TypeScript and ES6.

I haven't had time to write anything large in this framework, but have been impressed with the design of the framework and the clever ideas that it incorporates. For me, it was also a chance to get to grips with modern JavaScript development using npm and the associated tools.

The first book I read was [9]AngularJS Up & Running: Enhanced productivity With Structured Web Apps by Shyam Seshadri & Brad Green.

This book takes you through the facilities that the Angular framework offers, and is filled with examples that can be downloaded from one of the author's GitHub repository. It starts with a quick introduction to Angular and writes a basic AngularJS Hello World. Chapter 2 goes into directives and controllers, concentrating on an app that displays a collection of data items using databinding and the ng-repeat directive. Chapter 3 covers unit testing using Karma and Jasmine. Angular, with its inbuilt dependency injection makes it easy to do unit testing of your controllers. Chapter 4 touches on Forms and Inputs, and then moves on to the subject of services which are covered in more detail in chapter 5, where the authors discuss the differences between services and controllers.

Chapter 6 covers HTTP communication with the server and Chapter 7 discusses unit testing the server calls that your application is making. Chapter 8 covers filters and Chapter 9 covers how you unit test them. Chapter 10 discusses the ngRoute module, and how it helps you implement history and SEO for your application. Chapter 11 goes into directives in more details and Chapter 12 covers how you unit test them.

Chapter 13 covers directives in more details, and covers the Angular life cycle, such as the digest cycle, in more detail. This is followed by Chapter 14 that covers end-to-end testing using Protractor, and the final chapter gives some guidelines and best practices.

I thought this book was a good introduction, and it was good having the examples to play with in a browser on my laptop. I enjoyed the details about how the framework worked at the low level, and it was good that the authors demonstrated how you might convert a downloaded JavaScript component, a slider, into an Angular component.

While I was trying to get a deeper understanding of the Angular framework, I came across this [10]sample chapter from the book Build Your Own AngularJS by Tero Parviainen. The chapter seemed to explain some of the details that I needed to understand Angular a bit better, so

I bought the whole book. I'm very glad I did. The book is great from the point of view of understanding Angular, but also as a means of getting into JavaScript development. The author develops a variant of the Angular framework in a test driven fashion using npm and its associated libraries.

You get over a thousand pages of detailed JavaScript development which works through most of the features of Angular, starting with Scopes, which were covered in the sample chapter I linked to earlier, and moving all the way up to directives. The whole book is thoroughly interesting and the author explains the framework very well. Moreover, you can work through the examples too by downloading the code from the author's GitHub repository.

There is really nothing like implementing something to get an understanding of how it all works. The book covers the whole dependency injection framework that underlies Angular in great detail, and mentions features like decorators that were only sparsely covered in the higher level book. The whole dependency injection idea makes the framework very customisable, and it's great that the whole system is built upon this framework with all of the system components being made available by dependency resolution. Moreover, the Angular implementation of promises was much more understandable when we got into the low level implementation details - particularly promise rejection, and the underlying implementation of deferreds. I can't recommend this book enough.

There are also a number of good Angular podcasts around including [11]Angular Air and [12]Adventures in Angular. And for a very brief introduction there's Dan Wahlin's [13]AngularJS in 20 minutes.

1. <https://en.wikipedia.org/wiki/AngularJS>
2. [http://www.quirksmode.org/blog/archives/2015/01/the\\_problem\\_with.html](http://www.quirksmode.org/blog/archives/2015/01/the_problem_with.html)
3. <https://medium.com/@mnemonick/why-you-should-not-use-angularjs-1df5ddf6fc99>
4. <http://larseidnes.com/2014/11/05/angularjs-the-bad-parts/>
5. <http://weblogs.asp.net/dwahlin/what%20%99s-%E2%80%9Cright%E2%80%9D-with-angularjs>
6. <http://www.johnpapa.net/why-does-there-have-to-be-something-wrong-with-angularjs/>
7. <http://developer.telerik.com/featured/will-angular-2-be-a-success-you-bet/>
8. <https://blog.angularjs.org/2014/03/angular-20.html>
9. [http://www.amazon.co.uk/AngularJS-Running-Enhanced-Productivity-Structured/dp/1491901942/ref=sr\\_1\\_sc\\_1?s=books&ie=UTF8&qid=1448138158&sr=1-1-spell&keywords=angu](http://www.amazon.co.uk/AngularJS-Running-Enhanced-Productivity-Structured/dp/1491901942/ref=sr_1_sc_1?s=books&ie=UTF8&qid=1448138158&sr=1-1-spell&keywords=angu)
10. [http://www.teropa.info/build-your-own-angular/build\\_your\\_own\\_angularjs\\_sample.pdf](http://www.teropa.info/build-your-own-angular/build_your_own_angularjs_sample.pdf)
11. <https://angularair.com/>
12. <https://devchat.tv/adventures-in-angular/>
13. <https://www.youtube.com/watch?v=tnX0-i7944M>

## 9.10 December

### 9.10.1 Sometimes it's not good to build on other libraries (2015-12-24 08:03)

The implementation of async/await in C # 7 is very complicated, and there are a number of places where the use of code generation and the use of underlying libraries shines through to the implementation. I did a lightning talk at work about this subject, which is available [1]here.

In the talk, I mentioned that it is a shame that there isn't metadata for items such as lambda expressions, requiring tools to infer that classes are the manifestation of lambda expressions by looking for patterns. It was interesting to [2]see Joe Duffy mention that the encoding of lambda expressions as instances of compiler generated classes leads to some issues when trying to make compiled coded fast in another of his excellent posts on Midori.

There's a good talk on the design process of C # 7 by Lucian Wischik [3]here and the [4]ASP.NET Fall Sessions published on Channel 9 have some good talks on the future of the .NET platform including [5]the command line tools and [6]the future of packages. It was also good to have a talk on where the [7]Kestrel web server fits into the picture.

1. <https://onedrive.live.com/embed?cid=3F21DF299C355E7F&resid=3F21DF299C355E7F%216301&authkey=APvfSh0jUx9rUxM&em=2>
  2. <http://joeduffyblog.com/2015/12/19/safe-native-code/>
  3. <http://www.infoq.com/presentations/design-c-sharp-7>
  4. <https://channel9.msdn.com/events/ASPNET-Events/ASPNET-Fall-Sessions>
  5. <https://channel9.msdn.com/Events/ASPNET-Events/ASPNET-Fall-Sessions/Introducing-the-dotnet-CLI>
  6. <https://channel9.msdn.com/Events/ASPNET-Events/ASPNET-Fall-Sessions/Class-Libraries>
  7. <https://channel9.msdn.com/Events/ASPNET-Events/ASPNET-Fall-Sessions/ASPNET-5-Kestrel>
-



# 2016

## 10.1 February

### 10.1.1 Actors have become popular again (2016-02-02 07:00)

[1]Reactive messaging patterns with the Actor model: Applications and Integration with Scala and Akka by Vaughn Vernon

I very much enjoyed this book's discussion of Actors and the reasons why the [2]Actor model is well matched with modern applications, though I enjoyed the book less when it got into the various patterns.

Chapter one talks about the design of modern Reactive applications and then discusses the origin of the Actor model. It's vey good. Chapter two gives a brief introduction to the Scala programming language and the Akka library... actor systems, supervision, remoting and clustering are all explained really well. Chapter three discusses the need for scalability with a good discussion of how clock speed is no longer increasing but instead we are given many more cores for our applications to use. It is hard to write multi-threaded code, and the Actor model provides a simple model where we don't need to worry about memory barriers and locking - though the issues associated with locking including deadlock and live lock can still manifest themselves in an application at higher levels.

From Chapter four onwards, the author lists a large number of patterns for using actors. There's a discussion about each of the patterns and an example written using the Akka test framework that demonstrates it. Patterns such as Message Router, Publish Subscribe Channel, Scatter-Gather and Service Activator. I must admit that I found it hard work to go through each of the patterns, read the discussion and then understand some of the multipage examples... I will probably go back to some of the examples when I get more time.

I did learn a lot along the way. For example, I was interested how a modern application's need for guaranteed message processing would map to actors and their message delivered "at least once" guarantees. This is covered by the Guaranteed Delivery pattern which uses the[3] Akka persistence and AtLeastOnce traits to store messages that haven't been acknowledged and which hooks the Actor restore protocol to ensure that nothing gets lost if actors are restarted.

After reading this book, I went and read one of the early Phd theses on the subject - Gul Agha -**Actors: A Model of Concurrent Computation in Distributed Systems**. There are versions

of this available for download if you Google. It's interesting to read how the Actor model was pushed as a model for computation, with lots of effort in the thesis to give the model a rigorous semantics, and also interesting to see the emphasis on the become operation which allows the Actor to change the message processing function. This feature is not something that gets pushed in the modern interpretation of the model.

There is an [4]open source project to make the Akka framework on the .NET platform.

There is also [5]Pony, a language and runtime with Actors at the very core. I arrived at it via [6]this interesting talk. There is a lot of interesting technology associated with the language - such as a type system that makes it possible to prevent aliasing of message data, as well as interesting features of the runtime such as the way [7]it detects unreferenced actors.

1. [http://www.amazon.co.uk/Reactive-Messaging-Patterns-Actor-Model/dp/0133846830/ref=sr\\_1\\_1?ie=UTF8&qid=1454010255&sr=8-1&keywords=actor+patterns](http://www.amazon.co.uk/Reactive-Messaging-Patterns-Actor-Model/dp/0133846830/ref=sr_1_1?ie=UTF8&qid=1454010255&sr=8-1&keywords=actor+patterns)
  2. [https://en.wikipedia.org/wiki/Actor\\_model](https://en.wikipedia.org/wiki/Actor_model)
  3. <http://doc.akka.io/docs/akka/snapshot-scala/persistence.html>
  4. <https://github.com/akkadotnet/akka.net>
  5. <http://www.ponylang.org/>
  6. [https://www.youtube.com/watch?v=KvLjy8w1G\\_U](https://www.youtube.com/watch?v=KvLjy8w1G_U)
  7. <https://github.com/ponylang/ponylang.github.io/blob/master/papers/OGC.pdf>
- 

### 10.1.2 Another good DevWinter (2016-02-04 07:07)

It was [1]DevWinter last weekend, and as usual there was an [2]interesting set of talks.

The "my adventure with elm" talk was good. The speaker gave a very brief introduction to Reactive programming, a brief introduction to the Elm language and then implemented the Snake-and-apples game in Elm inside a browser on one of the [3]Try Elm sites. Apart from a couple of times when he needed to uncomment a couple of pre-written functions, he wrote the whole application in front of us. This was a good introduction to the language and a very well presented talk.

The "anatomy of a catastrophic performance problem" was also very good. A witty presentation of a real life performance problem, which showed how frequently us developers think we've analysed the problem and hence push out a fix, without trying the fix on a reproduction [which wasn't available in this case].

In the afternoon, I attended the double session on [4]ClojureScript. This wasn't very hands on in the end, and the presenter spent a lot of time discussing the Clojure language and the various benefits of functional styles of programming as well as selling the advantages of transpiling to JavaScript rather than writing in JavaScript in the first place. The presenter did use the lein [5]Figwheel plugin to get a REPL connected to a ClojureScript session running inside a browser which also reloads when modifications are made to the source. This is all

build using the [6]lein tool, and getting started is as simple as typing:

```
lein new figwheel myproject
cd myproject
lein figwheel
... wait for compilation to happen
Open browser on port http://localhost:3449
... and the REPL will connect to the browser
(js/alert "Hello")
... and the alert dialog is raised inside the browser
```

If you then open the developer console in your browser and make an edit to any of the project's ClojureScript files, you will see the modified code reloaded into the browser session allowing quick development turnaround.

The best talk of day was "my first unikernel with mirage os" by Matthew Grey. This was a hands on session based on the code in the speaker's [7]GitHub repository. I'd been meaning to play with [8]Mirage for some time, as it is a perfect match with some of my interests - operating systems written in functional languages [which [9]I first read about when I spent a year working in industry before university where I spent some free time playing with [10]LispKit lisp] and hypervisors. The idea is that you can take an application written in OCaml and produce a [11]Unikernel from it. The Mirage team have made it possible to deliver a UniKernel, that runs on top of [12]Xen, very easily from an OCaml application. The Mirage team have also implemented various utilities such as a [13]TLS library and web server that your application can use. Matt Gray's repository contains [14]a vagrant script that can be used to get a Ubuntu development environment that is suitable for playing around with Mirage. Once you have this running inside VirtualBox, it is easy to get the various examples running.

The speaker gave a brief overview of Unikernels and then helped the audience to get going. There was another talk on Mirage in the afternoon, but I didn't attend that.

What did I enjoy about DevWinter? The range of the talks, on all kinds of topics. Unlike the typical Microsoft event I go to, the talks cover a range of topics that are interesting and are aimed at what might happen in the future. I also very much enjoyed the developer experience talks. A very nice venue makes this a great way to spend a Saturday twice a year.

1. <http://devcycles.net/2016/winter/>
2. <http://devcycles.net/2016/winter/programme/>
3. <http://elm-lang.org/try>
4. <https://github.com/clojure/clojurescript>
5. <https://github.com/bhauman/lein-figwheel>
6. <http://leinigen.org/>
7. <https://github.com/mattgray/devwinter2016>
8. <https://mirage.io/>
9. <https://www.cs.ox.ac.uk/files/3313/PRG42.pdf>
10. [https://en.wikipedia.org/wiki/Lispkit\\_Lisp](https://en.wikipedia.org/wiki/Lispkit_Lisp)
11. <https://en.wikipedia.org/wiki/Unikernel>
12. <https://en.wikipedia.org/wiki/Xen>
13. <https://mirage.io/blog/introducing-ocaml-tls>

14. <https://github.com/mattgray/mirage-xen-virtualbox>

---

### **10.1.3 Your .NET applications can start being more native (2016-02-08 07:30)**

I was interested in trying out the dotnet command line interface and seeing how it all works. Microsoft have after all just told us that they will be delaying the release of DNX to allow the integration into the CLI model. You can download a build of the command line tools from [1]here and it's really easy to get going.

You can generate a demonstration "hello world" project using

`dotnet new`

You then get all of the packages it depends on using

`dotnet restore`

And then build it using

`dotnet compile`

Now it's business as usual and you can run it as normal

```
cd bin\Debug\dnxcore50
```

dotnet2.exe [named as this because I was in a folder named dotnet2 when I created the project]

What excited me more about the command line tools though, is that they now have started offering the opportunity to compile .NET applications to native code. Be warned though that this is very early functionality and they only guarantee support for small hello world applications.

You can choose to compile using an ahead of time version of the normal JIT compiler [on x64] or can choose to go via generated C++ code. I wanted to see the kind of native code that can be produced and therefore chose the latter option.

```
dotnet compile -native -cpp
```

[Note that you have to be in a VS2015 x64 Native Tools command window to get the right tools available on the PATH]

This generates an executable in a native subdirectory of bin\debug\dnxcore50\native

dotnet2.exe

which runs very quickly - there's no jittering in order for the application to get running, and it is noticeably quicker on my fairly old laptop.

The demo application is a very simple hello world, and you can find the emitted C++ in the directory

obj/Debug/dnxcore50/native/dotnet2.cpp

I was interested in how the GC got linked into the project, particularly as I had heard of [2]CoreRT and couldn't see any appropriately named DLL when I attached windbg to the running executable.

I therefore modified the code to generate garbage [and built it in a folder named dotnet so ended up with an application named dotnet.exe]

```
public static void Main(string[] args)
{
    Console.WriteLine("Hello World!");
    for (int i = 0; i < 100000; i++)
    {
```

```
var x = new object();
}
Console.ReadLine();
}
```

The generated Main method takes the form

```
#line 8 "C:\\Users\\clive.ong\\Desktop\\dotnet t\\Program.cs"
void dotnet::ConsoleApplication::Program::Main(System _Private _CoreLib::System::String _Array* args) {int32 _t i=0; System _Private _CoreLib::System::Object* x=0; uint8 _t _I2=0;
_bb0: {
```

```
#line 8 "C:\\Users\\clive.ong\\Desktop\\dotnet t\\Program.cs"
```

```
#line 9 "C:\\Users\\clive.ong\\Desktop\\dotnet t\\Program.cs"
void* _1= __load_string_literal("Hello World!"); System _Console::System::Console::WriteLine
_13((System _Private _CoreLib::System::String*) _1);
#line 10 "C:\\Users\\clive.ong\\Desktop\\dotnet t\\Program.cs"
i=0; { goto _bb28; }; } _bb16: {
```

```
#line 11 "C:\\Users\\clive.ong\\Desktop\\dotnet t\\Program.cs"
```

```

#line 12 "C:\\Users\\clive.t ong\\Desktop\\dotne t\\Program.cs"
void* _8=__allocate_object(System_Private_CoreLib::System::Object::__getMethodTable());
System_Private_CoreLib::System::Object::__ctor((System_Private_CoreLib::System::Object*)_8);
x=(System_Private_CoreLib::System::Object*)_8;
#line 13 "C:\\Users\\clive.t ong\\Desktop\\dotne t\\Program.cs"

#line 10 "C:\\Users\\clive.t ong\\Desktop\\dotne t\\Program.cs"
int32 _t_10=i; int32 _t_11=_10+1; i=_11; } _bb28: {

#line 10 "C:\\Users\\clive.t ong\\Desktop\\dotne t\\Program.cs"
int32 _t_3=i; int32 _t_4=_3<100000; _I2=_4; int32 _t_6=_I2; if (_6!=0) { goto _bb16; }; }
_bb40: {

#line 14 "C:\\Users\\clive.t ong\\Desktop\\dotne t\\Program.cs"
void* _7=System_Console::System::Console::ReadLine();
#line 15 "C:\\Users\\clive.t ong\\Desktop\\dotne t\\Program.cs"
return; } }

```

Unfortunately, running the application with a debugger attached I got an access violation

```

dotnet!WKS::gc_heap::mark_object_simple1+0x180:
00007ff6'c2a4a460 4d8b01 mov r8,qword ptr [r9] ds:0000003c'00032000=?????????????????
0:000> k
# Child-SP RetAddr Call Site
00 0000003c'7715f3c0 00007ff6'c2a4ae60 dotnet!WKS::gc_heap::mark_object_simple1+0x180
01 0000003c'7715f430 00007ff6'c2a2e850 dotnet!WKS::gc_heap::mark_object_simple+0x1e0
02 0000003c'7715f480 00007ff6'c2a283be dotnet!WKS::GCHeap::Promote+0x90

```

```

03 0000003c'7715f4b0 00007ff6'c2a21aa9 dotnet!GcBulkEnumObjects+0x2e
04 0000003c'7715f4e0 00007ff6'c2a14c6c dotnet!Module::EnumStaticGCRefs+0x69
05 0000003c'7715f540 00007ff6'c2a4b233 dotnet!RuntimeInstance::EnumAllStaticGCR - 
efs+0x6c
06 0000003c'7715f5a0 00007ff6'c2a43606 dotnet!WKS::gc_heap::mark_phase+0x193
07 0000003c'7715f630 00007ff6'c2a432e3 dotnet!WKS::gc_heap::gc1+0xd6
08 0000003c'7715f690 00007ff6'c2a2d823 dotnet!WKS::gc_heap::garbage_collect+0x753
09 0000003c'7715f6f0 00007ff6'c2a5a629 dotnet!WKS::GCHeap::GarbageCollectGenera-
tion+0x303
0a 0000003c'7715f740 00007ff6'c2a2c4ee dotnet!WKS::gc_heap::try_allocate_more_
space+0x1b9
0b 0000003c'7715f780 00007ff6'c2a194ac dotnet!WKS::GCHeap::Alloc+0x5e
0c 0000003c'7715f7b0 00007ff6'c29d9b53 dotnet!RhpNewFast+0x5c
0d 0000003c'7715f7e0 00007ff6'c29d3132 dotnet!System_Private_
CoreLib::System::Runtime::InternalCalls::RhpNewFast+0x13 [c:\users\clive.tong\desktopbackslashdot-
debug\dnx core50\native\dotnet.cpp @ 39893]
0e 0000003c'7715f810 00007ff6'c29ee993 dotnet!System_Private_
CoreLib::System::Runtime::RuntimeExports::RhNewObject+0x92
[c:\users\clive.tong\desktopbackslashdotnet\obj\debug\dnx core50\native\dotnet.cpp @ 37660]
0f 0000003c'7715f890 00007ff6'c29dc783 dotnet!RhNewObject+0x13
[c:\users\clive.tong\desktopbackslashdotnet\obj\debug\dnx core50\native\dotnet.cpp @ 37666]
10 0000003c'7715f8c0 00007ff6'c29d26fe dotnet!dotnet::ConsoleApplication::Progr -
am::Main+0x53 [c:\users\clive.tong\desktopbackslashdotnet\program.cs @ 12]
11 0000003c'7715f930 00007ff6'c29ef1ea dotnet!dotnet::_Module_::StartupCodeMain+0x4e
[c:\users\clive.tong\desktopbackslashdotnet\obj\debug\dnx core50\native\dotnet.cpp @ 37467]
12 0000003c'7715f980 00007ff6'c2a62718 dotnet!main+0x4a [c:\users\clive.tong\desktopbackslash-
ram.cs @ 5676]
13 (Inline Function) ----- dotnet!invoke_main+0x22 [f:\dd\vctools\crt\vcstartup\src\startup-
\exe_common.inl @ 74]
14 0000003c'7715f9d0 00007ff9'd5872d92 dotnet!_scrt_common_main_seh+0x124
[f:\dd\vctools\crt\vcstartup\src\startup\exe_common.inl @ 264]
15 0000003c'7715fa10 00007ff9'd5b39f64 KERNEL32!BaseThreadInitThunk+0x22
16 0000003c'7715fa40 00000000'00000000 ntdll!RtlUserThreadStart+0x34

```

Notice that the source locations show that many of the frames are compiled versions of the C++ code that we found in the obj directory, but the other parts of the runtime are linked in and there is no source location.

Running the command line with the -v option, you can see the files that are passed into the cpp compiler, and you can see that lots of them come from the dotnet sdk directory

```
Running      C:\Program      Files      (x86)\Microsoft      Visual      Studio      14.0\Com-
mon7\Tools\..h..\VC\bin\amd64\ link.exe      "/NOLOGO"      "/DEBUG"      "/MANIFEST:NO"
"/IGNORE:4099"      "/out:C:\Users\clive.tong\Desktop      textbackslashdotnet\bin\Debug\
dnxcore50\native\dotnet.exe"      "kernel32.lib"      "user32.lib"      "gdi32.lib"      "winspool.lib"
"comdlg32.lib"      "advapi32.lib"      "shell32.lib"      "ole32.lib"      "oleaut32.lib"      "uuid.lib"
"odbc32.lib"      "odbccp32.lib"      "C:\Program      Files\dotnet\bin\sdk\PortableRuntime.lib-
"      "C:\Program      Files\dotnet\bin\sdk\bootstrappercpp.lib"      "/MACHINE:x64"
"C:\Users\clive.tong\Desktop\packslashdotnet\obj\Debug\dnxcore50\native\dotnet.obj"
```

The PortableRuntime here is the actual CoreRT code. If you clone the [3]CoreRT project and build using the build.cmd script, you can indeed make your own version of this SDK.

It includes code such as a [4]garbage collector and a [5]runtime that is fairly sophisticated, containing such features as thread hijacking which is implemented in this [6]file . There are loads of interesting comments if you browse the code.

It will be interesting to see how this portable runtime turns out... it's certainly true that no jittering makes start up time much better, but it isn't clear to me how features like Reflection are going to be supported. It will also be interesting to see how the debugging experience works when debugging AOT compiled code.

While we are on the subject of .NET, I'd like to recommend these talks from NDC London. A talk about the [7]new CLI and a talk on the [8]history of ASP.NET which goes into some detail about how things have changed over the years. There is also this talk on the [9]implementation of SignalR from several years ago.

1. <https://github.com/dotnet/cli>
  2. <https://github.com/dotnet/corert>
  3. <https://github.com/dotnet/corert>
  4. <https://github.com/dotnet/corert/tree/master/src/Native/gc>
  5. <https://github.com/dotnet/corert/tree/master/src/Native/Runtime>
  6. <https://github.com/dotnet/corert/blob/master/src/Native/Runtime/thread.cpp>
  7. <https://vimeo.com/153212604>
  8. <https://vimeo.com/154034601>
  9. <https://vimeo.com/68383353>
- 

#### **10.1.4 What do I need to know about Bitcoin? (2016-02-22 07:27)**

There have been several bitcoin discussions at work of late... what actually is [1]Bitcoin? How can the underlying [2]blockchain technology be used for other purposes? I did the [3]Coursera course on the topic some time ago, but these questions pushed me to go a little deeper into the world of BitCoin. I'd thought I'd try to summarise the articles I found useful here.

First, [4]the book associated with the Coursera course has just been made available. I thought this book was a great read. It covers the basic cryptography that you need to understand, and gives some example crypto currencies which don't quite work, before moving on to explain how [5]the original bitcoin design fixed these issues. The course is very broad, with lectures discussing the motivation of the miners, who need to devote large amounts of computing power [and hence resources] to the network, in the hope of being the chosen node who generates the next accepted block of the transaction chain, and who can therefore gain the reward for the block including the associated transaction fees.

The course also covers the tricks of representation which can be used to allow the block chain to represent the ownership of various assets. I'd overlooked this aspect of the system when I first did the course, but the ability to associate bits of extra data with a transaction has led to extra uses for the block chain. For a small transaction fee that is paid to the miners, a user can record information, say a cryptographic hash of a document, into the block chain, and the permanent ledger property of the block chain can be used to prove that the document was published at a time in the past.

For an interesting idea of the history, I next read Nathaniel Popper's Digital Gold: [6]Bit-Coin and the Inside Story of the Misfits and the Millionaires Trying To Reinvent Money. This book covers the story, from the initial idea and paper published on a mailing list by [7]Satoshi Nakamoto to the set of programmers who picked up on it and used their computers for the initial mining efforts. Eventually others invested in the idea, offering real world items in exchange for the virtual currency, and after some time there were investments in activities such as brokering firms who took the risk of converting real currencies into virtual bitcoins. The book covers the history of [8]Silk Road, the illegal drugs site, which uses the anonymous nature of bitcoin to allow the trade of items such as drugs. Popper's book is interesting - it

narrates the history in detail [which gets a little tiresome at times] and also tries to explain how a virtual thing can actually have a real world value. The initial miners' rewards for their mining efforts, including the blocks mined by the creator of bitcoin, are now worth considerable amounts of real world currency.

Obviously you need to know that your transactions are safe and there are loads of papers out there that analyse the safety of the currency. I enjoyed [9]this Msc thesis which used Google App Engine to do various experiments. For an idea of the representational tricks that mean you can use the block chain for recording ownership of things such as cars, [10]OpenAssets is one encoding you could take a look at.

Of course, you probably want to have a play to understand all of this. I started out trying to find where I could get some bitcoins. There are various faucets that occasionally give away free bitcoins, and many online bitcoin wallet management services, but I wasn't sure if I really wanted to sign up. Fortunately there is a test version of the bitcoin block chain that is used for testing the various clients - so you have the ability to do the transactions on a set of bitcoins that you can get for free from the Testnet faucet, though the coins have no actual value.

You're also going to need a library to program against bitcoin, and for this I selected [11]NBitcoin, whose author has a [12]free book on how to use it. It is available as a NuGet package so you can easily use it from Visual Studio.

First we need to get a private/public key pair as an identity, from which we can get an address that we can type into the [13]testnet faucet to get some bitcoins. [14]We generate a new key and serialize it to a file for use later. [15]We can then read the key back and generate a bitcoin address. We can type that into the faucet, and [16]we'll be notified of the pending transfer.

There are lots of websites that let you look up data from the block chain. I can use one to look at the [17]coins owned by my key and look at [18]the details of the transaction that populated it. We can see the process of block chain at work here. [19]The transaction is initially unconfirmed as it makes its way across the peer to peer network until a miner picks it up and it [20]becomes part of the block chain, though you need to wait until the block containing your transaction is several levels from the chain's head before you can be confident that it will remain [typically 6 levels with one level every 10 minutes on the real block chain]

The NBitcoin library is powerful. It contains a lot of utilities to work with various block chain information sources and contains extensions for dealing with asset management via coloured coins. [21]I quickly tested it out using some C # to transfer some bitcoins associated with key I had generated to another key. With bitcoin, one takes an output of a previous transaction, so I needed to fetch the transaction using the transaction id of the transfer from the blockr block chain information site. I split the output of the transaction that gave the money to me into three outputs: the majority gets transferred back to my account, a second small amount goes to the second account, and I add a little information to the transaction that becomes part of the block chain ledger and could be used to record asset transfer. Any remainng bitcoin is used as a fee for a miner, encouraging one of the miners to include the transaction in the block chain. [22]We can see the results on one of the chain exploration sites.

In the C # code I needed to access the peer network to get my transaction to the min-  
844

ers. You can get a bitcoin client from this [23]site. Running the bitcoin client for a while, bitcoind -testnet

generated a list of peers in the peers.dat file which I could then use with the NBitcoin library to push my transaction out to the world. Alternatively the library can use a running local node, but I didn't want to leave the node running so instead decided to use the peer data file. There's lots of documentation [24]here to discover how you can use the other utilities included in the download.

The block chain idea is fascinating - a distributed ledger with an immutable history, and there are many people trying to find uses for it. One example [25]is the notion of side chains, which manage content but include a pointer to this data from the block chain by using a hashed signature which is included in the real block chain. There's loads more experimenting to do, and I'm sure there are many interesting discoveries to come.

1. <https://en.wikipedia.org/wiki/Bitcoin>
2. [https://en.wikipedia.org/wiki/Block\\_chain\\_\(database\)](https://en.wikipedia.org/wiki/Block_chain_(database))
3. <https://www.coursera.org/course/bitcointech>
4. [https://d396qusza40orc.cloudfront.net/bitcointech/readings/princeton\\_bitcoin\\_book.pdf](https://d396qusza40orc.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf)
5. <https://bitcoin.org/bitcoin.pdf>
6. <http://www.amazon.co.uk/Digital-Gold-Bitcoin-Millionaires-Reinvent/dp/0062362496>
7. [https://en.wikipedia.org/wiki/Satoshi\\_Nakamoto](https://en.wikipedia.org/wiki/Satoshi_Nakamoto)
8. [https://en.wikipedia.org/wiki/Silk\\_Road\\_\(marketplace\)](https://en.wikipedia.org/wiki/Silk_Road_(marketplace))
9. <https://dl.dropboxusercontent.com/u/3658181/PiotrPiasecki-BitcoinMasterThesis.pdf>
10. <https://github.com/OpenAssets/open-assets-protocol/blob/master/specification.mediawiki>
11. <https://github.com/MetacoSA/NBitcoin>
12. <https://aois.blob.core.windows.net/public/Blockchain%20Programming%20in%20CSharp.pdf>
13. <https://tpfaucet.appspot.com/>
14. [https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6305&authkey=!AMdwRgH\\_y0uBXHc&v=3&iithint=photo%2c png](https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6305&authkey=!AMdwRgH_y0uBXHc&v=3&iithint=photo%2c png)
15. <https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6306&authkey=!AEEjfW1psee9IWQ&v=3&iithint=photo%2c png>
16. <https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6307&authkey=!AFR5ft0VTwMavxA&v=3&iithint=photo%2c png>
17. <https://www.biteeasy.com/testnet/addresses/mgN841vuDM7bMJdPuSji92noTt1XCy8usx>
18. <https://www.biteeasy.com/testnet/transactions/51fa006edfb26b4c29b32422e8147cea47448ac5c6dd6c098b7c94b222d9a435>
19. <https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6311&authkey=!ABsDoeLdRnJkAC8&iithint=folder%2c>
20. [https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6308&authkey=!A08ixnl4x8\\_yTw8&v=3&iithint=photo%2c png](https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6308&authkey=!A08ixnl4x8_yTw8&v=3&iithint=photo%2c png)
21. [https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6312&authkey=!AC2cxB\\_JsUsIAJQ&v=3&iithint=photo%2c png](https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6312&authkey=!AC2cxB_JsUsIAJQ&v=3&iithint=photo%2c png)
22. <https://onedrive.live.com/redir?resid=3F21DF299C355E7F!6313&authkey=!ACnow105NBFIaBc&v=3&iithint=photo%2c png>
23. <https://bitcoin.org/en/download>
24. <https://bitcoin.org/en/developer-examples#testing-applications>

### 10.1.5 Just how serious is a bug (2016-02-29 07:36)

I was thinking the other day about how hard it is to evaluate the impact of a bug fix. You have a bug report and determine the fix for it - just how do you then weigh the impact of the bug fix against the instability that this might cause if you release it? And I think that this is a very hard call.

I came up against this problem nearly a year ago. Microsoft were just about to release the [1]4.6 version of the .NET framework, and we were lucky enough to get a beta version to try. Several of us installed this beta onto our development machines, and continued working as normal. One of the testers in the team noticed that a PDF rendering component that we were using, and had been using for years, was no longer laying out the graph but was putting various elements in the wrong position. Mysteriously, it seemed to work on other people's machines, and also seemed to work when we ran the application inside Visual Studio, but not when we ran it inside Windbg. We also didn't see any failures if we built as x86, and so we spent a while checking whether previous builds of the product had accidentally been x86. It was only when I was doing the washing up that night, that I twigged that this is exactly what happens if you have a JIT bug. Running inside VS is going to turn off some of JIT optimisations, whereas running inside Windbg is going to leave these optimisations turned on.

The next morning I went in to work and verified this by setting the application config to use the legacy JIT, and sure enough the bug didn't happen. It was then a case of gathering more data and isolating the method that was giving the wrong result. This turned out to be a point where the JIT made an optimised tail call. I therefore reported this on [2]Connect. As is usual, I was then asked for a [3]self contained reproduction, which I supplied the next day. Time passed and then the issue was marked as fixed, with the fix being noted as available in a later release of 4.6

Around a month later .NET 4.6 was released to the world. And this [4]blog post came out. A x64 tail call was affecting ASP.NET code, and it turned out that this was [5]the same issue that I had reported.

The question is: how do you gauge the impact of a bug like this and decide whether the fix goes out straight away, or whether you test it a lot more and release it then? In [6]the advisory, Microsoft said that they had run lots of in house code and hadn't found a manifestation of the issue. However, that might not have been quite good enough, as people can take such a bug and try to convert it into an exploit - see the comment on the thread in the CoreClr issue where the return address can be obtained - so if you get unlucky and there's a widely available framework that allows an exploit to work, you probably do need to push out a fix. There is also the issue that the .NET framework is often used to run code written in C #, which probably doesn't have a lot of tail call optimisable method invocations, but it can also be used as runtime for languages like F # where the design patterns lead to lots more tail calls happening [as you recur down algebraic data types].

Pondering the issue, it seems to me that it is really hard to get enough data to inform your decision. What's the ratio between CLR runs of F # programs compared to C # programs? What's the percentage of C # programs that would hit this issue? And F # programs? What's the chance of the bad compilation being turned into an exploit?

In the end I suspect it just comes down to a call by a product manager who guesses the severity, and then uses feedback to determine if the call was right - obviously the more "important" the sender of the feedback, the more weight they are given. And that's a shame.

1. <https://blogs.msdn.microsoft.com/dotnet/2015/07/20/announcing-net-framework-4-6/>
  2. <http://connect.microsoft.com/>
  3. <https://connect.microsoft.com/VisualStudio/feedback/details/1372514>
  4. <http://nickcraver.com/blog/2015/07/27/why-you-should-wait-on-dotnet-46/>
  5. <https://github.com/dotnet/coreclr/issues/1296>
  6. <https://blogs.msdn.microsoft.com/dotnet/2015/07/28/ryujit-bug-advisory-in-the-net-framework-4-6/>
- 

## 10.2 March

### 10.2.1 Vagrant, up and running (2016-03-07 07:42)

I'd obviously heard of Vagrant a long time ago, but only used it in anger for the first time a few weeks ago [when playing with [1]MirageOS]. I decided that I need to understand a little more about how it works, so I bought the book [2]Vagrant: Up and Running by Michael Hashimoto.

The book is fairly short at only 138 pages, and is really a guide to the various commands that Vagrant offers, together with some example use cases. The introductory chapter discusses the need for Vagrant and desktop virtualisation. Chapter two walks us through the creation of a local Linux instance using Vagrant's init and up commands. Chapter three looks at provisioning, and the example here is generating an image with Apache serving a web site. Chapter four looks at networking, extending the example by having the web server talk to a database. Chapter five looks at multi-machine clusters, showing how easy it is to provisions a group of machines that emulate a production deployment. Chapter six talks about the concept of boxes.

Chapter seven of the book talks about extending Vagrant using plugins. This is the section of the book that I was looking forward to. The previous chapters covered the kind of things that you could do with Vagrant, and I was interested in how Vagrant actually does its stuff, but sadly this chapter doesn't really go into quite enough detail, rather concentrating on how you'd add new features to Vagrant rather than explaining the implementation.

Fortunately the source of Vagrant is available on GitHub, and it is a mass of Ruby code that is fairly easy to read. The default provider for Vagrant is the one for [3]VirtualBox and the code for this provisioner can be found [4]here. It turns out that the various Vagrant commands

[5]end up using the vboxmanage command line executable that is installed as part of the VirtualBox installation. Hence Vagrant is really an abstraction across a broad set of providers, allowing you to use them without having to worry about their specifics - a very clever and useful idea.

The book is a good, informative quick read and gives you an idea of what Vagrant can do for you. You can then dig into the implementation and details after reading it.

1. <https://mirage.io/>
  2. [http://www.amazon.co.uk/Vagrant-Up-Running-Mitchell-Hashimoto/dp/1449335837/ref=sr\\_1\\_1?ie=UTF8&qid=1456654382&sr=8-1&keywords=vagrant+up+and+running](http://www.amazon.co.uk/Vagrant-Up-Running-Mitchell-Hashimoto/dp/1449335837/ref=sr_1_1?ie=UTF8&qid=1456654382&sr=8-1&keywords=vagrant+up+and+running)
  3. <https://www.virtualbox.org/>
  4. <https://github.com/mitchellh/vagrant/tree/ca38d3673f7b7f494c4a621bb34fc89260cddcf/plugins/providers>
  5. [https://github.com/mitchellh/vagrant/blob/ca38d3673f7b7f494c4a621bb34fc89260cddcf/plugins/providers/virtualbox/driver/version\\_4\\_0.rb](https://github.com/mitchellh/vagrant/blob/ca38d3673f7b7f494c4a621bb34fc89260cddcf/plugins/providers/virtualbox/driver/version_4_0.rb)
- 

## 10.3 May

### 10.3.1 Switching to Angular 2 (2016-05-09 07:32)

[1]Switching to Angular 2 by Minko Gechev

In the past [2]I spent some time trying to get up to speed with Angular 1, and after some [3]announcements at Xmas time about the progress being made on Angular 2, I decided it was time to see how things have changed between the major releases. I ordered this book, but had to wait until April before it was published. In the meantime Angular 2 has moved to being very close to full release - [4]the recent ng-conf keynote explains this.

In short, the differences are very great... Angular 2 is very much aimed at being tooling friendly, giving it, as commented in [5]a recent podcast I listened to, more of a Visual Basic RAD feel with proper components each with a lifecycle, inputs and outputs [though I know of no tools so far that support it]. Moreover there is support for server side rendering of the initial page to avoid the delays as the page boots [so called Angular Universal], and also the possibility of the page using web workers for doing the computation.

Chapter 1 of the book is really good discussion of the current state of the web, and the lessons learned from experience with Angular 1, all of which have lead to the modified design of Angular 2. Some concepts from Angular 1, like the notion of scope have been removed, some important concepts like dependency injection have kept but made easier to use, and the framework has been redesigned to make it easier to support server-side rendering.

Chapter 2 takes us through the basic concepts of Angular 2. There's a big emphasis on building user interfaces via composition. Indeed in Angular 2, we have Directives which have no view and Components which do. Services and Dependency Injection still play a role, but

features such as change detection are much simpler and more user optimisable - detection can be turned off, customised in various ways, and the framework also knows about various persistent [functional] data types which make change detection much quicker. The whole digest cycle of Angular 1 is gone - [6]zones, which are explained [7]here, can be used to capture state transitions that may change the state of the user interface. Templates remain for Components to express their views, though filters have been replaced by a richer notion of Pipes.

Angular 2 aims to be a lot more declarative [think tool support again]. Though everything can be transpiled down to more primitive versions of JavaScript, there is an emphasis in the book on using [8]TypeScript which already supports some proposed extensions to JavaScript such as [9]decorators. Chapter 3 of the book takes us through the TypeScript language, with its classes, interfaces, lambda expressions and generics.

Chapter 4, Getting Started with Angular 2 Components and Directives, digs into the [10]building blocks of your Angular applications. We start with a basic hello world application, and the author explains how to get it running using code from his GitHub repository. We then move on to a ToDo application. This application emphasises breaking down the application into components that are connected together. For example, the input box for adding a ToDo, raises an event that is handled by another component in the GUI. The chapter covers projection and the ViewChildren and the ContentChildren decorators. This chapter also takes us through the [11]lifecycle of the component, describing the eight basic lifecycle events that the component can optionally handle, and then the tree style change detection that we now get from the framework - no more multiple change detection passes bounded with a maximal number of passes.

Chapter 5 goes through dependency injection in Angular 2. We now use an injector [which can have a parent and children] to control the scope of injectable classes, and we use decorators to declare the injected items. Again, this decorator syntax can be expressed using a lower level internal DSL if you do not want to use these higher level facilities.

Chapter 6 looks at [12]Angular Forms, which allow you to write the type of GUI you'll need for the CRUD parts of your application in a fairly declarative manner, explains the validator framework and how you can write your own validators. The second part of the chapter looks at routing.

Chapter 7 explains pipes and how you communicate with stateful services. In this part we have a quick look at async pipes and observables.

The last chapter looks at server side rendering and the developer experience. There are now command line tools for quickly starting a new Angular project.

I thought the book was a really good read. It seemed to cover the concepts quite concisely, and the examples made the concepts clear and understandable. It emphasised modern JavaScript and showed the flaws of the Angular 1 design. Now I need to put this all into practice writing some large application.

1. [https://www.amazon.co.uk/Switching-Angular-2-Minko-Gechev/dp/1785886207/ref=sr\\_1\\_1?ie=UTF8&qid=1462646898&sr=8-1&keywords=switching+to+angular+2](https://www.amazon.co.uk/Switching-Angular-2-Minko-Gechev/dp/1785886207/ref=sr_1_1?ie=UTF8&qid=1462646898&sr=8-1&keywords=switching+to+angular+2)

2. <https://clivetong.wordpress.com/2015/11/24/some-good-reads-on-angular-1-x/>
  3. <http://angularjs.blogspot.co.uk/2015/12/angular-2-beta.html>
  4. <https://www.youtube.com/watch?v=gdlpE9vPQFs&index=7&list=WL>
  5. <http://herdingcode.com/herding-code-216-bash-on-windows-angular-2-surface-book-kindle-oasis-windows-phone/>
  6. <https://github.com/domenic/zones>
  7. <http://blog.thoughtram.io/angular/2016/01/22/understanding-zones.html>
  8. <http://www.typescriptlang.org/>
  9. <https://github.com/wycats/javascript-decorators>
  10. <https://angular.io/docs/ts/latest/guide/architecture.html>
  11. <https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html>
  12. <https://www.youtube.com/watch?v=ihYc9y7dQA0&list=WL&index=21>
- 

## 10.4 November

### 10.4.1 At last you can prove your understanding of the C# specification (2016-11-14 07:01)

There was some discussion at work about the line of code:

```
IEnumerable<object> xx = new List<string>();
```

This line of code didn't work in the initial versions of C #, even those with generics, because of the need for variance in the IEnumerable interface. [Variance allows the compiler to relate the instantiations of IEnumerable<A> and IEnumerable<B> when the types A and B have a subtype relationship]

Of course, when you're discussing this kind of thing, it's important that you can talk about the parts of the [1]C # language specification that justify the steps the compiler is going to be taking. I believed that the conversion was because of an implicit reference conversion in the specification [6.1.6 Implicit Reference Conversion], but, of course, it's really hard to be certain that this is the rule the compiler is going to use and that there isn't some other rule which is actually being applied.

So what do we do?

I remembered reading how the Roslyn compiler had been written with an aim of keeping it very close to the C # specification, so that it was easier to verify that the implementation was correct. The Roslyn source is available [2]here on GitHub and it's fairly easy to [3]build the compiler if you have Visual Studio 2015 Update 3.

You can then write a simple source file containing the above code, and set the csc project as the startup project with the debug command line set to point to this file. The various conversions that annotate the semantic tree are listed in a [4]ConversionKind enumeration

and it is fairly easy to find uses of the ImplicitReference enumeration member to see where the annotation is added to the tree. This gave me a way to breakpoint and then look at the call stack to determine where I should set a breakpoint and start stepping. [This isn't always trivial because the call stack doesn't really tell you how you got to a certain point, but rather tells you where you are going to go when certain method calls finish. These concepts are sometimes different (for example in the case of tail calls)]

For our example code, the key point is that we find the implicit reference conversion used in the [5]ConversionsBase.cs file where we see a call to the method [6]HasAnyBaseInterfaceConversion with derivedType List<string> and baseType IEnumerable<object>. When we walk across the interfaces of the derivedType argument by calling the method [7]d.AllInterfacesWithDefinitionUseSiteDiagnostics, we'll enumerate across the type IEnumerable<string> and the compiler will check that it is variance converable to IEnumerable<object> in the call to [8]HasInterfaceVarianceConversion .

At this point the call stack looks like this:

```
ConversionsBase.HasAnyBaseInterfaceConversion
ConversionsBase.HasImplicitConversionToInterface
ConversionsBase.HasImplicitReferenceConversion
ConversionsBase.ClassifyStandardImplicitConversion
ConversionsBase.ClassifyImplicitBuiltInConversionSlow
ConversionsBase.ClassifyImplicitConversionFromExpression
ConversionsBase.ClassifyConversionFromExpression
Binder.GenerateConversionForAssignment
Binder.BindVariableDeclaration
Binder.BindVariableDeclaration
Binder.BindDeclarationStatementParts
```

What did I learn from this exercise?

There is now a C # implementation of the specification, so it is actually possible to check that you understand the parts of the specification that make some code valid. No longer do we guess what a C++ implementation of the compiler is doing, but we can animate the specification by stepping through the C # code. From the parts of the code that I have read, I'm not sure that I'd completely agree that the code follows the specification (making it easy to map from one to the other), but having an open source implementation does mean you can search for terms that you see in the specification to help you narrow down the search.

There are loads of other parts of the specification that I want to understand in more detail, so this is definitely an exercise that I am going to repeat in the future.

1. <https://www.microsoft.com/en-gb/download/details.aspx?id=7029>
2. <https://github.com/dotnet/roslyn>
3. <https://github.com/dotnet/roslyn/blob/master/docs/contributing/Building,%20Debugging,%20and%20Testing%20on%20Windows.md>
4. <https://github.com/dotnet/roslyn/blob/65cc61578e9646cf76a297d8a9e0005afa57378a/src/Compilers/CSharp/Portable/Binder/Semantics/Conversions/ConversionKind.cs>
5. <https://github.com/dotnet/roslyn/blob/master/src/Compilers/CSharp/Portable/Binder/Semantics/Conversions/Co>

nversionsBase.cs  
6. <https://github.com/dotnet/roslyn/blob/master/src/Compilers/CSharp/Portable/Binder/Semantics/Conversions/Co>  
nversionsBase.cs#L2215  
7. <https://github.com/dotnet/roslyn/blob/master/src/Compilers/CSharp/Portable/Binder/Semantics/Conversions/Co>  
nversionsBase.cs#L2230  
8. <https://github.com/dotnet/roslyn/blob/master/src/Compilers/CSharp/Portable/Binder/Semantics/Conversions/Co>  
nversionsBase.cs#L2232

---

# 2017

## 11.1 January

### 11.1.1 A dump has so many uses (2017-01-02 06:57)

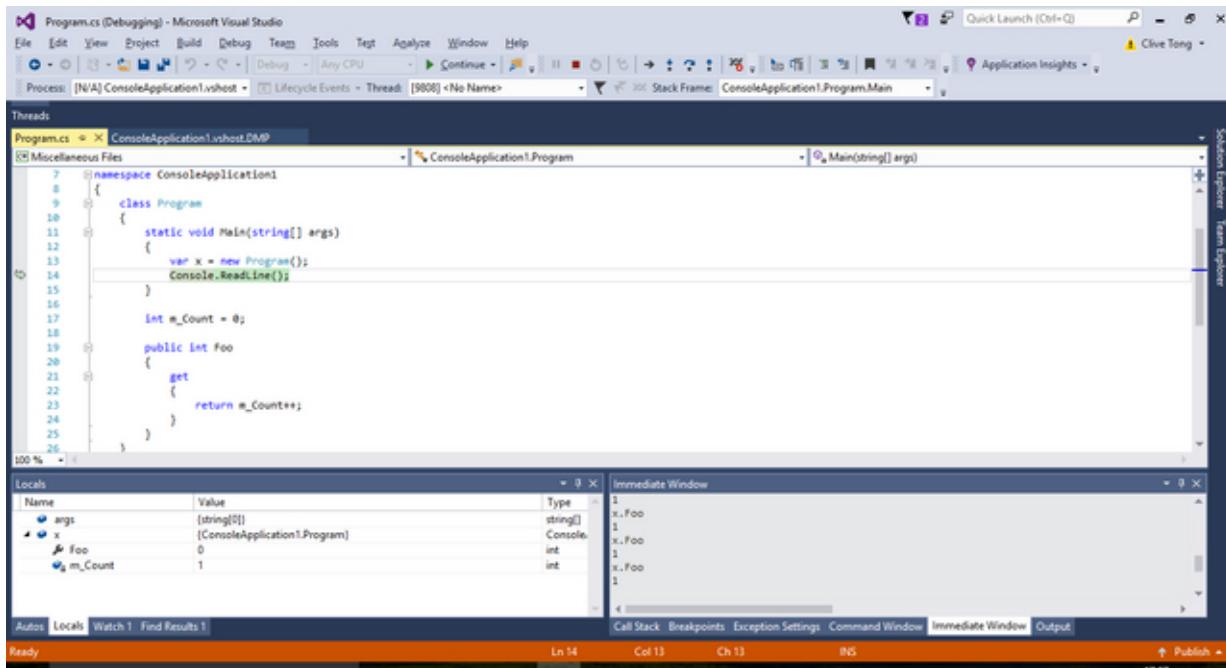
I'd been meaning to write a blog post about Windows dump files for a while now. I've used them at work a number of times. For example, when I worked on the CLR profilers, a dump file was a natural way to debug issues when there were problems with the COM component that we would inject into the profiled process. With the appropriate symbols, which you ought to have saved away to an internal symbol server, this was a good way of getting information on the issue.

I've used dump files in other scenarios too. When our cloud application seemed to be taking too much CPU when in production use, it was easy to take a dump of the process - the dump stalls the process only for a short amount of time so it is often possible to take a dump when an application is in production use. Better still, utilities like [1]procdump from Sysinternals allow you to take a dump of a managed process when interesting events happen, so we can take a dump when a first chance exception happens for example and capture the state of the process when the exception is thrown.

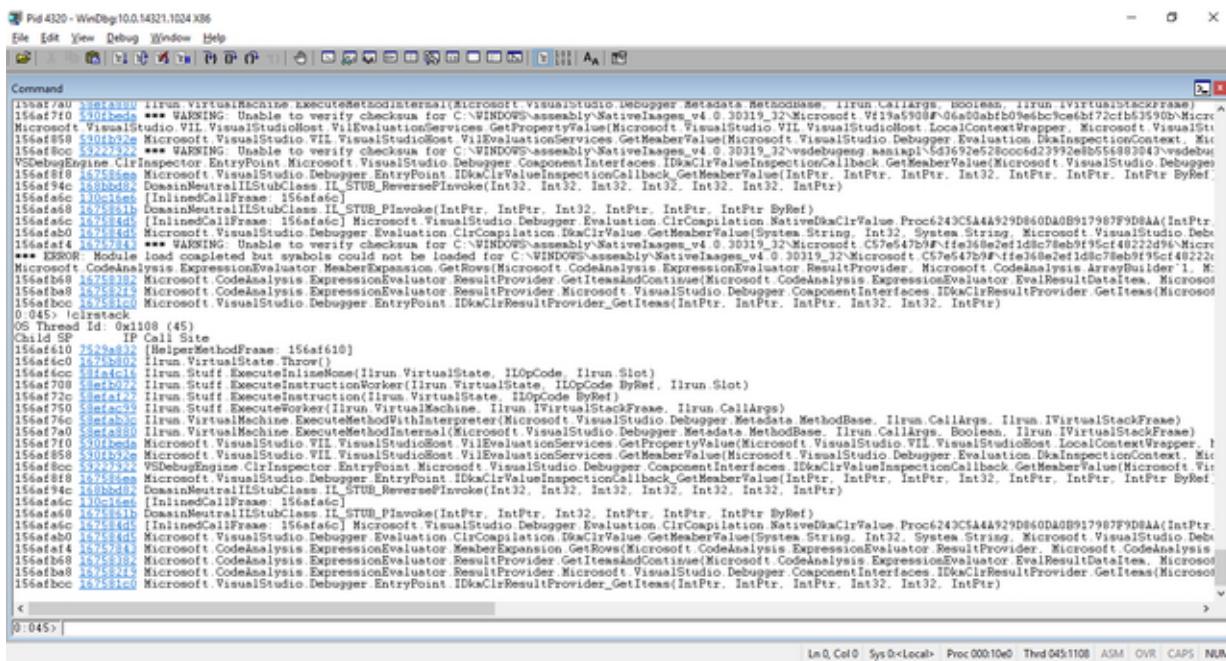
I've used Windbg to load and study such files in the past, though Visual Studio supports them pretty well these days. Moreover, Microsoft have recently released a managed .NET library, [2]ClrMD, which allows you to programmatically open and process such files, and the [3]source is now open

Anyway, I was reminded of all of this when I came across a recent talk by [4]Sasha Goldshtain, where he summarises the uses of dump files and shows some examples of the types of automated processing you could do using ClrMD. The talk also covers the many issues that dump files help you to debug (though you need to be slightly careful as the dump file may also contain sensitive data from the user's process).

The Visual Studio experience when you open a dump file is quite impressive. The normal debugging windows are all available, so you can see the threads and the locals of the various methods. One thing that has anyways impressed me is that the locals window manages to display the property values of the various objects.



I'd read in the past that this is possible because the VS code contains an IL interpreter, and wanted to prove this. I took the source code shown about and modified the property getter to throw an exception. Then, attaching WinDbg to the Visual Studio process and breaking on exceptions, I managed to stop at the following stack trace.



So indeed, Visual Studio contains code to allow the programming environment to simulate the execution of a property getter. That is rather cool!

Since I haven't blogged for a while, I'd also like to point out some interesting articles I've read over Xmas. On the C # front there's a good article on when you [5]should avoid using `async/await` and another on [6]the difference between `rethrow` and `rethrow ex;`

I've also been watching a load of [7]Virtual Machine Summer School and [8]Smalltalk videos, and never fail to be impressed with the work that Smalltalk did to get VM technology into the mainstream. There's this [9]interesting talk on the history of VMs and [10]advice from someone who has implemented many Java runtimes (and if you like the idea of a managed OS then [11]this talk on [12]COSMOS is also very interesting), plus a load of talks on [13]improving the VMs performance and the many [14]contributions of [15]Smalltalk.

1. <https://technet.microsoft.com/en-us/sysinternals/dd996900.aspx>
  2. <https://www.nuget.org/packages/Microsoft.Diagnostics.Runtime>
  3. <https://github.com/Microsoft/clrmd>
  4. <https://vimeo.com/191352972>
  5. <http://blog.stephencleary.com/2016/12/eliding-async-await.html>
  6. <https://blogs.msdn.microsoft.com/ansaxen/2016/12/09/c-exception-handling-a-case-study/>
  7. <https://www.youtube.com/playlist?list=PLJq3XDLIJkib2h2f0bomdFRZrQeJg4UIW&app=desktop>
  8. [https://www.youtube.com/channel/UCZ2hK\\_jLLa00BdpNk2Md2jw](https://www.youtube.com/channel/UCZ2hK_jLLa00BdpNk2Md2jw)
  9. <https://www.youtube.com/watch?v=QnQYhrpX39M>
  10. <https://www.youtube.com/watch?v=Hqw57GJSrac&list=PLJq3XDLIJkib2h2f0bomdFRZrQeJg4UIW&index=2>
  11. <https://www.youtube.com/watch?v=wgEBo-u19Wc>
  12. <https://github.com/CosmosOS/Cosmos>
  13. <https://www.youtube.com/watch?v=FF3EQg6fWvM>
  14. <https://www.youtube.com/watch?v=PHzNxVzIqGA>
  15. [https://www.youtube.com/watch?v=kz\\_YwEdu1IQ](https://www.youtube.com/watch?v=kz_YwEdu1IQ)
- 

### **11.1.2 'Tis the season for loads of Haskell (2017-01-04 07:20)**

The Xmas break is a good time to catch up with things. At the functional lunch at work we've been working though the [1]INCTA Haskell course exercises which take you through the implementation of many pieces of the Haskell libraries via a series of questions with associated tests. We've just done the parts on [2]Functor, [3]Applicative and [4]Monad. The questions were all really interesting, but I felt I needed a better introduction to the material. Luckily I came across the online version of [5]LearnYouAHaskell and the sections on [6]Functors, [7]Monads and [8]more Monads which are very good introduction to the relationship between the various concepts. After reading generally, useful Monads such as [9]the State Monad have numerous wiki pages devoted to them.

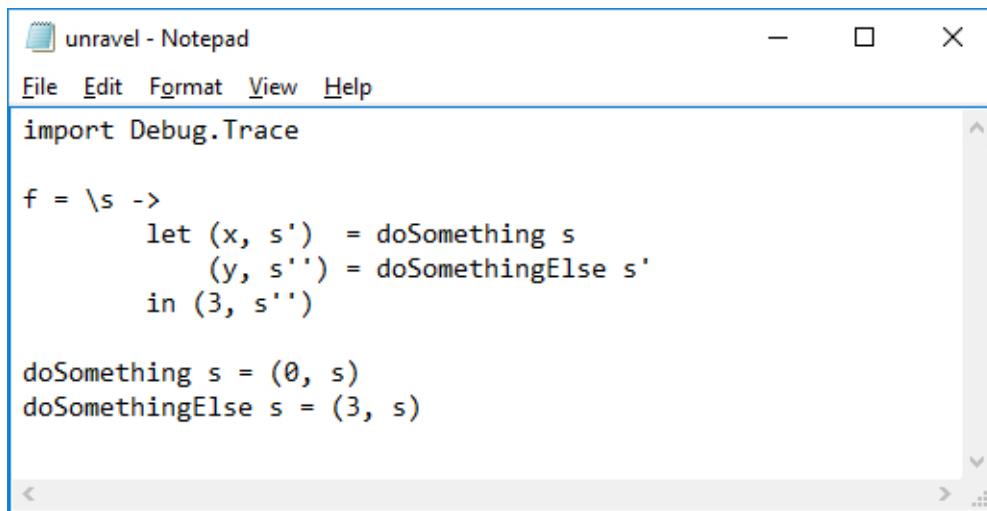
Of course in Haskell, Monads are not quite enough. When you write anything complicated, you end up needing to have multiple Monads (think IO and some other Monad such as the exception Monad), so it is also important to get a grip on Monad Transformers, which let you deal with a stack of Monads. There is a [10]reasonable introduction to these [11]on the Haskell wiki, and there is a list of the [12]standard transformers on the typeclassopedia.

Somehow the use of Monad Transformers still feels like magic, even when they are introduced one by one [13]like in this paper.

Of course, you might ask why Monads actually matter and [14]this blog post covers the four problems that Monads attempt to solve. Though I'm not sure I fully understood the [15]Kleisli category.

Once you want to play with Haskell, the obvious tool to use is [16]GHC. The really cool thing is that the [17]source to GHC is available on GitHub, so you can look at the actual implementation of all of the Monads described in the previous articles. Linked from the Haskell wiki page are a few blog articles such as the [18]24 days of GHC extensions, which includes articles on the more experimental features such as [19]Rank N types and [20]Bang Patterns.

I very much enjoyed having more time to play with GHCI, the interactive REPL which sits on top of the GHC compiler. There were a few commands that I hadn't noticed before, in particular [21]the single stepping which allows you to set breakpoints that are triggered when a particular piece of code is evaluated (and with Haskell being a lazy language the point of evaluation is often not very clear). This would have made life a bit easier in the past. For example, in [22]this blog post on the strictness of Monads, in the example the author uses trace to report when a thunk is evaluated. Using the stepper in GHCI, it is possible to get the debugger to show us this.



```
unravel - Notepad
File Edit Format View Help
import Debug.Trace

f = \s ->
    let (x, s') = doSomething s
        (y, s'') = doSomethingElse s'
    in (3, s'')

doSomething s = (0, s)
doSomethingElse s = (3, s)
```

By setting breakpoints, we can see the laziness, and in particular that doSomethingElse is evaluated before doSomething without needing to change the code and use the trace printing.

The screenshot shows a Windows command-line interface window titled "GHCi". The console output is as follows:

```

GHCi, version 8.0.1: http://www.haskell.org/ghc/ :? for help
Prelude> :load c:\users\clive\Desktop\unravel.hs
[1 of 1] Compiling Main           ( C:\users\clive\Desktop\unravel.hs, interpreted )
Ok, modules loaded: Main.
*Main> :break 4
Breakpoint 0 activated at C:\users\clive\Desktop\unravel.hs:4:24-36
*Main> :break 5
Breakpoint 1 activated at C:\users\clive\Desktop\unravel.hs:5:24-41
*Main> :break 6
Breakpoint 2 activated at C:\users\clive\Desktop\unravel.hs:6:12-19
*Main> :break 8
Breakpoint 3 activated at C:\users\clive\Desktop\unravel.hs:8:17-22
*Main> :break 9
Breakpoint 4 activated at C:\users\clive\Desktop\unravel.hs:9:21-26
*Main> print (f 2)
Stopped in Main.f, C:\users\clive\Desktop\unravel.hs:6:12-19
_result :: (Integer, t) = _
s'' :: t =
[C:\users\clive\Desktop\unravel.hs:6:12-19] *Main> :list
5      (y, s') = doSomethingElse s'
6      in (3, s')
       ^^^^^^
7
[C:\users\clive\Desktop\unravel.hs:6:12-19] *Main> :step
(3,Stopped in Main.f(...), C:\users\clive\Desktop\unravel.hs:5:24-41
_result :: (Integer, t) = _
s' :: t =
[C:\users\clive\Desktop\unravel.hs:5:24-41] *Main> :list
4      let (x, s') = doSomething s
5      (y, s') = doSomethingElse s'
       ^^^^^^^^^^^
6      in (3, s')
[C:\users\clive\Desktop\unravel.hs:5:24-41] *Main> :step
Stopped in Main.doSomethingElse, C:\users\clive\Desktop\unravel.hs:9:21-26
_result :: (t, t1) = _
s :: t1 =
[C:\users\clive\Desktop\unravel.hs:9:21-26] *Main> :list
8 doSomething s = (0, s)
9 doSomethingElse s = (3, s)
       ^^^^^^
10
[C:\users\clive\Desktop\unravel.hs:9:21-26] *Main> :step
Stopped in Main.f(...), C:\users\clive\Desktop\unravel.hs:4:24-36
_result :: (Integer, t)

```

Moreover, the debugger lets you see how much of the various data values have been evaluated as the printing in the debugger (using :print) does not force the evaluation. This makes it a really good way to understand the evaluation to [23]whnf. You can force the evaluation too using the :force command, though this obviously changes the behaviour of the program you are debugging.

The screenshot shows a continuation of the GHCi session from the previous image, demonstrating the use of :print and :force:

```

*Main> print (f 3)
Stopped in Main.f, C:\users\clive\Desktop\unravel.hs:6:12-19
_result :: (Integer, t) = _
s'' :: t =
[C:\users\clive\Desktop\unravel.hs:6:12-19] *Main> :step
(3,Stopped in Main.f(...), C:\users\clive\Desktop\unravel.hs:5:24-41
_result :: (Integer, t) = _
s' :: t =
[C:\users\clive\Desktop\unravel.hs:5:24-41] *Main> :print s'
s' = (_t1::t)
[C:\users\clive\Desktop\unravel.hs:5:24-41] *Main> :force s'
*** Ignoring breakpoint
*** Ignoring breakpoint
s' = 3
[C:\users\clive\Desktop\unravel.hs:5:24-41] *Main> :print s'
s' = 3

```

All considered I learned a lot about Haskell over the holiday. There are [24]many papers and book still to read but the language continues to bring a lot of novel concepts out into the open. And it is hard not to find it fascinating.

1. <https://github.com/NICTA/course/tree/master/src/Course>
  2. <https://github.com/NICTA/course/blob/master/src/Course/Functor.hs>
  3. <https://github.com/NICTA/course/blob/master/src/Course/Applicative.hs>
  4. <https://github.com/NICTA/course/blob/master/src/Course/Monad.hs>
  5. <http://learnyouahaskell.com/chapters>
  6. <http://learnyouahaskell.com/functors-applicative-functors-and-monoids>
  7. <http://learnyouahaskell.com/a-fistful-of-monads>
  8. <http://learnyouahaskell.com/for-a-few-monads-more>
  9. [https://wiki.haskell.org/State\\_Monad](https://wiki.haskell.org/State_Monad)
  10. [https://en.wikibooks.org/wiki/Haskell/Monad\\_transformers](https://en.wikibooks.org/wiki/Haskell/Monad_transformers)
  11. [https://wiki.haskell.org/All\\_About\\_Monads#Combining\\_monads\\_the\\_hard\\_way](https://wiki.haskell.org/All_About_Monads#Combining_monads_the_hard_way)
  12. [https://wiki.haskell.org>Typeclassopedia#Standard\\_monad\\_transformers](https://wiki.haskell.org>Typeclassopedia#Standard_monad_transformers)
  13. <http://catamorph.de/documents/Transformers.pdf>
  14. <https://cdsmith.wordpress.com/2012/04/18/why-do-monads-matter/>
  15. [https://en.wikipedia.org/wiki/Kleisli\\_category](https://en.wikipedia.org/wiki/Kleisli_category)
  16. <https://wiki.haskell.org/GHC>
  17. <https://github.com/ghc/ghc/tree/master/libraries>
  18. <https://ocharles.org.uk/blog/pages/2014-12-01-24-days-of-ghc-extensions.html>
  19. <https://ocharles.org.uk/blog/guest-posts/2014-12-18-rank-n-types.html>
  20. <https://ocharles.org.uk/blog/posts/2014-12-05-bang-patterns.html>
  21. [https://downloads.haskell.org/~ghc/7.6.2/docs/html/users\\_guide/ghci-debugger.html](https://downloads.haskell.org/~ghc/7.6.2/docs/html/users_guide/ghci-debugger.html)
  22. <http://blog.ezyang.com/2011/05/unraveling-the-mystery-of-the-io-monad/>
  23. [https://wiki.haskell.org/Weak\\_head\\_normal\\_form](https://wiki.haskell.org/Weak_head_normal_form)
  24. <http://www.cs.ox.ac.uk/jeremy.gibbons/publications/iterator.pdf>
- 

## 11.2 October

### 11.2.1 Some books since last time (2017-10-28 18:57)

For some reason I just haven't got around to blogging for a long while, but fortunately I have had time to read a fair number of computing related books which I thought I would type up here. There are a couple of management related books thrown in for good measure.

[1]Managing Humans by Michael Lopp

This is a collection of stories about the author's experiences managing development teams. A fun, humorous read, which made it even clearer that management is a lot about applying common sense to a range of activities.

[2]Troubleshooting with the Windows Sysinternals Tools by Mark Russinovich and Aaron Margosis

The [3]sysinternals tools are amazing, with specific tools offering information that would be hard to dig out yourself. This book takes you through the various tools one-by-one and tells you many of the lesser known features of the tool. The book contains a large section of case studies on how the tools were used to diagnose a wide range of problems.

[4]Smalltalk-80 Bits of history, words of advice by Glenn Krasner

I remember how much I enjoyed this book when I first read it 30 years ago. Lots of chapters written by different people on topics around the early Smalltalk systems, including details about implementations, porting efforts to get the standard image to run on diverse sets of hardware, and discussions of improvements to the system for the future.

[5]Understanding Computation by Tom Stuart

This book uses implementation in Ruby as a way of understanding computation from formal semantics to automata theory to Turing machines. And you get to learn some Ruby along the way. I really enjoyed this book. The writing is interesting, and you really don't ever understand something until you implement it.

[6]The One Device - the secret history of the iPhone by Brian Merchant

A very interesting read on the history of the iPhone. The book has various chapters on the various components, such as the battery and the screen, giving lots of interesting background about each area.

[7]Hit Refresh - The quest to Rediscover Microsoft's Soul and Imagine a Better Future For Everyone by Satya Nadella

An interesting read, part-autobiographical about Microsoft's CEO. We learn some details about Nadella's early years and the groups that he worked with when he joined Microsoft. There are chapters that talk about where Microsoft is going in the future - Mixed reality, artificial intelligence and quantum computing. I'm not sure I learned as much from the book as I had hoped, but worth a quick read.

[8]Debugging Applications for Microsoft .NET and Microsoft Windows by John Robbins

This book is now fairly old and I was lucky to find a copy for one pound in a charity shop. Some good advice on debugging, and some nice debugging war stories, from a renowned conference speaker. Some of the .NET related material is a little out of date, but there's still enough information to make this a fun read.

[9]Developing Windows NT Device Drivers by Edward Dekker and Joseph Newcomer

Again a fairly old book that you can pick up quite cheaply. Lots of insights into the world of device drivers, and along the way a lot of insights into the implementation of the windows operating system. A very good read.

## [10]Type-driven Development with Idris by Edwin Brady

There are more and more discussions on dependent types on the various news feeds that I subscribe to. Idris is a Haskell -like language designed for type driven development, a system where the dependent types can be used to specify the desired solution, and the system can attempt to do some primitive theorem proving to verify the dependent type constraints. The book is a good introduction to the idea and to the language itself.

1. <https://www.amazon.co.uk/Managing-Humans-Humorous-Software-Engineering/dp/1430243147>
  2. [https://www.amazon.co.uk/Troubleshooting-Windows-Sysinternals-Tools-Russinovich/dp/0735684448/ref=sr\\_1\\_fkmr0\\_1?s=books&ie=UTF8&qid=1509203128&sr=1-1-fkmr0&keywo](https://www.amazon.co.uk/Troubleshooting-Windows-Sysinternals-Tools-Russinovich/dp/0735684448/ref=sr_1_fkmr0_1?s=books&ie=UTF8&qid=1509203128&sr=1-1-fkmr0&keywo)
  3. <https://docs.microsoft.com/en-gb/sysinternals/>
  4. [https://www.amazon.co.uk/Smalltalk-80-Bits-History-Words-Advice/dp/0201116693/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1509203341&sr=1-1&keywords=smalltalk-80+bits+of+his](https://www.amazon.co.uk/Smalltalk-80-Bits-History-Words-Advice/dp/0201116693/ref=sr_1_1?s=books&ie=UTF8&qid=1509203341&sr=1-1&keywords=smalltalk-80+bits+of+his)
  5. [https://www.amazon.co.uk/Understanding-Computation-Machines-Impossible-Programs/dp/1449329276/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1509203513&sr=1-1&keywords=understa](https://www.amazon.co.uk/Understanding-Computation-Machines-Impossible-Programs/dp/1449329276/ref=sr_1_1?s=books&ie=UTF8&qid=1509203513&sr=1-1&keywords=understa)
  6. [https://www.amazon.co.uk/One-Device-Secret-History-iPhone/dp/0593078403/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1509203753&sr=1-1&keywords=the+one+device](https://www.amazon.co.uk/One-Device-Secret-History-iPhone/dp/0593078403/ref=sr_1_1?s=books&ie=UTF8&qid=1509203753&sr=1-1&keywords=the+one+device)
  7. [https://www.amazon.co.uk/Hit-Refresh-Rediscover-Microsofts-Everyone/dp/000824765X/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1509203890&sr=1-1&keywords=hit+refresh](https://www.amazon.co.uk/Hit-Refresh-Rediscover-Microsofts-Everyone/dp/000824765X/ref=sr_1_1?s=books&ie=UTF8&qid=1509203890&sr=1-1&keywords=hit+refresh)
  8. [https://www.amazon.co.uk/Debugging-Applications-Microsoft%C2%AE-Microsoft-Pro-Developer/dp/0735615365/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1509204188&sr=1-1&keywords="](https://www.amazon.co.uk/Debugging-Applications-Microsoft%C2%AE-Microsoft-Pro-Developer/dp/0735615365/ref=sr_1_1?s=books&ie=UTF8&qid=1509204188&sr=1-1&keywords=)
  9. [https://www.amazon.co.uk/Developing-Windows-Device-Drivers-Programmers/dp/0768682258/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1509204326&sr=1-1&keywords=developing+window](https://www.amazon.co.uk/Developing-Windows-Device-Drivers-Programmers/dp/0768682258/ref=sr_1_1?s=books&ie=UTF8&qid=1509204326&sr=1-1&keywords=developing+window)
  10. [https://www.amazon.co.uk>Type-driven-Development-Idris-Edwin-Brady/dp/1617293024/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1509205147&sr=1-1&keywords=type-driven+developme](https://www.amazon.co.uk>Type-driven-Development-Idris-Edwin-Brady/dp/1617293024/ref=sr_1_1?s=books&ie=UTF8&qid=1509205147&sr=1-1&keywords=type-driven+developme)
- 

## 11.3 November

### 11.3.1 Bitcoin - where does it go next? (2017-11-27 07:07)

[1]Attack of the 50 Foot Blockchain: Bitcoin, Blockchain, Ethereum & Smart Contracts by David Gerard

I did a fair bit of reading about BitCoin in the past (after my interest had been piqued by the Coursera course on the topic), and have spent some time following the various newsgroups and issues, but have been troubled about whether Bitcoin stands at chance of

succeeding in the real world.

This book is really good. It takes a massively anti-Bitcoin and associated technologies stance, and puts forward really good arguments about why Bitcoin is a massive fad. As usual the truth is probably somewhere in the middle, but the author's arguments about the troubles of scaling Bitcoin really make it seem useless - it can handle 7 transactions a second compared to Visa's 50000, and many vendors gave up with it because of lack of interest, never mind the fact that it takes so long to verify a transaction that it is an impractical method of buying things for many types of transaction.

The author also gives some examples of where smart contracts have turned out to be anything but smart. He points out that legal contracts all suffer from interpretation and the regular need for arbitration, and so any kind of contract whose meaning is defined by a segment of code is never going to work at the edges where the contract depends on inputs from the real world.

There is also a good set of arguments as to why private blockchains fail to hit the mark. The current Blockchain burns as much power of Ireland, with much of the proof of work being used to randomise the miner who controls the transaction. Once you move to a private blockchain, you regain the centralisation which was one of the main selling points of blockchains, so you might just as well go back to a database instead... indeed, lots of people do not want their transactions listed in detail on a public medium so any kind of global ledger is unlikely to gain traction.

In summary, a thought provoking, short read. Like many things, the technology is clever and it is really a question as to whether there is a place for it in the real world.

1. [https://www.amazon.co.uk/Attack-50-Foot-Blockchain-Contracts/dp/1974000060/ref=sr\\_1\\_1?ie=UTF8&qid=1511700797&sr=8-1&keywords=attack+of+the+blockchain](https://www.amazon.co.uk/Attack-50-Foot-Blockchain-Contracts/dp/1974000060/ref=sr_1_1?ie=UTF8&qid=1511700797&sr=8-1&keywords=attack+of+the+blockchain)

---

davidgerard (2018-03-01 15:48:28)  
Just saw this - thank you :-)

### **11.3.2 How do you nest your virtualization? (2017-11-29 07:07)**

I was looking through some of the YouTube talks from Ignite, when I came across [1]this interesting talk on nested virtualization in Hyper-V. Since September you have been able to provision virtual machines on Azure which support nested virtualization. This is obviously a very powerful feature, and enables many scenarios (such as testing) which you couldn't easily do before.

This made me start thinking about how you get nested virtualization to work on other platforms such as AWS. I'd come across virtualization using binary translation in the past (as that was the way that VMWare did its thing back in the day), and came across [2]this fairly recent paper that talks about the method. The resulting virtualization which can run in a cloud

environment is [3]covered in the paper.

That then leads to the question of whether software implementation can compare with the hardware assisted virtualization, and there are some papers such[4] as this one that study the problem. [5]Hardware support on Intel requires some extensions, the so called VT-X extensions, which are available on more modern processors and which make things a lot easier for the implementation.

1. <https://youtu.be/iRx0Iwl1JPw>
  2. <https://blogs.oracle.com/ravello/nested-virtualization-with-binary-translation>
  3. <https://www.usenix.org/system/files/conference/hotcloud13/hotcloud13-fishman.pdf>
  4. [https://www.vmware.com/pdf/asplos235\\_adams.pdf](https://www.vmware.com/pdf/asplos235_adams.pdf)
  5. <https://pdfs.semanticscholar.org/f6bf/459d260e89762dff68f65e0c14e149bf4da.pdf>
- 

## 11.4 December

### 11.4.1 Some what types of reflection are there? (2017-12-12 07:03)

I can't remember how it came up in the conversation, but I remembered the other day the famous paper by [1]Brian Smith that introduced the idea of the reflective tower of interpreters. The paper is a little hard to read these days, but does try to explain the notion of structural and behavioural reflection in programming languages.

In order to try to understand a little more deeply, I turned to two other Scheme related papers for help. [2]The first talks about reflection and this is extended[3] in the second paper to give a different semantics behind the reflective tower. There is a slightly more modern implementation of the tower in Common Lisp, and some good explanations [4]in this paper.

I also came across a number of related papers - [5]a general discussion of reflection, and a discussion of [6]how to allow compiled methods in this reflective tower world. A lot of this more recent work was also connected to [7]multi-stage programming and [8]partial evaluation

1. [https://www-master.ufr-info-p6.jussieu.fr/2006/Ajouts/Master\\_esj\\_2006\\_2007/IMG/pdf/Smit84.pdf](https://www-master.ufr-info-p6.jussieu.fr/2006/Ajouts/Master_esj_2006_2007/IMG/pdf/Smit84.pdf)
  2. <https://www.cs.indiana.edu/pub/techreports/TR161.pdf>
  3. <https://www.cs.indiana.edu/ftp/techreports/TR196.pdf>
  4. <http://www.p-cos.net/documents/s32008.pdf>
  5. <http://www2.parc.com/csl/groups/sda/projects/reflection96/docs/malenfant/malenfant.pdf>
  6. <http://lampwww.epfl.ch/~amin/doc/lms-black.pdf>
  7. <https://pdfs.semanticscholar.org/1726/703918e320dff60e013f76fa2a3bd22bc7b8.pdf>
  8. <http://www.itu.dk/people/sestoft/pebook/jonesgomardsestoft-a4.pdf>
-

# 2018

## 12.1 March

### 12.1.1 Some recent C# learnings (2018-03-06 08:40)

There are a couple of C # related things that I've come across recently, and also some interesting C # related talks from NDC which I'll detail below.

The first interesting thing I came across is that the names of your method parameters are semantically meaningful. I should probably have realised this in the past, but didn't until someone at work showed me how to stop an ambiguous reference.

The example was something like:

```
interface IA { }
interface IB { }
class C : IA, IB { }

static void Method(IA a) { } // set up some methods are equally good
static void Method(IB b) { }

static void Main()
{
    Method(new C());
}
```

The call to Method in Main cannot be resolved to either of the possible overloads. However, if you change it to

```
Method(a: new C());
```

or

```
Method(b: new C());
```

then the call is no longer ambiguous.

I'd obviously realised in the past that you had to be careful with optionals as they are inlined, potentially across assemblies, but I had never seen overload resolution changed by naming the parameters.

The second observation is to do with inlining in a Parallel.ForEach. I had some code that simplifies to roughly the following.

```
static void Main(string[] args)
{
    void Recurse(int x) =>
        Parallel.ForEach(new int[] { x }, Recurse);
    Recurse(1);
}
```

In the real code I'd added some code to try to check that we didn't end up calling a Parallel.ForEach in the context of another Parallel.ForEach using a ThreadStatic variable to record the context on the thread.

If you run the above code with a breakpoint on the Recurse method, you'll see it using the stack of where you first run it, due to the system trying to run the call inline [see TaskScheduler.TryRunInline]. However, rather than eventually getting a StackOverflowException, the computation is eventually moved to a different thread.

Poking around in Reflector you can find the class System.Threading.Tasks.StackGuard which has a method named [1]CheckForSufficientStack which uses Win32 methods to see how much stack space is left on the current thread. If there isn't enough stack space then we don't inline the recursive call, but instead move to another thread.

I've also seen some good C # talks: [2]C # 7.1 and 7.2 from NDC, [3]high performance C # code from experience working on Bing, [4]msbuild, which is a great introduction and covers the history of the transition to json project files and back,  
[5]what is .NET Standard?, [6]more C # 7 related performance changes  
[7]and there is also a discussion of the future of Rx and the associated Reactor project

1. <https://referencesource.microsoft.com/#mscorlib/system/threading/Tasks/Task.cs,7058>
2. <https://www.youtube.com/watch?v=IF1aPR6da3A&list=WL&t=0s&index=28>
3. <https://www.youtube.com/watch?v=fI1XGVIQjkA&list=WL&t=724s&index=27>
4. <https://www.youtube.com/watch?v=CiXlVrbBepM&list=WL&t=0s&index=10>
5. <https://www.youtube.com/watch?v=CmS6ta17VLk&list=WL&index=19>
6. <https://www.youtube.com/watch?v=CSPSvBeqJ9c&index=6&list=WL>
7. <https://github.com/Reactive-Extensions/Rx.NET/issues/466#issuecomment-370496523>

---

### **12.1.2 Not sure I love it, but I do understand it better (2018-03-14 06:47)**

There was a recent talk at NDC entitled [1]How to stop worrying and love msbuild by Daniel Plaisted. It was an interesting talk that discussed the history of the change to the csproj file to make it a lot smaller and tidier. The new format makes the project file look like the following, which is a massive contrast to the old multi-line mess.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>
</Project>
```

Of course, we have to ask where all of the existing XML has gone. The build system needs to get it from somewhere, and in the other parts of the talk the speaker discusses various debugging tools that give you a better idea of what is going on.

In the past I've always found it really hard to debug msbuild proj files. There seem to be masses of files that get imported and the trick has always seemed to be to pick a suitable target and then work from that. Using verbose logging you can track the actions of the build system as it does its work, and then work from that.

The first trick that the talk mentions is to use the pre-processor to get all of the content into a single text file.

```
msbuild /pp:out.txt
```

That works really well. Our small project file above expands to around 10000 lines of text, which is commented so that you can see the various imports and what those imports contain. It's really interesting looking through it to see what the build system defines.

To understand the dynamic side of things, there is a new binary logging format that the build system can dump. You can then[2] load this into a tool to search through the execution of the build.

```
msbuild /bl
```

The structured log viewer tool makes it really easy to find your way around the build. You can search for a text string, which makes it really easy to find tasks and properties, and there is a

timeline that tells you went and how long the various targets took to execute. It is fascinating to see how much work the system does before it actually calls the CSC task to compile a single file of C #.

I also notice that the documentation about msbuild has got better. A good starting point is the page that talks about the [3]difference between Items and Properties.

msbuild has always felt like the kind of technology I would love. I like programming systems with a small core language (so msbuild has items/properties and targets and tasks) and which then add abstractions on this small core. This kind of approach leaves you needing to understand only the small core language and allows you to explore into the abstractions that have been built up around it. For many Lisp and Smalltalk systems it is this exploration phase that it made easy by the tooling (the development environment) and this has always felt like the part that the msbuild ecosystem was missing. Maybe, these new tools are going to take this pain away at long last, though it still seems to be missing a way to actually single step through the execution of a build in some kind of build debugger which would allow the user to dynamically change things to see the effects. [Apparently you could at one point[4] debug the build scripts using Visual Studio though the debug flag doesn't seem to be available any more]

1. <https://www.youtube.com/watch?v=CiXlVrbBepM&list=WL&t=0s&index=10>
  2. <http://msbuildlog.com/>
  3. <https://docs.microsoft.com/en-gb/visualstudio/msbuild/comparing-properties-and-items>
  4. <https://blogs.msdn.microsoft.com/visualstudio/2010/07/06/debugging-msbuild-script-with-visual-studio/>
- 

## 12.2 April

### 12.2.1 C# fun with structs (2018-04-03 05:47)

I've just spent a while trying to catch up with all the changes to valuetypes in C # 7.x. As someone who doesn't use the struct keyword in C # very often at all, it is quite amazing all of the changes to C # around this area. [1]This talk is a very good summary.

The key point about structs is that they allow you to avoid some of the heap allocation that can be very bad for certain types of application. In the recent versions of C # the language has changed to allow the user to maintain a pointer to a struct, and hence use that rather than a copy of the original struct.

The classic example is something like the following (where we use a local function to make the example easier to read).

```
var data = new[] {1, 2, 3, 4, 5};  
ref int second = ref GetItem(data);  
second = 5;
```

```

ref int GetItem(int[] incoming)
{
    ref int x = ref incoming[2];
    return ref x;
}

```

In the example, the local variable, second, is an alias to the item in the data array, and so any change to either can be seen when viewing the data via the other item. The first thing one notices is the addition of a lot of "ref" keywords to make it clear that the local variable is holding an alias, and that the result of the method call is an alias to something else. It's a shame that there was a better syntax for this.

There are two other things that have happened in this area. The use of the "in" modifier and the ability to define a struct as readonly.

In the classic struct as a parameter case, the struct is copied on the way in to the method. Hence, for the example struct,

```

struct A
{
    public int _x;
    public void Increment() => _x++;
}

```

we get the behaviour.

```

A a = new A();
Increment(a);

void Increment(A x)
{
    x.Increment();
    x.Increment();
    Debug.Assert(x._x == 2);
}

Debug.Assert(a._x == 0);

```

We can avoid the copy using ref, but this then affects the caller.

```

A a = new A();
Increment(ref a);

```

```

void Increment(ref A x)
{
    x.Increment();
    x.Increment();
    Debug.Assert(x._x == 2);
}

Debug.Assert(a._x == 2);

```

We can pass the argument as an "in" parameter, which stops it being copied on entry to the method.

```

A a = new A();
Increment(in a);

void Increment(in A x)
{
    x.Increment();
    x.Increment();
    Debug.Assert(x._x == 0);
}

Debug.Assert(a._x == 0);

```

Now, of course, we have to answer the question: how is it that we don't see the parameter *x* being changed after the call to *x.Increment*. The answer is that the compiler takes a defensive copy when it makes the call. You can see this in the IL that is generated (and this is all covered really well by [2]this blog post).

In the above code, the IL for the invocation of *Increment* is

```

L_0000: nop
L_0001: ldarg.0
L_0002: ldobj ConsoleApp15.Program/A
L_0007: stloc.0
L_0008: ldloca.s a
L_000a: call instance void ConsoleApp15.Program/A::Increment()
L_000f: nop
L_0010: ldarg.0
L_0011: ldobj ConsoleApp15.Program/A
L_0016: stloc.0
L_0017: ldloca.s a
L_0019: call instance void ConsoleApp15.Program/A::Increment()
L_001e: nop

```

Changing the definition of A to

```
readonly struct A
{
    public readonly int _x;
    public void Increment() => Console.WriteLine(_x);
}
```

the compiler notices that the copy can be avoided and hence the IL changes to the more expected

```
L_0000: nop
L_0001: ldarg.0
L_0002: call instance void ConsoleApp15.Program/A::Increment()
```

It all goes to show that valuetypes are a bit confusing in C #, and it is really hard to know whether you need to optimise them. You really need to do performance measurements to tell whether extra copying is really going to make a difference to the performance of your application.

I got interested in the struct related changes after looking into the implementation of Span, which is implemented as a "ref struct". [3]Span is a powerful new abstraction over data types such as arrays, and allows you to slice them without copying, and without performance sapping view types. To implement such a thing, the view, the Span instance, needs to be stack allocated and guaranteed to have a dynamic scope that causes it to be de-allocated before the stack frame is unwound - this is a new idea for the CLR which has never really guaranteed that stack allocated things were different before.

You can play with Span using the pre-release System.Memory Nuget package.

```
var data = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var subdata = data.AsSpan().Slice(3, 4);
```

The subdata item is a Span, and looking at that in the debugger's raw view, shows you that it has the fields:

```
_byteOffset 0x00000010 System.IntPtr
_length 4 int
_pinnable {int[9]} System.Pinnable
```

This efficient implementation, means that we require the Span object is confined to the stack, which is all covered [4]in this proposal document. Span (and the heap allocated version, Memory) are likely to make their way into many framework classes in the coming releases

because of the amount that they can reduce allocation in cases where data is pulled from a buffer. The System.Memory package is .NET Standard and so is already available to run on a large number of platforms.

1. <https://www.youtube.com/watch?v=CDLvAF1jIPo&list=WL&index=27&t=0s>
  2. <https://blogs.microsoft.com/sepeplia/2018/03/07/the-in-modifier-and-the-readonly structs-in-c/>
  3. <https://msdn.microsoft.com/en-us/magazine/mt814808.aspx>
  4. <https://github.com/dotnet/csharplang/blob/master/proposals/csharp-7.2/span-safety.md>
- 

## 12.2.2 Dependent Types (2018-04-05 05:47)

I've been doing a fair amount of reading about dependent types recently, mainly because we have been working through the [1]Idris book as a [2]weekly lunchtime activity at work.

For a good article on the current state of play in Haskell, [3]Richard Eisenberg's thesis is very good - chapter 2 discusses a load of extensions of Haskell to enable type level programming. He also has [4]a good series of blog posts on various Dependent Types aspects. There are a number of interesting YouTube videos too, including this one of [5]using dependent types in a RegExp library.

At some level, the type checking becomes theorem proving. For the Idris language, there are [6]slides from a talk on how this process happens, and there's [7]another phd thesis on type checking Haskell which discusses some of the trade offs.

While we are talking about type checking, [8]there is a repository where the author is writing variants on Hindley-Milner type checking in F#. It's a very interesting learning resource, as the examples are small enough to step in the debugger, so you can watch what happens when a let bound function is generalised by the system.

There's always that interesting point between type checking and proof, and [9]this github repository uses [10]Idris to generate proofs about various Minesweeper boards.

1. <https://www.amazon.co.uk>Type-driven-Development-Idris-Edwin-Brady/dp/1617293024>
2. <https://github.com/red-gate/ClojureLunch/tree/master/idris>
3. <http://www.cis.upenn.edu/~sweirich/papers/eisenberg-thesis.pdf>
4. <https://typesandkinds.wordpress.com/2015/09/09/what-are-type-families/>
5. <https://www.youtube.com/watch?v=wNa3MMbhws4>
6. <https://eb.host.cs.st-andrews.ac.uk/writings/idris-stp.pdf>
7. <https://pdfs.semanticscholar.org/4a05/d76d1c582d0e1cab117542424941f9d43372.pdf>
8. <https://github.com/7sharp9/write-you-an-inference-in-fsharp>
9. <https://github.com/A1kmm/proofsweeper>
10. <https://www.idris-lang.org/>

---

### 12.2.3 Modern JavaScript for the win (2018-04-10 05:47)

[1]Practical Modern Javascript by Nicolas Bevacqua

I needed a book to get up to speed with ECMAScript 6, a book that would assume some knowledge of JavaScript and would take me through the new features. This book meets those requirements exactly. As we work though the sections, it takes its time to describe what problems the new features are trying to fix, and puts it all together in a really informative 300 page book.

The introduction discusses the various themes behind the added features, and then tells you how to configure Babel so that you can see how some of the new features can be transpiled down into code that will run on a large number of browsers. There's an online REP for Babel [2]here. ES6 has a large amount of new syntactic sugar, and it is great to be able to play around and see how such additions are converted into the old language - it really aids the understanding.

It was impressive to me how much JavaScript has moved forwards as a language. I remember going to a talk by Mark Miller many years ago [3]where he discussed using revocable proxies to enable mashups, and it is impressive to see how the proxy mechanism can be used for this kind of thing, as well as many other types of meta-programming. I also remember reading articles that described the various patterns to define object-oriented class like entities, and it's great to see that one standard pattern has now been put into the language.

Promises, generators and iterators (which should really be ordered the other way to describe how the features build on one another) bring really useful higher level programming to the language, and the inclusion of proper maps and sets give us a proper language at last. There's also a new module system to make programming in the large much easier, and also a number of new built-ins as well as additional methods on existing types.

What features do I think are really useful?

The extensions to the property syntax - method definitions, shorthands and computed property names

```
var x = "hello"
var y = { x, [x]:29, test(x) {return x+2}}
```

Argument destructuring and the spread operator.

```
var f = x => x + 1
var x = [1,2,3,4]
var y = [...x, ...x]
var [p, ...q] = x
```

Arrow functions and let and const together with temporal dead zone

```
const x = 20
let y = 40
let f = a => a * x + y
```

The class statement which expands to one of the old class patterns where you define a constructor function and set the relevant prototype.

```
class MyPoint {
  constructor(x,y) {
    this.x = x
    this.y = y
  }
  sum() {
    return this.x + this.y
  }
}
class NextPoint extends MyPoint {
  constructor(x,y,z) {
    super(x,y)
    this.z = z
  }
}
var p = new NextPoint(1,2,3)
```

ES6 adds a new extensibility mechanism, the Symbol, which allows you to add items to an object that won't be found by the standard `for..in`, `Object.keys` and `Object.getOwnPropertyNames`. This gives a way to add methods to make an object iterable, for example, without old code finding that it is being returned new keys in its iterations of an object's properties.

There are some new utility methods on `Object`, such as `assign` which allows you to take a default object and modify the properties,

```
Object.assign({}, defaults, options);
```

Also "is" which is a modified form of the `==` operator where

```
Object.is(NaN, NaN); // is true
Object.is(+0, -0); // is false
```

and Object.setPrototypeOf for setting the prototype (if you don't want to set it when using Object.create).

There is also the concept of decorators which allow you to add additional functionality to classes and statically defined properties. [4] This post offers a few examples.

The most interesting part of the book was the chapter on Promises, which also covers iterators and the implementation of async. The book works through the design of promises, starting with a description of callback hell and describing how promises help with this issue.

```
var reject, resolve
var p = new Promise((a,b) => { resolve = a; reject = b })
p.then(() => console.log("ok1"));
p.catch(() => console.log("fail")).then(() => console.log("ok2"));
reject(new Error("testing"))
```

It builds up nicely, with the author spending time talking about Promise.all and Promise.race, before moving on to the Iterable protocol.

```
const mySequence = {
  [Symbol.iterator]() {
    let i = 0
    return { next() { i++; var value = i; var done = i > 5; return
{value,done} } } }
}

for (var x of mySequence) { console.log(x); }
```

The book that talks about generators,

```
function* test() {
  yield 1;
  yield 2;
}

for (var x of test()) { console.log(x); }
```

and then async/await

```
var resolve;

var p = new Promise(r => resolve=r)

async function play() {
```

```

        console.log("start")
        await p
        console.log("end")
    }

play() // prints "start"
resolve(20) // prints "end"

```

The author spends time describing the transformation that happens when translating the `async` function into a generator, and how the steps of the generator are forced.

Chapter 5 looks at the new `Map`/`WeakMap`/`Set`/`WeakSet` datatypes and this is followed by a chapter on the new proxies, which allow controlled access to objects. This part of the language allows you to provide objects to foreign libraries, without having to give that foreign library full control of the object. There is a lot of discussion about the various traps that can be set up and handled, and how you can use the `Reflect` object to carry on with the default action after intercepting the call.

Chapter 7 of the book discusses a whole load of built-in improvements in ES6, including much better handling of Unicode characters that aren't on the BMP, and this is followed by a great chapter on the new module system.

In the last, short chapter of the book, the author discusses the language changes, and which parts of the language he considers to be important.

All considered this is a very good book, showing the numerous changes to the language and discussing the reasons behind the changes.

1. <https://www.amazon.co.uk/Practical-Modern-JavaScript-Nicolas-Bevacqua/dp/149194353X>
  2. <https://babeljs.io/repl/>
  3. <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36574.pdf>
  4. <https://www.sitepoint.com/javascript-decorators-what-they-are/>
- 

#### **12.2.4 Progressive Web Applications (2018-04-16 05:57)**

[1]Building Progressive Web Apps: Bringing The Power of Native to the Web by Tal Ater

I was interested in reading this book because I'd been hearing a lot about [2]service workers and wanted to understand what they were about. The whole Progressive Web Application has been something that Google has been pushing, and it has taken a while for other browser vendors like Microsoft and Apple to declare their support. However, with Microsoft's recent inclusion of service workers in the latest developer release of Edge, this seems to be

a technology that is going to be big - there will be talks on PWAs at //BUILD, and [3]there are rumours that PWAs may replace UWP as one of the implementation mechanisms behind applications in the Windows Store.

The book is a really good introduction to PWAs.

The author's [4]github repository contains a lot of related material, but for the book he has chapter oriented versions of a web application, [5]Gotham Imperial Hotel, which he converts into a progressive web application (see the branches corresponding to the various chapters). This is a really good learning mechanism, and it is nice to be able to try and step through the code to see what is happening. Google Chrome, for example, lets you fake internet disconnection so that you can try out the service worker code that handles the offline state.

After a brief introduction to service workers, the first chapter of the book takes us through service workers as a proxy and how they can intercept HTTP requests and hence act as a caching mechanism. Most importantly, the service worker still runs even if the browser is offline, and given that the service can determine that it is running offline, it can return different HTML and hence give the user an experience even in this case.

The next chapter deals with the installation of a service worker. Service workers are queued for installation, and are only actually installed when there are no tabs in the browser visiting the target site - it would obviously be confusing if different versions of a service worker were running in the different tabs targeting a particular site. The chapter explains the lifecycle, which includes a state where the service worker can cache resources without it being expected to handle any request interception. If the worker can't get the data it needs, it can set its state to say that it shouldn't be used, and we should continue with the existing worker. All of this state transition code is asynchronous, and will use Promises to avoid blocking.

The next chapter demonstrates how service workers let us write offline first applications. In this model we are really using PWAs as a deployment model. The chapter covers various patterns of caching - for example, you might try to pre-cache pages, or might instead cache on first access, but then you might also check for updates when the cache is hit and refresh the cache with the new version of the page. The author talks through the various patterns.

The next chapter looks at IndexDB. This is a local database, implemented in the browser, that allows you to store JSON objects, and offers a cursor to walk through the various tables and indexes. There are various patterns you may use to update the database between versions of the service worker.

The service worker isn't just a proxy for intercepting web requests. The next chapter covers background sync. If our application has been working offline, when we next go back online we'd like to resync with the application's web site. The service worker is able to receive callbacks from a sync manager, implemented in the browser, which can take care of tracking the synchronisation steps that need to be carried out and maintain the list of pending synchronisations. This service will take care of starting up our service worker if it isn't running, and will do so at a time where we are online.

The service is acting as a proxy between a site and the many possible tabs in the browser talking to that site. You might therefore expect there to be a mechanism for the various tabs to talk to each other (so that they can coordinate between themselves over state), and this

is what the next chapter covers. Messages can be posted, and message handlers can be registered.

Installation of a PWA, to give it a more permanent presence in the browser, is covered in the next chapter. Many web sites these days, nag you to install the mobile application version of their web site via interstitials, and this chapter talks about the heuristic driven way a browser allows you to install a PWA. In Chrome, for example, there are a number of things that trigger installation - regular use of the site for example.

Once your PWA is installed as an application, you might need a way for the site to notify you of interesting things that have happened. The next chapter covers the two different push notification mechanisms that you can use. There appears to be more standardisation work required in this area,

The book finishes with a couple of chapters of UX for Progressive Web Applications and where PWAs are going in the future.

The author has done a good job of breaking down the material into the various chapter sized chunks, and having a github application that can show you the changes related to each chapter is a really good learning mechanism. PWAs seem to have many uses, from pseudo-applications that are easy to install, to a richer way to allow web applications to run offline.

1. [https://www.amazon.co.uk/Building-Progressive-Web-Apps-Bringing-ebook/dp/B075HP52WY/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1523689763&sr=1-1&keywords=building+progressive](https://www.amazon.co.uk/Building-Progressive-Web-Apps-Bringing-ebook/dp/B075HP52WY/ref=sr_1_1?s=books&ie=UTF8&qid=1523689763&sr=1-1&keywords=building+progressive)
  2. <https://developers.google.com/web/fundamentals/primers/service-workers/>
  3. [https://www.zdnet.com/google-amp/article/microsoft-tees-up-windows-10-support-of-progressive-web-apps/?\\_twitter\\_impression=true](https://www.zdnet.com/google-amp/article/microsoft-tees-up-windows-10-support-of-progressive-web-apps/?_twitter_impression=true)
  4. <https://github.com/TAlAter>
  5. [https://github.com/TalAter/gotham\\_imperial\\_hotel](https://github.com/TalAter/gotham_imperial_hotel)
- 

### 12.2.5 Stack allocated closures make it into C# (2018-04-17 05:47)

Back in the days when I worked on Lisp compilers, I remember adding stack allocation of the closure records when calling local functions. In C # 7, we now have local functions and it is interesting to look at the optimisations that are applied for these.

Just for a base line let's have a quick look at the implementation of lambda expressions which close over variables the current method. The implementation, which has been around for a long time, re-homes the locals into a heap allocated (Display) class. This extends the lifetime of the variables allowing the reference from the lambda expression to govern their lifetime.

```
static Func<int,int,int> Check(int a, int b)
```

```

{
    return (x, y) => (x + y + a + b);
}

```

This is converted into code that has the following form. "a" and "b" have been re-homed into the heap allocated instance.

```

private static Func<int, int, int> Check(int a, int b)
{
    <>c__DisplayClass1_0 class_ = new <>c__DisplayClass1_0();
    class_.a = a;
    class_.b = b;
    return new Func<int, int, int>(class_.b__0);
}

```

The DisplayClass has the following definition, where we see the fields corresponding to the captured variable, the definition of the lambda method is encoded into this class too.

```

[CompilerGenerated]
private sealed class <>c__DisplayClass1_0
{
    public int a;
    public int b;

    internal int b__0(int x, int y)
    {
        return (((x + y) + this.a) + this.b);
    }
}

```

Local functions take us to code that has the following form.

```

static Func<int, int, int> Check2(int a, int b)
{
    return Local;

    int Local(int x, int y)
    {
        return (x + y + a + b);
    }
}

```

This is code generated slightly differently,

```

private static Func<int, int, int> Check2(int a, int b)
{
    <>c__DisplayClass2_0 class_ = new <>c__DisplayClass2_0();
    class_.a = a;
    class_.b = b;
    return new Func<int, int, int>(class_.g__Local|0);
}

```

We have the same style of `DisplayClass`, with the body of the local added as a method (as expected).

```

[CompilerGenerated]
private sealed class <>c__DisplayClass2_0
{
    public int a;
    public int b;

    internal int g__Local|0(int x, int y)
    {
        return (((x + y) + this.a) + this.b);
    }
}

```

However, there are now more optimisation possibilities. First, if the local function is scoped to the method in which it is defined, then it would be good to avoid the heap allocation.

```

static int Check3(int a, int b)
{
    return Local(1,2) + Local(3,4);

    int Local(int x, int y)
    {
        return (x + y + a + b);
    }
}

```

This is indeed what happens.

```

private static int Check3(int a, int b)
{
    <>c__DisplayClass3_0 class_;
    class_.a = a;
    class_.b = b;
    return (g__Local|3_0(1, 2, ref class_) + g__Local|3_0(3, 4, ref class_));
}

```

```
}
```

The DisplayClass has been optimised to a struct

```
[CompilerGenerated]
private struct <>c__DisplayClass3_0
{
    public int a;
    public int b;
}
```

and the body has been added as a method into the class in which contains the method containing the local

```
[CompilerGenerated]
internal static int g__Local|3_0(int x, int y, ref <>c__DisplayClass3_0
class_Ref1)
{
    return (((x + y) + class_Ref1.a) + class_Ref1.b);
}
```

The compiler has essentially noticed that the local method cannot escape from the method that uses it, and hence we can try to avoid the heap allocation.

We should also quickly look at the case where the local method doesn't capture any locals.

```
static int Check4(int a, int b)
{
    return Local(1, 2) + Local(3, 4);

    int Local(int x, int y)
    {
        return (x + y);
    }
}
```

In this case, the method compiles to the following

```
private static int Check4(int a, int b)
{
    return (g__Local|4_0(1, 2) + g__Local|4_0(3, 4));
}
```

and the local method is simply defined as a static method int he defining class

```
[CompilerGenerated]
internal static int g__Local|4_0(int x, int y)
{
    return (x + y);
}
```

While we are here we could quickly cover one memory management gotcha around closures and their implementation.

```
static (Func<int,int,int>, Func<int,int,int>) Check(int a, int b)
{
    return ((x, y) => (x + y + a), (x, y) => (x + y + b));
}
```

The implementation decides to put the local variables into a single DisplayClass

```
private static ValueTuple<Func<int, int, int>, Func<int, int, int>> Check(int a, int b)
{
    <>c__DisplayClass1_0 class_ = new <>c__DisplayClass1_0();
    class_.a = a;
    class_.b = b;
    return new ValueTuple<Func<int, int, int>, Func<int, int, int>>(class_.b_0), new Func<int, int, int>(class_.b_1));
}
```

This means that if either of the returned lambda expressions is alive (from the point of view of the GC), then the variables "a" and "b" are still alive. This might not seem to matter too much, but if "a" and "b" were large objects (for example), it does mean that their lifetime can be extended further than you might expect.

---

## 12.2.6 Reactive extensions in action (2018-04-18 05:06)

[1]Rx.NET In Action by Tamir Dresher

The reactive extensions have been around for a long time. [2]I remember coming across them in C # something like a decade ago, but I don't think I've seen a book or documentation that covers the whole of the implementation - sure, people spend a lot of time talking about the various combinators and hot and cold observables, but they don't spend much time talking about schedulers and the threading model that sits below the system.

Part one of the book, which consists of three chapters, gives a basic introduction to Reactive programming, and also covers some of the C # you need to make use of the Rx libraries.

By way of some examples, the first chapter introduces us to the idea of making events a first class object, the IObservable interface and its duality to the IEnumerable, and points out the differences between the push and pull models of event delivery. The author goes on to look at the properties described in the [3]Reactive Manifesto. We are also introduced to marble diagrams, which allow us to visualise the various interactions.

Chapter two takes us through a "Hello, Rx" application. This time it isn't Google suggest, which used to be the canonical example that is used in various write ups. In this book we look at a stock tracker application. This allows the author to cover how standard classes of .NET events can be converted easily into event streams, and the author gets a chance to talk a little about the threading concerns. I think that's great, as threading is often hidden under the covers in tutorials, but as soon as you want the events to be processed by a GUI you get into the GUI library's threading requirements.

Chapter three covers functional thinking in C #. Rx.NET encourages you to handle a pipeline for processing, with events feeding into the top of the pipeline, various filtering and processing happening in the middle, and then elements subscribing to the resulting output of the pipeline. This is a mechanism that the functional style handles very well.

The second part of the book has chapters on various Rx.NET concepts.

Chapter four starts with creating observables, which is demonstrated by writing an observer that logs the received events to the console (and we'll use this observer through the book). Of course writing things yourself gives you a chance to break the protocol of the IObservable - in particular the protocol that the messages flow as

```
(OnNext) * (OnError | OnCompleted)
```

It is therefore often better to use the Rx library's helpers for defining your own classes, so the author points to the ObservableBase class which makes it easy for you to define your own named types, or better still there are many overloads on Observable.Create to avoid the need to name a new type.

```
var ob = Observable.Create(observer =>
{
    Console.WriteLine("Started");
    Task.Run(() => { Task.Delay(TimeSpan.FromSeconds(2));
    observer.OnNext(2); observer.OnCompleted(); });
});
```

```

        return () => { Console.WriteLine("Finished"); };
    });

ob.Subscribe(x => Console.WriteLine(x));

```

This chapter also looks at converting the various .NET event styles to observables, and looks at converting from Enumerable to Observable and back again. We also see some of the more primitive observables that handle looping and single values.

```

var evensBelow50 = Observable.Generate(0, x => x  state + 2, v =>
v);
var singleValue = Observable.Return(10);
var neverFinish = Observable.Never();
var empty = Observable.Empty();
var _ = Observable.Throw(new Exception("Bang"));

```

Chapter five covers how you make observables from asynchronous code. It starts with looking at async friendly versions of Observable.Create

```

var ob = Observable.Create(async (observer, ct) =>
{
    Console.WriteLine("Started");
    ct.Register(() => Console.WriteLine("Finished"));
    await Task.Delay(TimeSpan.FromSeconds(2));
    ct.ThrowIfCancellationRequested();
    observer.OnNext(2);
    ct.ThrowIfCancellationRequested();
    observer.OnCompleted();
});

```

And then looks at the conversions between Task and Observable handled by the ToObservable method, and how SelectMany and Concat can be used to link different computations together.

Chapter six looks at the observer/observable relationship, in particular how to delay and re-subscribe to the observable. We walk through the DelaySubscription method and various other operators like SkipWhile and TakeUntil.

```
var ob = Observable.Range(1, 5).Do(x => Console.WriteLine(x));
```

Some of the ideas are put together in a drawing application where the code tracks the mouse and the mouse button up and down lead to event streams starting and ending.

Chapter seven looks at controlling the temperature of observables. Observables can be

categorised as hot or cold. Here cold means that the observable replays a set of event to each subscriber, whereas a hot observable only plays new events. In the case of the hot observable, if you weren't subscribed when the event happened, then you don't get to see it.

We start with an `ISubject`. Instances of this interface can act as an observer and as an observable, and Rx provides four types that implement this interface - `Subject`, `AsyncSubject`, `ReplaySubject` and `BehaviorSubject`. The book covers what all of these subjects do, and how they can be used to proxy hot and cold observables to give you something with various interesting behaviours.

Chapters eight and nine go through the many operators, from `Max` and `Count` all the way through to operators for partitioning an incoming event stream into a set of windowed buffers.

Chapter ten talks about concurrency and synchronisation, and is the best explanation I have read of this side of the Rx world. There are many types of `IScheduler` that are implemented by the library, ranging from a scheduler that uses threads from the thread pool to schedulers that hijack the current thread and don't return until a series of actions have finished.

The last chapter talks about error handling and recovery, and it also touches on the subject of backpressure. It is also very good and informative.

It is worth also mentioning that the book has three appendixes - some general coverage of asynchronous programming in .NET, a section on the Disposables that the Rx library offers and a section on testing Rx which talks about how you might Unit Test your code and use test schedulers to control the execution.

It was really good to have a single place that covered all of this material. Typically you can find some of this in blog posts spread all over the internet, but having it in a consistent story that develops over eleven chapters is brilliant.

I also noticed that the pre-release `System.Reactive` Nuget package contains code around the [4]`IQbservable` interface. It will be interesting to see where that goes in the future.

1. [https://www.amazon.co.uk/Reactive-Extensions-NET-examples-C/dp/1617293067/ref=sr\\_1\\_fkmr0\\_1?ie=UTF8&qid=1523780034&sr=8-1-fkmr0&keywords=rx.net+in+action](https://www.amazon.co.uk/Reactive-Extensions-NET-examples-C/dp/1617293067/ref=sr_1_fkmr0_1?ie=UTF8&qid=1523780034&sr=8-1-fkmr0&keywords=rx.net+in+action)
  2. <https://clivetong.wordpress.com/2010/07/12/calm-%E2%80%93-don%E2%80%99t-react/>
  3. <https://www.reactivemanifesto.org/>
  4. <https://clivetong.wordpress.com/2014/05/07/hello-source-meet-filter/>
- 

### 12.2.7 Let's get started with Docker (2018-04-19 05:14)

[1]Essential Docker for ASP.NET Core MVC by Adam Freeman

We are allowed to spent time at work on a Friday afternoon exploring new technologies,

so a colleague and I decided to work through this book. Microsoft have recently started supporting Docker running on Windows, and I thought this would be an interesting way to see how well the Windows Docker eco-system has been progressing. Also, this book targets ASP.NET 1.1 and I wanted to see if things were easier with the latest 2.1 version of ASP.NET.

The first two chapters in the book are a really brief introduction to Docker, followed by a list of the docker utility's commands.

Installing Docker on windows was really easy, requiring us to run an installer. We did have to turn on Hyper-V for Docker to use. This clashed with the Oracle VirtualBox that we typically use for testing, but fortunately I had a spare machine on which I could leave it turned on.

In chapter four of the book you write a fairly simple ASP.NET Core application which you then publish.

```
dotnet publish --framework netcoreapp2.0 --configuration Release --output dist
```

This application is then copied across to a docker container as part of the DockerFile

```
FROM microsoft/aspnetcore:2.0.3
COPY dist /app
WORKDIR /app
EXPOSE 80/tcp
ENV ASPNETCORE_URLS http://+:80
ENTRYPOINT ["dotnet", "dockerplay.dll"]
```

which we can then use to build a Docker container.

```
docker build . -t apress/exampleapp -f Dockerfile
```

The next chapter of the book deals with Volumes and Software Defined Networking. Volumes allow you to define some storage which can be attached to a container - this allows the container to run an application that writes to the file system to store its state, say a database. When we need to rebuild the container we can then re-attach the file system to the new container, and hence not lose any data.

This is where we diverged a little from the book. The book aims at Linux and mySQL, where we wanted to use SQL Server running on windows.

For this we pulled a pre-build image containing SQL Server.

```
docker pull microsoft/mssql-server-windows-express
```

And then used a volume to store the state.

```
docker volume create --name testdata
```

```
docker run -d -p 7002:1433 -e sa_password=ffddfdfdfdfd -e ACCEPT_EULA=Y -v testdata:c:\data microsoft/mssql-server-windows-express
```

The book moves on to SDN and the demo application uses two different network segments - one for the frontend and one for the backend. In the book, a proxy is used to load balance across the three servers that are set up.

Unfortunately there was no haproxy that would run in a Windows container, so we decided to use NGINX. Again, we had to build our own container for this, and I couldn't build on nano server (because my windows drive had become corrupted)

```
FROM microsoft/windowsservercore
ENV VERSION 1.13.9

SHELL ["powershell", "-command"]
RUN Invoke-WebRequest -Uri http://nginx.org/download/nginx-1.13.9.zip -OutFile c:\nginx-$ENV:VERSION-win64.zip; \
    Expand-Archive -Path C:\nginx-$ENV:VERSION-win64.zip -DestinationPath C:\ -Force; \
    Remove-Item -Path c:\nginx-$ENV:VERSION-win64.zip -Confirm:$False; \
    Rename-Item -Path c:\nginx-$ENV:VERSION -NewName nginx

# Make sure that Docker always uses default DNS servers which hosted by Dockerd.exe
RUN Set-ItemProperty -Path
'HKLM:\SYSTEM\CurrentControlSet\Services\Dnscache\Parameters' -Name
ServerPriorityTimeLimit -Value 0 -Type DWord; \
    Set-ItemProperty -Path
'HKLM:\SYSTEM\CurrentControlSet\Services\Dnscache\Parameters' -Name
ScreenDefaultServers -Value 0 -Type DWord; \
    Set-ItemProperty -Path
'HKLM:\SYSTEM\CurrentControlSet\Services\Dnscache\Parameters' -Name
ScreenUnreachableServers -Value 0 -Type DWord

# Shorten DNS cache times
RUN Set-ItemProperty -Path
'HKLM:\SYSTEM\CurrentControlSet\Services\Dnscache\Parameters' -Name MaxCacheTtl
-Value 30 -Type DWord; \
    Set-ItemProperty -Path
'HKLM:\SYSTEM\CurrentControlSet\Services\Dnscache\Parameters' -Name
MaxNegativeCacheTtl -Value 30 -Type DWord
```

```
COPY nginx.conf c:/nginx/conf
WORKDIR /nginx
EXPOSE 80
CMD ["nginx", "-g", "\"daemon off;\""]
```

We had to write a config file that knew about the three instances that we wanted to load balance across

```
#user nobody;
worker_processes 1;

error_log logs/error.log;
error_log logs/error.log notice;
error_log logs/error.log info;

#pid      logs/nginx.pid;

events {
    worker_connections 1024;
}

http {
    upstream myapp1 {
        server dockerplay_mvc_1;
        server dockerplay_mvc_2;
        server dockerplay_mvc_3;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://myapp1;
        }
    }
}
```

We could run the various commands documented in the book to start the instances and add them to the right network. We could then load balance using the NGINX and we could refresh the web page to see that requests were being served by different machines at different times.

[There is a little too much hardwired in by name for my taste. The SDN inside docker runs a DNS that lets you look up other containers by name to get their IP address]

The next chapter of the book looks at Docker Compose. This gives you a way to wire things up using a single configuration file.

```
version: "3"

volumes:
  testdata:

networks:
  frontend2:
  backend2:

services:

  sqlexpress2:
    image: "microsoft/mssql-server-windows-express"
    volumes:
      - testdata:c:\data
    networks:
      - backend2
    environment:
      - sa_password=fddfdfdfssff
      - ACCEPT_EULA=Y

  dbinit:
    build:
      context: .
      dockerfile: Dockerfile
    networks:
      - backend2
    environment:
      - INITDB=true
      - DBHOST=sqlexpress2
      - DBPORT=1433
    depends_on:
      - sqlexpress2

  mvc:
    build:
      context: .
      dockerfile: Dockerfile
    networks:
      - backend2
      - frontend2
    environment:
      - DBHOST=sqlexpress2
      - DBPORT=1433
    depends_on:
      - sqlexpress2
```

```

ports:
- 4020:4020
- 4021:4021

loadbalancer:
image: nginx
build:
  context: ..\nginx
  dockerfile: Dockerfile
ports:
- 8112:80
networks:
- frontend2

```

This is a really neat technology, allowing you to scale the various components up and down. Unfortunately for us, we didn't have an easy way to reconfigure the load balancer when the scaling happens. In the book, the load balancer configuration has "links" and "volumne" lines that allow the compose to pass details of the instantiations of the load balanced service. We didn't have time to look in to this.

The next chapter in the book looks at Docker Swarm. There was no equivalent on Windows, so we didn't try it.

The last chapter of the book looks at allowing debugger access into the container. Visual Studio can do this if you run the appropriate components, but we didn't try too hard to get this working. Later versions of Visual Studio can build containers and automatically configure them to allow debugger access.

I think our main observation was that Windows Docker seems to be a long way behind Docker on Linux.

The book was good as a set of instructions to follow, with the brief explanations helping a little to understand what was going on. Using a book that was a version behind was a good way of forcing us to debug and understand what was happening a little better.

On a related note, there's an [2]interview that discusses Service Fabric which is used to run loads of the Azure infrastructure.

1. [https://www.amazon.co.uk/Essential-Docker-ASP-NET-Core-MVC/dp/1484227778/ref=sr\\_1\\_1?ie=UTF8&qid=1523796683&sr=8-1&keywords=essential+docker+for+asp.net](https://www.amazon.co.uk/Essential-Docker-ASP-NET-Core-MVC/dp/1484227778/ref=sr_1_1?ie=UTF8&qid=1523796683&sr=8-1&keywords=essential+docker+for+asp.net)
2. <https://www.youtube.com/watch?v=MrfcP6dS6mU&t=1236s&list=WL&index=40>

## **12.3 June**

### **12.3.1 What are micro-tasks in the browser all about? (2018-06-18 06:27)**

I gave a quick lightning talk at work about micro-tasks in the browser, based on a recent talk by Jake Archibald. The slides are available [1]here.

1. <https://onedrive.live.com/view.aspx?cid=3f21df299c355e7f&page=view&resid=3F21DF299C355E7F!7157&parId=3F21DF299C355E7F!129&app=PowerPoint>

---

## **12.4 August**

### **12.4.1 Kubernetes is the winner (2018-08-07 20:59)**

[1]Kubernetes: Up & Running by Kelsey Hightower, Brendan Burns and Joe Beda

Everywhere you go these days, it's all about containers and how they should be orchestrated. [2]Software Engineering Daily had a great series about several container management systems, and so it was time to get the book about Kubernetes, by several of the founders of the project. There is recent blog post on the history of the project [3]here.

The book itself is really good. It explains the need for an orchestration framework, and demonstrates the various parts of the Kubernetes system. It starts by showing you how to deploy a Kubernetes cluster and works through the use of the kubectl commands. It moves on to explain pods, and the labels and annotations that you can attach to the containers that are being managed. This is very hands on, working against a demonstration container that the authors have made available.

The following chapters cover service discovery, Replicasets, Daemonsets, Jobs and ConfigMaps and then there is a chapter that covers deployments and upgrades. The last two chapters cover how you integrate storage with your applications and how to deploy some real world applications.

The book, as you would expect, covers the material really well. If you want to try the material out on the Azure cloud, the Azure documentation contains [4]some worked tutorials.

If you need to understand Docker a little better, then I found [5]this post useful. Ben Hall also did a recent talk on [6]other container technologies. A competing idea is serverless, and there is a recent paper that looks at [7]the implementation behind this for the three major cloud platforms.

1. <https://www.amazon.co.uk/Kubernetes-Running-Dive-Future-Infrastructure/dp/1491935677>
  2. <https://softwareengineeringdaily.com/>
  3. <https://cloudplatform.googleblog.com/2016/07/from-Google-to-the-world-the-Kubernetes-origin-story.html?m=1>
  4. <https://docs.microsoft.com/en-us/azure/aks/tutorial-kubernetes-prepare-app>
  5. <https://ericchiang.github.io/post/containers-from-scratch/>
  6. <https://youtu.be/1RetLodCL1g>
  7. <http://pages.cs.wisc.edu/~liangw/pub/atc18-final298.pdf>
- 

## 12.4.2 Designing system for scalability (2018-08-10 06:07)

I've been doing some reading on designing systems for scalability, and I thought I could quickly post some of the useful YouTube videos that I have found. There are numerous system design problems and solutions that have videos on YouTube, but I haven't included the ones that I have watched.

Eventually I came across[1] this video on system design, that actually gives a good list of the various technologies that are used in some of the most scalable applications available today.

[2]This is an introduction to how Twitter is implemented, and mentions ideas like fanning-out to Redis and Memcached. There are videos about [3]Facebook and [4]Instagram

The [5]choice of database is obviously important, and it is useful to understand the in-memory databases like [6]Redis. Transactions also come up, [7]via myths and surprises, and how the [8]transaction levels relate to the CAP theorem.

[9]Uber deal with some of the reliability data by storing data on their drivers' mobile phones.

GraphQL came up several times as an alternative to REST APIs. It often requires fewer round trips, and makes tool support easy by using a schema. There is an introduction [10]here and the [11]coding of a server (which explains what you can do about the N+1 problem using an [12]online demo system).

There is a good general talk about lessons learned [13]here.

I had heard about Bloom Filters before, but hadn't come across the [14]Count-min sketch algorithm

1. [https://www.youtube.com/watch?v=UzLMhqg3\\_Wc](https://www.youtube.com/watch?v=UzLMhqg3_Wc)
2. <https://youtu.be/J5auCY4ajK8>
3. <https://youtu.be/dlixGkelP9U>

4. <https://youtu.be/hnpzNAPiCOE>
  5. <https://youtu.be/p3ytSdUQZzA>
  6. <https://youtu.be/qr4FVhBTq0I>
  7. <https://youtu.be/5ZjhNTM8XU8>
  8. <http://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html>
  9. <https://youtu.be/0EhTOKcwRok>
  10. [https://youtu.be/\\_9RgHXqH8J0](https://youtu.be/_9RgHXqH8J0)
  11. <https://youtu.be/Tpf9kVE2AY8>
  12. <https://launchpad.graphql.com/new>
  13. <https://youtu.be/kb-m2fasdDY>
  14. [https://en.wikipedia.org/wiki/Count%20%93min\\_sketch](https://en.wikipedia.org/wiki/Count%20%93min_sketch)
- 

## 12.5 October

### 12.5.1 Reliable, Scalable and Maintainable Systems (2018-10-02 18:46)

[1]Designing Data-Intensive Applications: The big ideas behind reliable, scalable and maintainable systems by Martin Kleppmann

I was lucky to have a six week sabbatical over the summer, and felt that it would be a good time to read up on the technologies behind some of the large scale distributed systems that are around at the moment. This book is a great read for getting up to speed.

It has three sections. The first is on the foundations of data systems, and starts with a quick discussion of what the words reliability, scalability and maintainability actually mean. The book then moves on to the various data models, where the author discusses the birth of NoSQL , query languages and the various graph databases. The underlying implementations are covered, including B-trees, SSTables and LSM-trees, and various indexing structures. The section finishes with a discussion of data encoding and evolution.

The second section covers distributed data, and there are chapters on replication, partitioning and the rather slippery notion of a transaction. Distributed systems can fail in many interesting ways, all covered in the next chapter, including some discussion of Byzantine faults. The final chapter in the section talks about consistency and consensus. In all of the discussion the author is really happy to go into low level implementation details, and all of the chapters have lists of references of papers that you can consult for more information.

The final section is on derived data - how do we process the mass of data that we have accumulated. The first chapter is on batch processing, which covers map-reduce and later variants. This is followed by a chapter on stream processing. The final chapter of the book is the author's idea for the future.

This book is a great read. It goes into loads of implementation details which helps the reader really get to grips with the ideas, though it might take more than a single read to

understand the many ideas that are covered.

1. [https://www.amazon.co.uk/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321/ref=sr\\_1\\_1?ie=UTF8&qid=1539524038&sr=8-1&keywords=designing+d](https://www.amazon.co.uk/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321/ref=sr_1_1?ie=UTF8&qid=1539524038&sr=8-1&keywords=designing+d)
- 

## 12.6 November

### 12.6.1 Some more reading (2018-11-12 06:45)

There are three other books that I have recently read without writing up here.

[1]React in action by Mark Tielens Thomas, which was a good introductory book on React. It explains the various concepts and takes you through a number of examples in the first two parts of the book. The third part of the book talks about higher level architecture, covering Redux, server side rendering and there is also a chapter on React Native. I found it a useful informative read.

[2]Redux in action by Marc Garreau, which talks about the Redux state management approach which integrates well with the React architecture (despite not exactly following the Flux architecture). Most of the book covers the use of Redux with React, though it can be used as a standalone component.

[3]Functional Programming: Application and Implementation by Peter Henderson. I read this book while working in industry for a year before going to university (so 1984), and it blew my mind. It talks about why functional programming is such a powerful model, and defines a small Lisp language (LispKit Lisp) which it uses in subsequent chapters. It walks through writing an interpreter for Lisp in Lisp, and then goes on describe an abstract machine which we can compile the language down to, a variant of Landin's SECD machine. The book describes how easy it is to write a compiler in Lisp for the Lisp variant. The book then looks at extensions to the semantics such as delayed evaluated (using lambda and thunks). Towards the end of the book, the author describes how to write a runtime to support LispKit and also gives the abstract machine code for the compiler itself. On GitHub you can find various implements, like[4] this one in F #.

It was great re-reading the book, and I owe it a lot. Functional programming and Lisp spanned the first twenty years of my career, and it was this book that got me started. I taught myself C in order to implement a Lisp interpreter so I could play around, and various papers on OS implementation using LispKit got me really interested in the field.

1. [https://www.amazon.co.uk/React-Action-Mark-Tielens-Thomas/dp/1617293857/ref=sr\\_1\\_1?ie=UTF8&qid=1541942463&sr=8-1&keywords=react+in+action](https://www.amazon.co.uk/React-Action-Mark-Tielens-Thomas/dp/1617293857/ref=sr_1_1?ie=UTF8&qid=1541942463&sr=8-1&keywords=react+in+action)
  2. [https://www.amazon.co.uk/Redux-Action-Marc-Garreau/dp/1617294977/ref=pd\\_sim\\_14\\_1?\\_encoding=UTF8&pd\\_rd\\_i=1617294977&pd\\_rd\\_r=d1197319-e5b4-11e8-a134-a7ffcb7ac2c9&](https://www.amazon.co.uk/Redux-Action-Marc-Garreau/dp/1617294977/ref=pd_sim_14_1?_encoding=UTF8&pd_rd_i=1617294977&pd_rd_r=d1197319-e5b4-11e8-a134-a7ffcb7ac2c9&)
  3. [https://www.amazon.co.uk/Functional-Programming-Implementation-Prentice-Hall-international/dp/0133315797/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1541943303&sr=1-1&keywor](https://www.amazon.co.uk/Functional-Programming-Implementation-Prentice-Hall-international/dp/0133315797/ref=sr_1_1?s=books&ie=UTF8&qid=1541943303&sr=1-1&keywor)
  4. <https://github.com/AshleyF/Lispkit/blob/master/Program.fs>
- 

## 12.6.2 Of CORSe, it's easy to test (2018-11-14 06:44)

I read this [1]informative post on CORS at the weekend, and realised that the best way to get to grips with it, is to try some experiments. I hadn't realised before how easy this would be in C#. It's easy to write a mini-web server that handles a single call using code like this.

```

IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
TcpListener listener = new TcpListener(ipAddress, 8182);
listener.Start();

using (var clt = listener.AcceptTcpClient())
using (NetworkStream ns = clt.GetStream())
using (StreamReader sr = new StreamReader(ns))
using (StreamWriter sw = new StreamWriter(ns))
{
    var msg = sr.ReadLine();
    Console.WriteLine(msg);

    if (msg.StartsWith("GET"))
    {
        string line;
        while ((line = sr.ReadLine()) != "")
            Console.WriteLine(line);

        Console.WriteLine();
        Console.WriteLine();

        sw.WriteLine("HTTP/1.1 200 OK");
        //sw.WriteLine("Access-Control-Allow-Origin:
chrome://newtab");
        sw.WriteLine();
        sw.WriteLine("{a:2020}");
    }
}

```

With that code running, you can start Chrome in Incognito mode, and then run the following code in the console.

```
var h = new XMLHttpRequest()
h.open("GET", "http://localhost:8182", true)
h.send()
```

which gives the error

```
(index):1 Access to XMLHttpRequest at 'http://localhost:8182/' from origin
'chrome://newtab' has been blocked by CORS policy: No
'Access-Control-Allow-Origin' header is present on the requested resource.
```

Uncommenting the Access Control line in the above code allows the request to succeed.

1. <http://performantcode.com/web/do-you-really-know-cors>

---

### 12.6.3 I'm hooked (2018-11-26 06:27)

It always amazes me how simple ideas can transfer from one domain to another.

A while ago I was looking at Azure Durable Functions. In this domain you want to write a function that can call out to another service, but for scalability reasons we'd like to avoid waiting for the result. Instead we'd like to stop the current execution and then reinstate the context when the value is available. Of course the difficulty is that this sounds like special compiler logic (like the type of transform used to implement generators in many languages). Instead, in the case of Azure Durable Functions, the designers demand that the functions are deterministic, and the trick is to store the [1]previous results in an event log. We can then get to the same execution point by replaying the function - instead of making real outgoing calls, the system returns the previously logged return value at each point where the code tries to call out of the context, knowing that this will lead us down the same path that we executed the first time. This lets us wind forward to the same place and then execute the next step.

React has always had two ways of defining a component. You can use a class and override the various lifecycle methods (like `componentDidMount`) or you can write more functional (Pure) components that just return the mark up that they want to render. These functional components can be a lot easier to read, and it can be easier to share logic as it is easy for functions to call other functions. Adding state into such functional components is harder though, and the React team have introduced a concept called Hooks to do this. This again requires the idea of a function where outgoing calls are made in a deterministic order, and

this lets the system use this order to return different values.

There's a good post that explains Hooks [2]here, but the quickest introduction is a talk at [3]React Conf with an informative excerpt [4]here. In the example that they show in the talks, the functional approach makes the code much easier to understand than the class based approach - the latter can lead to related logic being spread across the class's methods whereas the functional approach puts the code together.

While we are on the subject of React, there's a [5]post here talking about Flutter as an alternative to React Native.

1. <https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-checkpointing-and-replay>
  2. [https://medium.com/@dan\\_abramov/making-sense-of-react-hooks-fdbde8803889](https://medium.com/@dan_abramov/making-sense-of-react-hooks-fdbde8803889)
  3. <https://www.youtube.com/watch?v=dpw9EHDh2bM&index=37&t=0s&list=WL>
  4. <https://www.youtube.com/watch?v=wXLF18DsV-I&index=34&list=WL>
  5. <https://medium.com/asos-techblog/flutter-vs-react-native-for-ios-android-app-development-c41b4e038db9>
-



# 2019

## 13.1 January

### 13.1.1 Some odds and ends (2019-01-14 06:21)

My time at [1]Redgate is coming to an end, so I've been reading a fair few books on some of the new technologies that I'll be using in my new job. More on that later.

First some interesting videos and posts from the last month or so.

I've been getting to grips with Comonads as part of my push to understand Haskell in more depth. This talk on implementing the [2]Game of Life using Comonads was interesting, and it also led me to watching [3]this slightly more in depth talk. This talk on [4] UIs being Comonads, part of a series on Purescript was how I started this path.

This article on [5]KVM made sense, and gives enough of an overview to help you understand where KVM sits.

I've been using React a little, and have found it a great technology. Of course, I'd like to understand more about what is happening under the covers, and [6]this [7]series of [8]blog posts helped me understand the reconciliation process between the React tree and rendered browser DOM, by way of the new Fiber engine. This [9]YouTube video is also a good overview.

[10]MySQL High Availability

I found this book in the local library and thought I'd give it a read. I read the older version, which was written just after MySQL had extended the statement level replication to support row based replication. I enjoyed the level of detail, and the hands-on nature which shows you the commands you'll need to run to start the replication process. The idea of using replicas so that writes go to the master and reads go to one or more slaves is an idea that seems to come up all the time in system design examples. This book gave me a good feel for how this all works. After I read this book, I found [11]this blog post which gives a brief summary of the ideas.

[12]Fluent Python

I have done bits and pieces in Python in the past, but wanted to get a better grip of the more advanced features of the language. I was lucky enough to find this book in a local

bookshop.

The book has several sections:

- data structures, which talks about the basic sequence types, as well as dictionaries, sets and text
- functions as objects, which talks about functions as first class entities, and includes decorators and closures
- object oriented idioms which discusses object references, mutability and recycling, the meaning of a Pythonic object, interfaces and inheritance.
- control flow, which covers iterables, iterators and generators, contexts, coroutines and concurrency. This is a part of Python about which I knew nothing, and the book covered it really well.
- metaprogramming, which talks about adding dynamic attributes and properties to classes as well as class level metaprogramming

I really enjoyed the book. It discussed everything at the right level, with enough details to really understand what is happening under the covers. In fact, the author shows the byte code at times to help you understand what is going on.

#### [13]Managing Kubernetes

This book talks about the operational side of managed a Kubernetes installation. As such you get to see a little more about what happens in the standard components, and it also covers monitoring, disaster recovery and authentication which are aspects you really need to understand if your business depends on it.

#### [14]Pro .NET Memory Management

It's impossible to describe how good this book is. It covers the implementation of the .NET garbage collector in great detail, often linking back to the sources. It works through various case studies to show you issues that you may need to work around, and it spends a lot of time showing you how to use ETW and perfview to diagnose issues in your application that are caused by garbage collection. It's a massive book that takes quite a while to read, and I'm going to need to re-read it loads of times to really understand it all.

The level of detail in the book is truly awe-inspiring.

#### [15]Quantum Space

I've always been interested in the unified theories of physics and wanted to understand the progress in Loop Quantum Gravity. This book covers some of the history of the theory and is a good read.

1. <https://www.red-gate.com/>
2. <https://www.youtube.com/watch?v=6eiS2QTQKPE&index=13&list=WL&t=1072s>

3. <https://www.youtube.com/watch?v=F7F-Bz0B670>
  4. <https://www.youtube.com/watch?v=EoJ9xnzG76M&t=2937s>
  5. <https://lwn.net/Articles/658511/>
  6. <https://medium.com/react-in-depth/the-how-and-why-on-reacts-usage-of-linked-list-in-fiber-67f1014d0eb7>
  7. <https://medium.com/react-in-depth/inside-fiber-in-depth-overview-of-the-new-reconciliation-algorithm-in-react-e1c04700ef6e>
  8. <https://medium.com/react-in-depth/in-depth-explanation-of-state-and-props-update-in-react-51ab94563311>
  9. <https://www.youtube.com/watch?v=ZCuYPiUIONs&index=14&list=WL&t=430s>
  10. [https://www.amazon.co.uk/MySQL-High-Availability-Building-Centers/dp/1449339581/ref=sr\\_1\\_1?ie=UTF8&qid=1547374911&sr=8-1&keywords=mysql+high+availability](https://www.amazon.co.uk/MySQL-High-Availability-Building-Centers/dp/1449339581/ref=sr_1_1?ie=UTF8&qid=1547374911&sr=8-1&keywords=mysql+high+availability)
  11. <https://www.brianstorti.com/replication/>
  12. [https://www.amazon.co.uk/Fluent-Python-Luciano-Ramalho/dp/1491946008/ref=sr\\_1\\_1?ie=UTF8&qid=1547378149&sr=8-1&keywords=fluent+python](https://www.amazon.co.uk/Fluent-Python-Luciano-Ramalho/dp/1491946008/ref=sr_1_1?ie=UTF8&qid=1547378149&sr=8-1&keywords=fluent+python)
  13. [https://www.amazon.co.uk/Managing-Kubernetes-Operating-Clusters-World/dp/149203391X/ref=sr\\_1\\_1?ie=UTF8&qid=1547378675&sr=8-1&keywords=managing+kubernetes](https://www.amazon.co.uk/Managing-Kubernetes-Operating-Clusters-World/dp/149203391X/ref=sr_1_1?ie=UTF8&qid=1547378675&sr=8-1&keywords=managing+kubernetes)
  14. [https://www.amazon.co.uk/Pro-NET-Memory-Management-Performance/dp/148424026X/ref=sr\\_1\\_1?ie=UTF8&qid=1547378861&sr=8-1&keywords=pro+.net+memory+management](https://www.amazon.co.uk/Pro-NET-Memory-Management-Performance/dp/148424026X/ref=sr_1_1?ie=UTF8&qid=1547378861&sr=8-1&keywords=pro+.net+memory+management)
  15. [https://www.amazon.co.uk/Quantum-Space-Gravity-Structure-Universe/dp/0198809115/ref=sr\\_1\\_1?ie=UTF8&qid=1547379148&sr=8-1&keywords=quantum+space](https://www.amazon.co.uk/Quantum-Space-Gravity-Structure-Universe/dp/0198809115/ref=sr_1_1?ie=UTF8&qid=1547379148&sr=8-1&keywords=quantum+space)
- 

## 13.2 February

### 13.2.1 And some more books (2019-02-03 07:43)

[1]The Go Programming language by Alan Donovan and Brian Kernighan

I read this as preparation for a new job. It feels like the definitive guide to the Go language. The book is a great learning tool, highlighting practical uses of the language as it covers the syntax, though the language itself will definitely take some getting used to.

[2]Clean Architecture by Robert C Martin

I'm not 100 % sure of how I felt about this book. It mentioned lots of ideas that I had seen before - for example the SOLID principles, and linked them into ideas around components. There is also a 45 year journey through the author's career which was an interesting read, but I don't think it helped much with the concepts.

I've also been watching a lot of videos on Linux and containers. [3]This talk on the difference between containers and hypervisors is really good. The talk mentions technologies such as gvisor which is also [4]covered in this blog post.

1. [https://www.amazon.co.uk/Programming-Language-Addison-Wesley-Professional-Computing-ebook/dp/B0184N7WWS/ref=sr\\_1\\_1?s=digital-text&ie=UTF8&qid=1549143009&sr=1-1&](https://www.amazon.co.uk/Programming-Language-Addison-Wesley-Professional-Computing-ebook/dp/B0184N7WWS/ref=sr_1_1?s=digital-text&ie=UTF8&qid=1549143009&sr=1-1&)
2. [https://www.amazon.co.uk/Clean-Architecture-Craftsmans-Software-Structure-ebook/dp/B075LRM681/ref=sr\\_1\\_1](https://www.amazon.co.uk/Clean-Architecture-Craftsmans-Software-Structure-ebook/dp/B075LRM681/ref=sr_1_1)

?s=digital-text&ie=UTF8&qid=1549143123&sr=1-1&keywords=c  
3. <https://www.youtube.com/watch?v=0aqAMGMUOFs&t=0s&index=2&list=WL>  
4. <https://thenewstack.io/how-to-implement-secure-containers-using-googles-gvisor/>

---

## 13.3 March

### 13.3.1 More things for using K8s for real (2019-03-21 07:45)

In my current job, the team are working on deploying an application onto Azure using Kubernetes, so it was time to do some reading around the technologies that are being used.

[1]Docker is used by the company as the universal packaging mechanism. The engine part of the application is written in C++, with a combination of Go and Python being used to wrap this to produce the final product. Multi-stage docker builds are used to build the application itself, as well as docker containers for running the various tests. The cloud version of the system is also deployed using [2]Kubernetes using more containers.

[3]Docker Up & Running: Shipping Reliable Containers In Production by Sean Kane and Karl Matthias

This book was a great introduction to docker, with the first chapters providing a good introduction as to why containers are a good idea. It covers the common use cases, and goes through the many command line calls you will need to use to understand what is going on. The next chapter is also very good, with a good explanation of what is happening at the Linux level. There is also a chapter on docker-compose, which is interesting though not particularly relevant if you are deploying to Kubernetes.

[4]Terraform Up & Running by Yevgeniy Brikman

This book was also a good read. Terraform is used by the system at work to do the initial provisioning of the AKS cluster and associated networking and external resources. This short book gives a good overview of Terraform, and has a good running example that shows you how it is used for real.

[5]The Master Algorithm: How the Quest for the ultimate learning machine will remake our world by Pedro Domingos

This was an easy read, which starts with a good overview to how machine learning has come of age. The book then describes the 5 tribes of machine learning - the symbolists, the connectionists, the evolutionaries, the Bayesians, and the analogisers, and gives a brief overview of how their techniques work. The rest of the book discusses how these techniques will need to be merged to get to a master algorithm, and the author covers some of the work he has done to do this on the [6]Alchemy system.

1. [https://en.wikipedia.org/wiki/Docker\\_\(software\)](https://en.wikipedia.org/wiki/Docker_(software))
  2. <https://en.wikipedia.org/wiki/Kubernetes>
  3. [https://www.amazon.co.uk/Docker-Shipping-Reliable-Containers-Production/dp/1492036730/ref=pd\\_lpo\\_sbs\\_14\\_t\\_1?encoding=UTF8&psc=1&refRID=1PASH0ZWDYPF2FBBTBDO](https://www.amazon.co.uk/Docker-Shipping-Reliable-Containers-Production/dp/1492036730/ref=pd_lpo_sbs_14_t_1?encoding=UTF8&psc=1&refRID=1PASH0ZWDYPF2FBBTBDO)
  4. [https://www.amazon.co.uk/Terraform-Up-Running-Yevgeniy-Brikman/dp/1491977086/ref=sr\\_1\\_fkmrnull\\_1?keywords=terraform+up+and+running&qid=1552761672&s=books&sr=1-1](https://www.amazon.co.uk/Terraform-Up-Running-Yevgeniy-Brikman/dp/1491977086/ref=sr_1_fkmrnull_1?keywords=terraform+up+and+running&qid=1552761672&s=books&sr=1-1)
  5. [https://www.amazon.co.uk/Master-Algorithm-Ultimate-Learning-Machine/dp/0141979240/ref=sr\\_1\\_1?keywords=the+master+algorithm&qid=1552763218&s=gateway&sr=8-1](https://www.amazon.co.uk/Master-Algorithm-Ultimate-Learning-Machine/dp/0141979240/ref=sr_1_1?keywords=the+master+algorithm&qid=1552763218&s=gateway&sr=8-1)
  6. <https://alchemy.cs.washington.edu/>
- 

## 13.4 April

### 13.4.1 More miscellaneous books (2019-04-29 06:54)

Another change of job, and hence another list of books I've read in the meantime.

#### [1]Designing Distributed Systems by Brendan Burns

This book goes through a number of patterns that you might want to use when designing distributed systems using Kubernetes. It starts by looking at the single node patterns of Sidecars, Ambassadors and Adapters, and then moves on to serving patterns, where the author talks about replicated load-balancing, sharded services, scatter/gather, FAAS and ownership election. There's then a third section on batch patterns, using generic work queues, event driven processing and coordinated batch processing.

The book is fairly thin, but each pattern is described with a motivating example and at least one example of an implementation using the pattern.

#### [2]The Manager's Path by Camille Fournier

The book looks at the various management roles in the high tech industry, and gives advice for getting the best out of each of the management roles. It starts with team level mentoring, looks at the Tech Lead role and then moves to more management-only focussed roles like engineering manager, CTO and VP of engineering. The book is filled with useful advice from the author who has been in each of the roles.

#### [3]Talk to me by James Vlahos

This book talks about the history of voice controlled AI used for the current virtual assistants. To be honest, I didn't find the book technical enough to be really interesting, though it was good to see where the various ideas had come from.

#### [4]How Google works by Eric Schmidt and Jonathan Rosenberg

The book is now fairly old, but gives some interesting details on how Google is managed. In particular, for me, I found the definition of a "Smart Creative" interesting, and the book then discusses how traditional management structures do not really work for such people. The book has chapters on Culture, Strategy, Hiring, Decisions, Communication and Innovation, and all of the chapters discuss experiences at Google in the given area. The book is definitely worth a read.

#### [5]Homo Deus - A brief history of tomorrow by Yuval Noah Harari

This is an interesting and thought provoking read. It looks at what may follow Humanism in the near-future, and where data driven decision making will take the human race.

1. [https://www.amazon.co.uk/Designing-Distributed-Systems-Brendan-Burns/dp/1491983647/ref=sr\\_1\\_1?crid=2KB17GUW9U3J7&keywords=designing+distributed+systems&qid=1556](https://www.amazon.co.uk/Designing-Distributed-Systems-Brendan-Burns/dp/1491983647/ref=sr_1_1?crid=2KB17GUW9U3J7&keywords=designing+distributed+systems&qid=1556)
  2. [https://www.amazon.co.uk/Manager%60s-Path-Camille-Fournier/dp/1491973897/ref=sr\\_1\\_1?keywords=the+managers+path&qid=1556393471&s=gateway&sr=8-1](https://www.amazon.co.uk/Manager%60s-Path-Camille-Fournier/dp/1491973897/ref=sr_1_1?keywords=the+managers+path&qid=1556393471&s=gateway&sr=8-1)
  3. [https://www.amazon.co.uk/Talk-Me-Google AMAZON-Voice-Controlled/dp/1847948057/ref=sr\\_1\\_2?keywords=talk+to+me&qid=1556392829&s=gateway&sr=8-2](https://www.amazon.co.uk/Talk-Me-Google AMAZON-Voice-Controlled/dp/1847948057/ref=sr_1_2?keywords=talk+to+me&qid=1556392829&s=gateway&sr=8-2)
  4. [https://www.amazon.co.uk/How-Google-Works-Eric-Schmidt/dp/1444792490/ref=sr\\_1\\_1?crid=35C5RC1634726&keywords=how+google+works&qid=1556392948&s=gateway&sprefix=ho](https://www.amazon.co.uk/How-Google-Works-Eric-Schmidt/dp/1444792490/ref=sr_1_1?crid=35C5RC1634726&keywords=how+google+works&qid=1556392948&s=gateway&sprefix=ho)
  5. [https://www.amazon.co.uk/Homo-Deus-Brief-History-Tomorrow/dp/1784703931/ref=sr\\_1\\_1?crid=364VZT8RWE92V&keywords=homo+deus+a+brief+history+of+tomorrow&qid=1556393](https://www.amazon.co.uk/Homo-Deus-Brief-History-Tomorrow/dp/1784703931/ref=sr_1_1?crid=364VZT8RWE92V&keywords=homo+deus+a+brief+history+of+tomorrow&qid=1556393)
-

# 2020

## 14.1 March

### 14.1.1 It's been a while (2020-03-16 06:45)

After returning to my previous employer, there's a been a lot to do. Though I've been reading books at the usual rate, I just haven't had time to type up notes on them here. However, it's time to catch up, so here's a list of some good books and papers from recent times.

[1]Linux Observability with BPF by David Calavera and Lorenzo Fontana

I did a [2]lightning talk at work on eBPF. I'd been hearing about eBPF for a while in various blog posts, and the idea of being able to attach VM code at points in the kernel is fascinating, particularly if that code can change the behaviour. [3]I read this book, which turned out to be a great introduction to the subject. It goes though the various parts of the eBPF story - the VM, the places you can attach the code, the BPF tools packages that sit on top of everything, and then covers the networking side of things where eBPF really started. It's a fairly thin book, but is dense - there's a link to a GitHub repository full of lots of good examples, and also instructions on how to set up a VM to run them.

eBPF is an amazing technology and is well worth reading about. [4]There's a quick primer on writing a simple eBPF function here.

Type checking papers related to Haskell

We have been thinking about type checking in our [5]weekly functional programming lunch at work. As part of that I've been reading a few papers related to type checking GADTs and type families in Haskell. This paper is a good introduction to [6]type checking GADTS before you take on the full [7]Outside In paper.

[8]A means of implementing GADTs in OCaml is described here.

[9]Microservices Patterns by Chris Richardson

There's so much talk about Microservices, and we've been trying to get to grips with it at a weekly video lunch we have been having. This is a brilliant book, which pulled together all of the ideas that I've seen in a number of talks. It covers everything - the reasons that you'd take a monolith and break it into Microservices (and when you'd keep a monolith), how you'd

organise the services, how you get transactions between the independent services (sagas and compensation, with various mail box patterns to ensure delivery of messages), querying via an API gateway and read models, and testing strategies and contract tests.

The book is full of examples which makes the concepts a lot easier to follow.

#### [10]Programming Typescript by Boris Cherny

This is a really good book on TypeScript (though it also taught me some JavaScript too). It's very clear and well written and moves through the set of things that TypeScript offers, covering some of the advanced parts of the type system.

We've been using TypeScript at work and this book helped me understand some of the more advanced parts.

The type system of TypeScript can be used to do some complicated things at compile time. [11]This repository shows some good examples.

1. [https://www.amazon.co.uk/Linux-Observability-Bpf-Programming-Performance/dp/1492050202/ref=sr\\_1\\_1?dchild=1&keywords=linux+observability+with+bpf&qid=1584259932&](https://www.amazon.co.uk/Linux-Observability-Bpf-Programming-Performance/dp/1492050202/ref=sr_1_1?dchild=1&keywords=linux+observability+with+bpf&qid=1584259932&)
  2. [https://1drv.ms/b/s!An9eNZwp3yE\\_u0ss51YrwnwfLORM?e=3E4vY5](https://1drv.ms/b/s!An9eNZwp3yE_u0ss51YrwnwfLORM?e=3E4vY5)
  3. [https://www.amazon.co.uk/Linux-Observability-Bpf-Programming-Performance/dp/1492050202/ref=sr\\_1\\_1?dchild=1&keywords=linux+observability+with+bpf&qid=1584259932&](https://www.amazon.co.uk/Linux-Observability-Bpf-Programming-Performance/dp/1492050202/ref=sr_1_1?dchild=1&keywords=linux+observability+with+bpf&qid=1584259932&)
  4. <https://sysdig.com/blog/the-art-of-writing-ebpf-programs-a-primer/>
  5. <https://github.com/red-gate/ClojureLunch/tree/master/simple-typechecker>
  6. <https://www.microsoft.com/en-us/research/publication/complete-and-decidable-type-inference-for-gadts/>
  7. <https://www.microsoft.com/en-us/research/publication/outsideinx-modular-type-inference-with-local-assumptions/>
  8. <https://blog.janestreet.com/more-expressive-gadt-encodings-via-first-class-modules/>
  9. [https://www.amazon.co.uk/Microservice-Patterns-examples-Chris-Richardson/dp/1617294543/ref=sr\\_1\\_1?crid=85I1883Y17L6&dchild=1&keywords=microservices+patterns&qid](https://www.amazon.co.uk/Microservice-Patterns-examples-Chris-Richardson/dp/1617294543/ref=sr_1_1?crid=85I1883Y17L6&dchild=1&keywords=microservices+patterns&qid)
  10. [https://www.amazon.co.uk/Programming-TypeScript-Making-JavaScript-Applications/dp/1492037656/ref=sr\\_1\\_1?dchild=1&keywords=programming+typescript&qid=1584260981&](https://www.amazon.co.uk/Programming-TypeScript-Making-JavaScript-Applications/dp/1492037656/ref=sr_1_1?dchild=1&keywords=programming+typescript&qid=1584260981&)
  11. <https://github.com/ronami/meta-typing/>
- 

## 14.2 August

### 14.2.1 Some Big Data reading (2020-08-23 15:11)

So it's off to join another company soon, and I've needed to get more familiar with Big Data ideas. What have I read as a means of doing that?

First a re-read of [1]Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems

This is a great book that gives an overview of all kinds of distributed systems theory, including issues such as consensus, coverage of the techniques for storing and partitioning massive amounts of data, and an overview of the techniques of processing the data via map/reduce and streaming. Every time I read the book I find some idea that I hadn't fully understood before.

Then, two free books that you can get from the Confluent web site.

Kafka: The Definitive Guide which is (unsurprisingly) a guide to Kafka. It covers how you install and use Kafka and has chapters on the internals of the system together with how you get the reliability guarantees that you need. There are also chapters on monitoring and administering a Kafka cluster, though the chapter I found most interesting was the final chapter on streaming with Kafka, which gives a good overview on how you join streams and tables.

Designing Event-Driven Systems: Concepts and Patterns for Streaming Services with Apache Kafka. The book has sections on event driven systems, explaining what they are and why you should use them, particularly as a way of bringing together disparate data from across a company. It then talks about consistency of the data, concurrent access and how the systems can evolve over time. The book is quite short, but is full of interesting ideas.

The next good source of information was the [2]Data Engineering Podcast. There are several hundred episodes available, including interesting episodes on [3]Presto, [4]Postgres, [5]Kafka, [6]time series databases, [7]Zookeeper and many more. Every episode I have listened to so far has been informative and less than an hour so easy to listen to.

There are also a number of papers that have been interesting to read - [8]Azure storage, [9]Spanner, [10]Chubby, [11]Raft, [12]LSM to name a few. Also, here are a couple of talks about [13]adding transactions to Kafka and [14]lessons learned in the production use of Kafka.

1. [https://www.amazon.co.uk/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321/ref=sr\\_1\\_1?dchild=1&keywords=designing+dataintensive&qid=1598](https://www.amazon.co.uk/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321/ref=sr_1_1?dchild=1&keywords=designing+dataintensive&qid=1598)
2. <https://www.dataengineeringpodcast.com/>
3. <https://www.dataengineeringpodcast.com/prestodb-at-starburst-data-with-kamil-bajda-pawlikowski-episode-3>

2/  
4. <https://www.dataengineeringpodcast.com/postgresql-with-jonathan-katz-episode-42/>  
5. <https://www.dataengineeringpodcast.com/ksqldb-kafka-stream-processing-episode-122/>  
6. <https://www.dataengineeringpodcast.com/timescaledb-round-2-episode-65/>  
7. <https://www.dataengineeringpodcast.com/apache-zookeeper-with-patrick-hunt-episode-59/>  
8. <https://azure.microsoft.com/en-gb/blog/sosp-paper-windows-azure-storage-a-highly-available-cloud-storage-service-with-strong-consistency/>  
9. <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/65b514eda12d025585183a641b5a9e096a3c4be5.pdf>  
10. <https://static.googleusercontent.com/media/research.google.com/en//archive/chubby-osdi06.pdf>  
11. <https://raft.github.io/raft.pdf>  
12. <https://www.cs.umb.edu/~poneil/lsmtree.pdf>  
13. <https://www.youtube.com/watch?v=WscozkoXLHM>  
14. <https://www.youtube.com/watch?v=1vLMuWsfMcA>

---

## 14.3 September

### 14.3.1 System Design Interviews (2020-09-01 06:51)

I've become really interested in the design of big systems, in particular how you can scale using commodity machines to planet size applications. There are lots of sites around that offer material for this purpose (and many YouTube channels too). In particular in the past I'd done a fair amount of reading on the [1]system design primer site. I recently noticed a book on Amazon that looked good for this subject.

[2]System Design Interview: An Insider's Guide by Alex Xu

This book offers some chapters that describe good ways of tackling system design interview questions, including an introductory chapter that takes a web application for a single machine with a small number of users and describes the techniques for scaling that up. After that the book contains 12 example system designs, ranging from consistent hashing and rate scaling, to designing Google Drive. The chapters are short and to the point, and there are numerous typos and sentences that don't quite scan properly, but all in all the book is a really good read which covers the key points for such interviews. Also each of the chapters has a set of 10+ links to additional material such as papers and engineering blogs of the various companies that implemented the problem. This makes it a quick but informative read, and a good way to review the material around this area.

1. <https://github.com/donnemartin/system-design-primer/blob/master/README.md#how-to-approach-a-system-design-interview-question>
  2. [https://www.amazon.co.uk/System-Design-Interview-insiders-Second/dp/B08CMF2CQF/ref=sr\\_1\\_1?dchild=1&keywords=system+design+interviews&qid=1598880277&sr=8-1](https://www.amazon.co.uk/System-Design-Interview-insiders-Second/dp/B08CMF2CQF/ref=sr_1_1?dchild=1&keywords=system+design+interviews&qid=1598880277&sr=8-1)
- 

#### 14.3.2 OAuth 2 in Action (2020-09-02 07:21)

I've never really understood where authentication fits into the OAuth protocol - I kept finding articles that mentioned OpenID Connect was an extension of OAuth that did this, but was still confused as there is often an authentication step when you use OAuth to delegate your authorization decision. How is this not authentication?

I recently found a book that explained this really well - [1]OAuth 2 In Action by Justin Richer and Antonio Sanso.

To me this was the perfect developer book. The authors explain what OAuth is, and go through each of the flows that are supported - OAuth works across many types of device and configuration, from simple delegation of authority between two web sites to handling a mobile device or desktop application talking to a REST backend. Most importantly they have examples in Javascript of each of these scenarios and take you through the Javascript code you need to implement to support the protocol. After covering OAuth the authors talk through the extensions which are built on top of OAuth. One of these extensions is OpenID Connect. Again, we get to implement this, which is definitely the way to get a good understanding of anything.

The book has [2]an associated Github repository for the code examples here.

[3]I did a lightning talk on this at work recently and [4]there's a good talk on YouTube that explains all of this.

1. <https://www.amazon.co.uk/OAuth-Action-Mr-Justin-Richer/dp/161729327X>
2. <https://github.com/oauthinaction/oauth-in-action-code>

3. [https://1drv.ms/p/s!An9eNZwp3yE\\_vGvxYbQBqLXiP7lT?e=RgjAXF](https://1drv.ms/p/s!An9eNZwp3yE_vGvxYbQBqLXiP7lT?e=RgjAXF)

4. <https://youtu.be/9960iexHze0>

---

### 14.3.3 Haskell's Type System is really amazing (2020-09-03 07:04)

I guess it's no surprise, but the Haskell Type system is really amazing. The support for GADTs and other interesting type level ideas like phantom types make the language a fascinating place to understand how much types can help with the construction of only valid programs.

I've recently been reading Sandy Maguire's [1]Thinking With Types which is, as the author claims, *the comprehensive manual for type level programming*.

The book starts from first principles, with Sum/Product and Exponential types and a discussion of the difference between terms, types and kinds. It then goes into detail about all of the type related extensions that GHC supports including GADTs and constraints, Rank-N and existential types. This is followed by a section on computing at the type level - understanding Haskell's dependent type support was one of the main reasons that I bought this book. This also covers type families and defunctionalization.

The book is great but it is going to take a couple of reads to understand the material.

I also read [2]Haskell Design Patterns: Take your Haskell and functional programming skills to the next level by exploring new idioms and design patterns by Ryan Lemmer

This book is also a quick read and about half of it is also connected to using the type system to make your programs more correct. However it also has some great sections on Haskell ideas like traversals, Monads and monad transformers, as well as a discussion of different styles of functional I/O.

At the recent Haskell Love online conference the talks on [3]Haskell to Core helped me to understand some of these ideas as the presenter covers how some of the type equality constraints are presented in the de-sugered intermediate language that GHC users, and Richard Eisenberg's [4]Parameters of Many Flavours also discusses the different types of implicit parameters that are carried by a Haskell function definition (and also discusses what is erased and what is available at runtime).

1. <https://leanpub.com/thinking-with-types>
  2. [https://www.amazon.com/Haskell-Design-Patterns-Ryan-Lemmer/dp/178398872X/ref=tmm\\_pap\\_swatch\\_0?\\_encoding=UTF8&qid=&sr=](https://www.amazon.com/Haskell-Design-Patterns-Ryan-Lemmer/dp/178398872X/ref=tmm_pap_swatch_0?_encoding=UTF8&qid=&sr=)
  3. <https://www.youtube.com/watch?v=fty9QL4aSRC>
  4. <https://www.youtube.com/watch?v=hkus92CX32A&list=WL&index=17&t=10s>
- 

#### **14.3.4 And a final batch of books (2020-09-07 07:03)**

Working from home for six months has made it really easy to get a lot of reading done. This is the final list of books that I've finished during the period.

[1]Good Strategy/Bad Strategy: The difference and why it matters by Richard Rumelt

We went through this book as part of the reading group for Tech Leads at work. The book looks at what a strategy is and contrasts it to the usual set of motivational ideas that we are often told is a strategy. The book emphasises a logical plan based on an analysis of the problem together with reasoning as to why a particular item was targeted. A mix of common sense and good tricks to get a good plan together.

[2]An Elegant Puzzle: Systems of Engineering Management by Will Larson

This again was going to be the object of a reading group; however, the switch to working from home meant that this never happened, but I read my way through the book anyway. I must admit that I found the book hard going.

[3]How Linux works by Brian Ward

I'm going to be moving back to using Linux day to day, and wanted a refresher on the lower level details of Linux. This book is brilliant for that purpose. As the blurb says, it covers all of the basics, though it does this in lots of interesting detail.

[4]Webpack 5 Up and Running: A quick and practical introduction to the JavaScript application bundler by Tom Owens

I wanted to get up to speed by the new version of webpack. To be honest this book just feels like cut and pasted parts of the existing documentation, and it really feels like the book could do with some good proof reading to correct the typos and misspellings.

#### [5]The Daemon, the Gnu, and the Penguin by Peter H Salus

A potted history of Unix and Linux. A quick read by interesting from a historical point of view.

#### [6]Einstein's Unfinished Revolution: The Search for What Lies Beyond the Quantum by Lee Smolin

Lee Smolin has written a loads of books over the years about the search for a unified theory of physics. This is another one that talks about his more recent ideas around quantum mechanics and how we can give it a realist interpretation. I must admit that I have enjoyed all of his books.

#### [7]Programming Rust: Fast, Safe Systems Development by Jim Blandy

There are several people at work who are massive fans of Rust, and this book does a really good job both of explaining the language and discussing the benefits of using Rust to get runtime safety. The book is really good and I will certainly be using the language in the future.

1. [https://www.amazon.co.uk/Good-Strategy-Bad-difference-matters/dp/1781256179/ref=sr\\_1\\_1?dchild=1&keywords=good+strategy+bad+strategy&qid=1599375876&sr=8-1](https://www.amazon.co.uk/Good-Strategy-Bad-difference-matters/dp/1781256179/ref=sr_1_1?dchild=1&keywords=good+strategy+bad+strategy&qid=1599375876&sr=8-1)
2. [https://www.amazon.co.uk/Elegant-Puzzle-Systems-Engineering-Management-ebook/dp/B07QYCHJ7V/ref=sr\\_1\\_2?dc hild=1&keywords=an+elegant+puzzle&qid=1599376108&sr=8-2](https://www.amazon.co.uk/Elegant-Puzzle-Systems-Engineering-Management-ebook/dp/B07QYCHJ7V/ref=sr_1_2?dc hild=1&keywords=an+elegant+puzzle&qid=1599376108&sr=8-2)
3. [https://www.amazon.co.uk/How-Linux-Works-Brian-Ward/dp/1718500408/ref=sr\\_1\\_2?dchild=1&keywords=how+linux+w orks&qid=1599376218&sr=8-2](https://www.amazon.co.uk/How-Linux-Works-Brian-Ward/dp/1718500408/ref=sr_1_2?dchild=1&keywords=how+linux+w orks&qid=1599376218&sr=8-2)
4. [https://www.amazon.co.uk/Webpack-Running-introduction-JavaScript-application/dp/1789954401/ref=sxts\\_sxwds-b ia-wc-drs1\\_0?cv\\_ct\\_cx=webpack+5+up+and+running&dchild](https://www.amazon.co.uk/Webpack-Running-introduction-JavaScript-application/dp/1789954401/ref=sxts_sxwds-b ia-wc-drs1_0?cv_ct_cx=webpack+5+up+and+running&dchild)
5. [https://www.amazon.co.uk/Daemon-Gnu-Penguin-Peter-Salus-ebook/dp/B004ZH30ZW/ref=sr\\_1\\_1?dchild=1&keywords =the+daemon+the+gnu+and&qid=1599376436&sr=8-1](https://www.amazon.co.uk/Daemon-Gnu-Penguin-Peter-Salus-ebook/dp/B004ZH30ZW/ref=sr_1_1?dchild=1&keywords =the+daemon+the+gnu+and&qid=1599376436&sr=8-1)

6. [https://www.amazon.co.uk/Einsteins-Unfinished-Revolution-Search-Quantum/dp/0241004489/ref=sr\\_1\\_1?crid=3LUA CKFEIMNRC&dchild=1&keywords=einsteins+unfinished+revol](https://www.amazon.co.uk/Einsteins-Unfinished-Revolution-Search-Quantum/dp/0241004489/ref=sr_1_1?crid=3LUA CKFEIMNRC&dchild=1&keywords=einsteins+unfinished+revol)
  7. [https://www.amazon.co.uk/Programming-Rust-Jim-Blandy/dp/1491927283/ref=sr\\_1\\_4?dchild=1&keywords=programmin g+rust&qid=1599376653&sr=8-4](https://www.amazon.co.uk/Programming-Rust-Jim-Blandy/dp/1491927283/ref=sr_1_4?dchild=1&keywords=programmin g+rust&qid=1599376653&sr=8-4)
- 

## 14.4 October

### 14.4.1 That's very unlikely (2020-10-25 15:28)

[1]The Art of Statistics: Learning from Data by David Spiegelhalter

This book was a brilliant refresher on statistics, aimed at people without a mathematical background. It considers a number of questions, often based on a clickbait newspaper headlines, and shows how the question should really be analysed using statistical methods. It's a really good read.

Rather strangely, I was reading [2]this paper on a probabilistic programming language and the author contributed to one of the cited papers, though the book does talk about using simulation as a technique for using distributions so I can see how they are related.

I've also been doing some reading on Haskell. For quite some time I've been trying to understand how you can extend the standard Hindley-Milner type inference to handle some of the more interesting features like Phantom types and GADTs. At long last I came across[3] this paper which describes how to do it. This also helps to explain some of the[4] type checker messages that I see from time to time. While doing some reading, I also came across this article on how the IO Monad is implemented, and why you don't get the same kind of guarantees from the unsafe functions for performing IO.

Last, two great videos on [5].NET. Performance improvements in .NET 5 by Stephen Toub which talks about recent optimisations - I hadn't come across some of them before, such as being able to turn off the zero initialization of local variables, [6]What's so hard about pinning? by Maoni Stephens which goes into some implementation details about the .NET garbage collector.

I've also been reading some posts on how Linux debuggers work - these [7]two talk about [8]getting access to registers and this paper talks about [9]how to stop the breakpoints stopping the target process for long. While we are talking about Linux, [10]this post goes into

detail about the durability guarantees behind various Linux file system operations.

I've also been doing more reading on Category Theory, and have again wondered about [11]the proof that polymorphic functions in Haskell correspond to the natural transformations in the relevant category.

1. [https://www.amazon.co.uk/Learning-Data-Statistics-Pelican-Books/dp/0241258766/ref=sr\\_1\\_1?crid=1RCW5E8G6BWI&dchild=1&keywords=the+art+of+statistics&qid=16032129](https://www.amazon.co.uk/Learning-Data-Statistics-Pelican-Books/dp/0241258766/ref=sr_1_1?crid=1RCW5E8G6BWI&dchild=1&keywords=the+art+of+statistics&qid=16032129)
  2. <https://pgm2020.cs.aau.dk/wp-content/uploads/2020/09/tehrani20.pdf>
  3. <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/gadt-pldi.pdf>
  4. <https://free.cofree.io/2020/09/01/type-errors/>
  5. <https://www.youtube.com/watch?v=Ehvr9wXJNEM&list=WL&index=15&t=2579s>
  6. <https://www.youtube.com/watch?v=troNdmHEu2g>
  7. <https://blog.0x972.info/?d=2014/11/13/10/40/50-how-does-a-debugger-work>
  8. <https://www.moritz.systems/blog/how-debuggers-work-getting-and-setting-x86-registers-part-1/>
  9. <https://s3.amazonaws.com/arena-attachments/309033/6f46f21a0abfe4de8f56468953378dfb.pdf>
  10. <https://www.evanjones.ca/durability-filesystem.html>
  11. <https://bartoszmilewski.com/2014/09/22/parametricity-money-for-nothing-and-theorems-for-free/>
-

# 2021

## 15.1 January

### 15.1.1 Streaming Systems (2021-01-10 07:49)

[1]Streaming Systems: The what, where and how of large-scale data processing by Tyler Akidau, Slava Chernyak and Reuven Lax

When I was doing a lot of reading about systems design, there were several times when the [2]Lambda Architecture was mentioned. In several videos I watched on YouTube, the presenter would suggest that you used streaming and approximation algorithms for top-N to get real time performance, and then use MapReduce for the batch processing of data as an overnight job to get the correct results (in non-real time).

This book discusses the many issues with this architecture. For example, it can be quite hard to get the same result from the two approaches, and doing things this way means that you are effectively implementing the processing twice. The author then pushes the fact that streaming systems have now improved to the point where you don't need the batch processing side of things.

The authors take us through the concepts behind the streaming implementations, using [3]Apache Beam as the implementation for the demonstrations. They take us through bounded and unbounded datasets, windows, triggers and the difference between event time and processing time. In order to handle failure the streaming system also needs to handle exactly once semantics which requires persistent state (say via snapshots like [4]Flink).

In the second part of the book, the authors look at how to extend SQL to allow users to express queries using joins. This is preceded by a discussion of the duality between tables and streams, and how you can think of streams as data in motion compared to tables which are a snapshot of the stream's state at a moment in time.

At the very end of the book there is a chapter on the history of large-scale data processing, which starts with MapReduce and looks at things that came after.

I really liked this book. It goes through the concepts, using a series of diagrams to explain how various mixes of triggers and windows would generate results for a long running example, I liked the examples in the Beam DSL and I enjoyed the history and discussion of extending SQL's relationships to allow joins across streams. The book also contains references to loads of interesting papers that I will now have to read.

1. [https://www.amazon.co.uk/Streaming-Systems-Tyler-Akidau/dp/1491983876/ref=sr\\_1\\_2?dchild=1&keywords=streaming+systems&qid=1610185740&sr=8-2](https://www.amazon.co.uk/Streaming-Systems-Tyler-Akidau/dp/1491983876/ref=sr_1_2?dchild=1&keywords=streaming+systems&qid=1610185740&sr=8-2)
  2. [https://en.wikipedia.org/wiki/Lambda\\_architecture](https://en.wikipedia.org/wiki/Lambda_architecture)
  3. <https://beam.apache.org/>
  4. <http://www.vldb.org/pvldb/vol10/p1718-carbone.pdf>
- 

### 15.1.2 Data Science from scratch (2021-01-11 07:52)

[1]Data Science From Scratch: First Principles with Python by Joel Grus

I absolutely loved this book. It has chapters introducing many aspects of data science, and then has many code examples of implementing many aspects of machine learning. For me, seeing the implementation details of the various algorithms, made everything fit into place. I should also say that the author has a great writing style with loads of witty remarks.

The book starts with a quick introduction to Python and then shows you how to use the matplotlib Python library to visualize your data. This is followed by chapters that give a quick introduction to linear algebra, statistics, probability and hypothesis testing, all with examples in Python. The next chapter looks at gradient descent which is going to be used in many of the chapters that follow. This is followed by two practical chapters on how to actually get hold of data from files and via the web, and parts of Python like namedtuples that will help you work with it effectively.

The following chapters then go through the various machine learning algorithms: k-nearest neighbours, naive Bayes, linear regression, multiple regression, logistic regression, decision trees, neural networks, deep learning, and clustering. This is followed by chapters on natural language processing, network analysis, recommender systems, and the some final chapters cover databases and sql, map reduce and data ethics.

I think that the book is just right. The material explains the algorithm to just the right depth, and the Python code makes it easy to see how you actually implement it.

---

1. [https://www.amazon.co.uk/Data-Science-Scratch-Joel-Grus/dp/1492041130/ref=sr\\_1\\_1?crid=1SKZGJDSULQS9&dchild=1&keywords=data+science+from+scratch%2C+2nd+edition&qid=1615800000&sprefix=dat%2Caps%2C347](https://www.amazon.co.uk/Data-Science-Scratch-Joel-Grus/dp/1492041130/ref=sr_1_1?crid=1SKZGJDSULQS9&dchild=1&keywords=data+science+from+scratch%2C+2nd+edition&qid=1615800000&sprefix=dat%2Caps%2C347)

### **15.1.3 That's the way to do it (2021-01-12 07:55)**

[1]The Design of Web APIs by Arnaud Lauret

[2]API first seems to be the way that people are writing applications these days.

This book is aimed at people developing REST APIs backed by an OpenAPI schema, and does a good job of covering a number of issues.

The first part deals with designing an API from the point of view of the user, emphasising that the API should make it easy for the user to solve the tasks that they want to do, rather than simply exposing the implementation - just making the implementation available is easy for the writer, but can make the API confusing to use and can force the user to learn names for internal implementation details. The next part of the book covers REST, and goes through the various ways that an API could be broken down into resources and the verbs that you could use to manipulate them. That is followed by a section on how to describe the API using OpenAPI.

The author then discusses how to make the API predictable, straightforward to use and secure before discussing how to make the API evolvable by versioning as well as how you go about documenting it for clients.

The discussions are good with lots of useful ideas, and I learned a few HTTP headers and codes that I didn't know before - the Sunset header and the 207 status code for example. There is also material on server side events and gRPC.

1. [https://www.amazon.co.uk/Design-Everyday-APIs-Arnaud-Lauret/dp/1617295108/ref=sr\\_1\\_1?dchild=1&keywords=the+design+of+web+apis&qid=1610192667&sr=8-1](https://www.amazon.co.uk/Design-Everyday-APIs-Arnaud-Lauret/dp/1617295108/ref=sr_1_1?dchild=1&keywords=the+design+of+web+apis&qid=1610192667&sr=8-1)
  2. <https://swagger.io/resources/articles/adopting-an-api-first-approach/>
- 

#### 15.1.4 Links aplenty (2021-01-13 07:08)

When I was at Facebook, I didn't really have time to blog about interesting blog posts that I'd come across, so I've accumulated a few interesting reads in my browser bookmarks.

[1]An introduction to SSA and the phi function.

[2]Paxos explained

[3]Tracing in Linux using eBpf

[4]The complexity of sliding block puzzles

[5]Some notes on proving the independence of the Continuum Hypothesis

[6]Issues writing a Linux kernel module (and RCU)

Some issues with Nagle's algorithm in [7]this post and [8]this one about delayed Ack ARM processors,[9] lock-free and [10]branch processing

[11]Modern storage and the failure of the supplied Apis

[12]Python at scale using strict modules

[13]Defunctionalization

[14]Module initializers in C #

[15]Avoiding iCache misses

[16]Linear types in Haskell

[17]The security of Helm charts

[18]Perceus: Strict reference counting for the Koka language

[19]How Apache Flink snapshots state so that it can restore on failure

[20]How debugging Blazor WebAssembly works

[21]Compile time dependency injection for C # - a way to improve application startup

[22]There's an interesting course on reinforcement learning on YouTube with the [23]slides here. This is a great explanation starting from Markov Processes and building up to the systems that DeepMind used to solve many hard problems.

And lastly [24]a plan for preparing for a software engineering interview at Facebook.

1. <https://blog.yossarian.net/2020/10/23/Understanding-static-single-assignment-forms>
2. <https://explain.yshui.dev/distributed%20system/2020/09/20/paxos.html>
3. <https://zinascii.com/2020/fbt-args-and-stack.html>
4. <https://groups.csail.mit.edu/mac/users/bob/sliding-blocks.pdf>
5. <https://www.scottaaronson.com/blog/?p=4974>
6. <https://reberhardt.com/blog/2020/11/18/my-first-kernel-module.html>
7. <https://vorner.github.io/2020/11/06/40-ms-bug.html>
8. <http://www.stuartcheshire.org/papers/NagleDelayedAck/>
9. <https://randomascii.wordpress.com/2020/11/29/arm-and-lock-free-programming/>
10. <http://sandsoftwaresound.net/arm-cortex-a72-fetch-and-branch-processing/>
11. <https://itnext.io/modern-storage-is-plenty-fast-it-is-the-apis-that-are-bad-6a68319fbc1a>
12. <https://instagram-engineering.com/python-at-scale-strict-modules-c0bb9245c834>
13. [https://www.joachim-breitner.de/blog/778-Don%E2%80%99t\\_think%2C\\_just\\_defunctionalize](https://www.joachim-breitner.de/blog/778-Don%E2%80%99t_think%2C_just_defunctionalize)
14. <https://khalidabuhakmeh.com/module-initializers-in-csharp-9>
15. <https://paweldziepak.dev/2019/06/21/avoiding-icache-misses/>
16. <https://arxiv.org/pdf/1710.09756.pdf>
17. <https://dlorenc.medium.com/whos-at-the-helm-1101c37bf0f1>
18. <https://www.microsoft.com/en-us/research/uploads/prod/2020/11/perceus-tr-v1.pdf>
19. <http://www.vldb.org/pvldb/vol10/p1718-carbone.pdf>
20. <https://safia.rocks/blog/blazor-wasm-debugging/>
21. <https://github.com/pakrym/jab>
22. [https://www.youtube.com/watch?v=2pWv7G0vuf0&list=PLqYmG7hTraZBiG\\_XpjnPrSNw-1XQaM\\_gB&index=2&t=0s](https://www.youtube.com/watch?v=2pWv7G0vuf0&list=PLqYmG7hTraZBiG_XpjnPrSNw-1XQaM_gB&index=2&t=0s)
23. <https://www.davidsilver.uk/teaching/>
24. <https://daqo.medium.com/facebook-senior-software-engineer-interview-the-only-post-youll-need-to-read-e4604ff2336d>

---

### 15.1.5 Effective STL (2021-01-14 07:28)

[1]Effective STL - 50 Specific Ways To Improve Your Use of the Standard Template Library by Scott Meyers

While I was at Facebook, I had the chance to write some C++ code, and C++ seems to have really moved on since I last wrote it. In particular, facilities like [2]Move semantics took some work to understand. The language now has some nice features like lambda expressions (though the lack of garbage collection makes them a little trickier to use than C #).

This book is a little out of date, but the content was still good for me to understand the STL, though I think I gained most C++ from watching lots of [3]CppCon talks and reading various posts on [4]this blog which includes this guide to a [5]feature of C++ 20.

1. [https://www.amazon.co.uk/Effective-STL-Specific-Professional-Computing/dp/0201749629/ref=sr\\_1\\_1?dchild=1&keywords=effective+stl&qid=1610201886&sr=8-1](https://www.amazon.co.uk/Effective-STL-Specific-Professional-Computing/dp/0201749629/ref=sr_1_1?dchild=1&keywords=effective+stl&qid=1610201886&sr=8-1)
  2. <https://herbsutter.com/2020/02/17/move-simply/>
  3. [https://www.youtube.com/channel/UCM1GfpWw-RUdWX\\_JbLCukXg](https://www.youtube.com/channel/UCM1GfpWw-RUdWX_JbLCukXg)
  4. <https://www.bfilipek.com/2020/04/variant-virtual-polymorphism.html>
  5. <https://www.bfilipek.com/2020/10/understanding-invoke.html>
- 

### 15.1.6 MapReduce for the win (2021-01-25 06:23)

[1]MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems by Donald Moner and Adam Shook

This book is fairly old now - the version I read was published in December 2012, and I was lucky enough to find it in a pile of books that I had been intending to read for some time. I've also been doing a lot of reading about processing Big Data using streaming systems, and have become interested in the so-called [2]Stream-Table duality. These kind of systems have the intention of being more real time than older systems designed using MapReduce, and so it's good to read a book that highlights all of the great things that MapReduce can do.

The book is full of practical examples. It is aimed at Hadoop (1.0.3) and uses a stackoverflow dataset for the example implementation of many of the patterns.

The first chapter of the book covers a brief history of MapReduce, and contains a pointer to the original paper, [3]MapReduce: Simplified Data Processing on Large Clusters. The example in this chapter is the canonical word count in Hadoop.

This is then followed by chapters on various patterns.

Summarization patterns looks at ways of counting in order to analyse data - various metrics like average, min and max values, standard deviation and generating inverted indexes. This

chapter also mentions the use of a combiner - using a reduce like operation before the data has been shuffled in order to improve performance and cut down on the amount of data that needs to be transferred.

Filtering patterns looks at efficiently slimming down the data, and includes discussions on generating Top-N and using Bloom filters to make it possible to quickly get rid of data we are not interested in. For the latter pattern, we need to build the filter and then make it accessible to machine doing the processing.

Data organisation patterns which looks at efficient partitioning, binning and sorting the data.

Join patterns that looks at how we efficiently implement the various join patterns on top of MapReduce. This includes joining against the data set that we are processing, as well as joining against external static data.

There are then two chapters that talk more about the meta issues. How jobs can be chained in Hadoop, and also how we can join different MapReduce jobs together to make things more efficient. There is also a chapter on various input and output patterns in Hadoop.

The book was a quick but enjoyable read, and it makes it really clear how powerful the MapReduce model is, as it allows a vast number of processing methods to be expressed using the same framework.

1. [https://www.amazon.co.uk/MapReduce-Design-Patterns-Effective-Algorithms/dp/1449327176/ref=sr\\_1\\_1?dchild=1&keywords=mapreduce+design+patterns&qid=1610898736&sr=8](https://www.amazon.co.uk/MapReduce-Design-Patterns-Effective-Algorithms/dp/1449327176/ref=sr_1_1?dchild=1&keywords=mapreduce+design+patterns&qid=1610898736&sr=8)
  2. <https://docs.confluent.io/platform/current/streams/concepts.html#duality-of-streams-and-tables>
  3. <https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf>
- 

## 15.2 February

### 15.2.1 Getting up to speed on K8s (2021-02-15 06:45)

I've just moved teams at work, and that has given me a chance to work on a system that runs on top of Kubernetes. It's been a while, maybe two years, since I last did anything serious with Kubernetes, so it was time to get out some of the old books.

[1]Kubernetes Up and Running by Kelsey Hightower, Brenda Burns and Joe Beda is a great introduction to the main ideas behind Kubernetes, and the main components. The book is fairly short, but is full of simple examples to demonstrate ideas such as Services and Pods, and gives a great overview of Kubernetes and how to use it.

The one thing that it misses (for me) is how it all works under the covers, and I enjoyed [2]Managing Kubernetes by Brendan Burns and Craig Tracey which gives you some insight into this with chapters on installing Kubernetes on a cluster, the API server and the scheduler which put things into perspective. The book is also aimed at describing how you maintain a real world K8s cluster and so there are additional chapters on user management, authorization, networking and disaster recovery.

The next important thing to learn about is Operators how how you can use custom resource definitions and the desired state reconciliation loop to define your own higher levels concepts. There are blog posts [3]like this one that help to explain this.

You're going to need some understanding of docker to really understand how it all hangs together and I enjoyed [4]Docker Up and Running by Kane and Matthias as a way to understand this. This talks a lot about how to build reliable containers for production use, and goes into some of the[5] low level details of debugging container.

It's also worthwhile looking at some of the issues people have had with Kubernetes in the past [6]like these failure stories - I ran into the missing anti-affinity rules problem in the past when I worked on a K8s system. There's also [7]an interesting article on how it might be designed now.

There a couple of additional books such as [8]Designing Distributed System by Brendan Burns which talks about some of the common patterns such as sidecars which are included into pods to add extra functionality. I also read [9]Practical Microservices with Dapr and .NET by Davide Bedin which offers a more Microsoft focused view of using Kubernetes and the [10]Distributed Application Runtime. This book covers the major ideas of Dapr like service discovery, state management and virtual actors and shows their use in the context of .NET Core applications.

1. [https://www.amazon.co.uk/Kubernetes-Running-Dive-Future-Infrastructure/dp/1492046531/ref=sr\\_1\\_1?crid=NW85RUFCUJC&dchild=1&keywords=kubernetes+up+and+running+di](https://www.amazon.co.uk/Kubernetes-Running-Dive-Future-Infrastructure/dp/1492046531/ref=sr_1_1?crid=NW85RUFCUJC&dchild=1&keywords=kubernetes+up+and+running+di)
2. [https://www.amazon.co.uk/Managing-Kubernetes-Operating-Clusters-World/dp/149203391X/ref=sr\\_1\\_1?crid=2GZ50LTMFEEOK&dchild=1&keywords=managing+kubernetes&qid=1613](https://www.amazon.co.uk/Managing-Kubernetes-Operating-Clusters-World/dp/149203391X/ref=sr_1_1?crid=2GZ50LTMFEEOK&dchild=1&keywords=managing+kubernetes&qid=1613)
3. <https://medium.com/faun/writing-your-first-kubernetes-operator-8f3df4453234>
4. [https://www.amazon.co.uk/Docker-Shipping-Reliable-Containers-Production/dp/1492036730/ref=sr\\_1\\_2?dchild=1&](https://www.amazon.co.uk/Docker-Shipping-Reliable-Containers-Production/dp/1492036730/ref=sr_1_2?dchild=1&)

```
keywords=docker+up+and+running&qid=1613319840&quartzVe
5. https://alexei-led.github.io/post/k8s_node_shell/
6. https://k8s.af/
7. https://blog.dave.tf/post/new-kubernetes/
8. https://www.amazon.co.uk/Designing-Distributed-Systems-Brendan-Burns/dp/1491983647/ref=sr_1_4?dchild=1&key
words=kubernetes+design+patterns&qid=1613320746&sr=8-4
9. https://www.amazon.co.uk/Practical-Microservices-Dapr-NET-cloud-native/dp/1800568371/ref=sxts_sxwds-bia-
wc-rsf1_0?cv_ct_cx=practical+microservices+with+Dapr&dch
10. https://dapr.io/
```

---

## 15.3 April

### 15.3.1 Some more books (2021-04-05 18:34)

As usual, I got a load of interesting books that I bought with credits I was given for Xmas.

[1]gRPC Up and Running: Building Cloud Native Application With Go and Java for Docker and Kubernetes by Kasun Indrasiri and Danesh Kuruppu

This is a fairly short book at less than 200 pages, but gives a good introduction to gRPC, including many worked examples in both Go and Java. It talks you thorough some of the different ways that applications can interact, including brief coverage of Thrift and GraphQL, and then jumps into a worked example implemented in both Go and Java so you can see how the various interface specifications are mapped into the two languages, gRPC lets clients and servers stream values as part of a method invocation, so these streams need to manifest themselves naturally in the implementation language. The book then discusses how everything is implemented on top of the HTTP/2 protocol, and then looks at some advanced features like interceptors (for before and after send and receive actions like authorization) and load balancing and deadlines (you can fail a call if it takes too long). There are then chapters on securing the communication channel, testing via a CI pipeline and some other useful projects.

[2]C++ Move Semantics: The Complete Guide by Nicolai Josuttis

Move semantics seem like a valuable optimization in a call-by-value language, though the interaction where the moved from object is put into a default state is still very weird to me. Anyway this book does a good job of describing the semantics and the why for move semantics. I enjoyed it but it all feels very complicated, and it appears to be easy to get into the domain

of undefined behaviour.

### [3]The GO Programming Language by Alan Donovan and Brian Kernighan

I have read this book in the past, but have just started writing GO at work, so though I should give it another read. The book is a little old now and doesn't cover parts of GO like the module system, though there are many blog posts that explain this part of the language. The book is rich with examples that help communicate the style of GO programming, and the authors are happy to express strong views on aspects such as testing and interfaces. This is a really good read and covers the language really well.

### [4]What We Cannot Know by Marcus du Sautoy

This book explores where the human understanding currently ends and how far it might expand in the future in a number of domains, from quantum physics, cosmology, logic and artificial intelligence. It's a good read and very interesting.

1. [https://www.amazon.co.uk/gRPC-Running-Building-Applications-Kubernetes/dp/1492058335/ref=sxts\\_sxwds-bia-wc-rsf1\\_0?cv\\_ct\\_cx=grpc+up+and+running&dchild=1&keywords](https://www.amazon.co.uk/gRPC-Running-Building-Applications-Kubernetes/dp/1492058335/ref=sxts_sxwds-bia-wc-rsf1_0?cv_ct_cx=grpc+up+and+running&dchild=1&keywords)
  2. [https://www.amazon.co.uk/Move-Semantics-Complete-Guide-First/dp/3967309002/ref=sr\\_1\\_1?dchild=1&keywords=c%2B%2B+move+semantics&qid=1617560616&s=books&sr=1-1](https://www.amazon.co.uk/Move-Semantics-Complete-Guide-First/dp/3967309002/ref=sr_1_1?dchild=1&keywords=c%2B%2B+move+semantics&qid=1617560616&s=books&sr=1-1)
  3. [https://www.amazon.co.uk/Programming-Language-Addison-Wesley-Professional-Computing/dp/0134190440/ref=sr\\_1\\_1?qid=1HC1Q86WQ8CAU&dchild=1&keywords=the+go+program](https://www.amazon.co.uk/Programming-Language-Addison-Wesley-Professional-Computing/dp/0134190440/ref=sr_1_1?qid=1HC1Q86WQ8CAU&dchild=1&keywords=the+go+program)
  4. [https://www.amazon.co.uk/What-Cannot-Know-consciousness-explained/dp/0007576595/ref=sr\\_1\\_1?dchild=1&keywords=what+we+cannot+know&qid=1617561045&s=books&sr=1-1](https://www.amazon.co.uk/What-Cannot-Know-consciousness-explained/dp/0007576595/ref=sr_1_1?dchild=1&keywords=what+we+cannot+know&qid=1617561045&s=books&sr=1-1)
- 

### 15.3.2 And a little gRPC (2021-04-18 07:10)

I recently did a lightning talk at work on gRPC. [1]The slides are available here.

1. [https://1drv.ms/p/s!An9eNZwp3yE\\_vQq07TM43NQVPTxq?e=CbIgq7](https://1drv.ms/p/s!An9eNZwp3yE_vQq07TM43NQVPTxq?e=CbIgq7)
-

### 15.3.3 And yet more links (2021-04-19 07:26)

The trouble with reading Hacker News and other sites every day is that I end up with a long list of links to interesting posts that tend to collect until I get chance to read them. Here's a set of links from recent days.

[1]Container networking is simple which gives a set of ip and nsenter commands to understand how Docker set up its networking.

[2]Kubernetes apply v patch v replace which talks about why there are three different kubectl commands

[3]crun, an OCI container runtime written in C

[4]Writing your first Kubernetes operator

[5]Kubernetes failure stories

[6]Get a shell into a Kubernetes node

[7]Kubernetes operator best practices

[8]Docker image history modification

[9]Docker without docker

[10]A compile dependency injection framework that uses source generators to do the work

[11]Target typed expressions in C #

[12]Go modules

[13]How AKKA clusters work

[14]Migrating millions of concurrent websockets to Envoy

[15]How we scaled Github using a sharded rate limiter

[16]Making MsQUIC blazingly fast

[17]TLB and pagewalk coherence in x86 processors

[18]All about thread local storage

[19]Speculating the x86 instruction set

[20]The microarchitecture behind meltdown

[21]Software development topics I've changed my mind on

[22]Cupid, the back story

[23]Column store in Sql Server

[24]Combinators, a centennial view

[25]The visitor pattern is Church Encoding

1. <https://iximiuz.com/en/posts/container-networking-is-simple/>
2. <https://blog.atomist.com/kubernetes-apply-replace-patch/#:~:text=When%20you%20can't%20patch,force%20on%20the%20command%20line.>
3. <https://github.com/containers/crun>
4. <https://faun.pub/writing-your-first-kubernetes-operator-8f3df4453234?gi=70cf78b3825d>
5. <https://k8s.af/>
6. [https://alexei-led.github.io/post/k8s\\_node\\_shell/](https://alexei-led.github.io/post/k8s_node_shell/)
7. <https://www.openshift.com/blog/kubernetes-operators-best-practices>
8. <https://www.justinsteven.com/posts/2021/02/14/docker-image-history-modification/>
9. <https://fly.io/blog/docker-without-docker/>
10. <https://github.com/pakrym/jab>
11. <https://developers.redhat.com/blog/2021/03/30/c-9-top-level-programs-and-target-typed-expressions/>
12. <https://www.practical-go-lessons.com/chap-17-go-modules>
13. <https://www.lightbend.com/blog/how akka-cluster-works-actors-living-in-a-cluster>
14. <https://slack.engineering/migrating-millions-of-concurrent-websockets-to-envoy/>
15. <https://github.blog/2021-04-05-how-we-scaled-github-api-sharded-replicated-rate-limiter-redis/>
16. <https://techcommunity.microsoft.com/t5/networking-blog/making-msquic-blazing-fast/ba-p/2268963>
17. <https://blog.stuffedcow.net/2015/08/pagewalk-coherence/>
18. <https://maskray.me/blog/2021-02-14-all-about-thread-local-storage>
19. <https://blog.can.ac/2021/03/22/speculating-x86-64-isa-with-one-weird-trick/>
20. <https://blog.stuffedcow.net/2018/05/meltdown-microarchitecture/>
21. <https://chriskiehl.com/article/thoughts-after-6-years>
22. <https://dannorth.net/2021/03/16/cupid-the-back-story/>
23. <https://www.nikoport.com/columnstore/>
24. <https://arxiv.org/ftp/arxiv/papers/2103/2103.12811.pdf>
25. <https://www.haskellforall.com/2021/01/the-visitor-pattern-is-essentially-same.html>

---

## 15.4 May

### 15.4.1 Processing unbounded data with Sql like languages (2021-05-24 05:52)

I've been doing lots of reading and listening to various podcasts about processing unbounded data using streaming variants of Sql. I did [1]a quick lightning talk at work on the subject.

1. [https://1drv.ms/p/s!An9eNZwp3yE\\_vQ7k-40BNa5nEoAq?e=ewoxf0](https://1drv.ms/p/s!An9eNZwp3yE_vQ7k-40BNa5nEoAq?e=ewoxf0)

---

## 15.5 June

### 15.5.1 Let's Flink about it some more (2021-06-21 06:03)

I'm still interested in the idea of streaming systems, despite one of the recent Software Daily podcasts suggesting that the industry is moving away from this style.

There's a free O'Reilly book, [1]Streaming Integration, that is an introduction to why streaming is important to a business. Getting accurate results quickly from data can be a big business advantage.

There are [2]some [3]videos from Flink Forward 2020 concerning the use of Sql as a query language for streaming data. And I have ordered some books on Flink to get a better understanding of how it all works.

Of course, lots of these systems use micro-batching to actually do the processing. It is, however, an interesting question about how you take queries using map, aggregates like sum, and join over multisets of data and work out how to efficiently re-calculate results given some extra input data. There's a an interesting implementation [4]here and an [5]explanation of how you might go about doing this, which is certainly worth a watch.

In other .NET related news, if you are thinking of porting an ASP.NET MVC application to .Net Core then [6]this ebook is a good read, and [7]this ebook is a discussion of how you might move a ASP.NET WebForms application to Blazor. If you think Linq is amazing, [8]Reaqtor has been made open source, and there are some good introductory blog posts about the extension of Rx to support remote services including the [9]serialization of expression trees for transmission across the wire (Bonsai trees) and taking snapshots of service state to allow us to restart. The book, A Little History of Reaqtor, which is available on the front page of the website, tells the story of how this was all developed over the years.

Oh, and if you are interested in logic and proof theory, [10]this podcast was a great historical overview of the development of the subject.

1. <https://go2.striim.com/oreilly-streaming-integration-book>
2. <https://www.youtube.com/watch?v=KDD8e4GE12w&list=WL&index=62&t=335s>
3. [https://www.youtube.com/watch?v=UnCkwIp\\_614&list=WL&index=59&t=59s](https://www.youtube.com/watch?v=UnCkwIp_614&list=WL&index=59&t=59s)
4. <https://github.com/jamii/dida>
5. <https://github.com/jamii/dida/blob/main/docs/why.md>
6. <https://docs.microsoft.com/en-us/dotnet/architecture/porting-existing-aspnet-apps/>
7. <https://devblogs.microsoft.com/aspnet/blazor-aspnet-webforms-ebook/>
8. <https://reactive.net/>

9. <https://reactive.net/blog/2021/05/sequences-linq-rx-reactor-part-05-remotable-expressions>
  10. <https://www.listennotes.com/podcasts/type-theory-forall/9-logic-and-proof-theory-kT-OME0lwsx/>
- 

### 15.5.2 Stream Processing with Apache Flink (2021-06-28 06:29)

[1]Stream Processing with Apache Flink: Fundamentals, Implementation and Operation of Streaming Applications by Fabian Hueske and Vasiliki Kalavri

I absolutely loved this book. I'd previously done a lot of reading about Streaming Systems, and after all of that theory it was good to see details about a concrete implementation.

The book starts with a good introductory chapter on stateful stream processing, talking about the different types of data processing and how systems evolved towards stream processing. The next chapter is on stream processing fundamentals and discusses dataflow graphs and the different semantics around time - event time and processing time. There is also a discussion on watermarks, which are required to allow the system to close windows and push results further through the pipeline.

The next chapter gives an overview of the Flink architecture, walking through the various components from the job manager to the various task runners. There is a good discussion of state and the consistent checkpointing mechanism that Flink offers as a way of allowing the stream to restart if something breaks in the pipeline. These processing streams are designed to run continuously for months, so we need a way to restart and get back to the current time if something goes wrong.

There's a chapter on setting up a Flink development environment, with the authors showing you how to run a small example Flink application. After that we really get into the implementation. There is a chapter describing the DataStream API which talks about transformations on the streams, how the streams can be executed in parallel and how keys (which are used for partitioning) and rich functions can be defined. Rich functions offer a start up and shutdown action as well as the standard method for processing data values as they pass through the stream. The authors then cover time-based operators and window based operators, which also leads to a discussion of timers (used for example in session windows where the window will shut if there is a sufficient time difference between some elements) and how the system handles late events (which arrive after the watermark has progressed).

There is more detail about implementation in the next chapters on stateful operators, which show how you'd write your own operators, which need to interact with the checkpointing

mechanism. The next chapter covers reading and writing to external systems - in order to achieve exactly once semantics we need to have input sources that can be rewound, and transaction sinks to allow the data to be committed exactly once. Other guarantees are available depending on the sources and sinks, and there is a good discussion to illustrate this. This chapter talks about two common sources, files and Kafka which both have desirable properties like restartability.

The last two chapters talk about the operational side of things, such as how you should set up a cluster for running Flink, and how you manage it for running long term.

The book is really good. Looking at the lower levels of implementation really helped me understand streaming systems in a lot more detail, and the book's many examples make it quite clear what is happening.

[And if you are interested in how to get started with Kafka, see [2]this blog post]

1. [https://www.amazon.co.uk/Stream-Processing-Apache-Fabian-Hueske/dp/149197429X/ref=sxts\\_rp\\_s1\\_0?cv\\_ct\\_cx=stream+processing+with+apache+flink&dchild=1&keywords=stream+processing+with+apache+flink](https://www.amazon.co.uk/Stream-Processing-Apache-Fabian-Hueske/dp/149197429X/ref=sxts_rp_s1_0?cv_ct_cx=stream+processing+with+apache+flink&dchild=1&keywords=stream+processing+with+apache+flink)
  2. <https://blog.tinybird.co/2021/06/25/startng-with-kafka/>
- 

### 15.5.3 A few more books (2021-06-30 06:23)

[1]The Art of Immutable Architecture: Theory and Practice of Data Management in Distributed Systems by Michael L. Perry

To be honest I'm not sure what I thought about this book. It contains material on distributed systems and the difficulty of achieving consensus, and also explains how immutability helps us work around some of the problems. In particular there is a good discussion of using [2]CRDTs. The author then goes on to describe his technique of [3]historical modelling which brings together immutability and eventual consistency to give us a way to architect distributed systems.

#### [4]Staff Engineer: Leadership beyond the Management Track by Will Larson

I really enjoyed the first half of this book which describes the Staff Engineer role, in an attempt to define what it is. There is lots of good advice in this section about how to do the role well - how to influence, how to present ideas and how to get the role in the first place.

The second half of the book is a series of interviews with Staff Engineers from a number of companies, where the various people are asked about what the role means in their company and how they got the role. It also asks them what an average day looks like. I must admit that I found this part of the book hard work and gave up reading the stories.

There's also a large set of references to articles and blog posts which a Staff Engineer should read. I found this list really useful and have been working my way through the references.

#### [5]Category Theory for Programmers by Bartosz Milewski

I worked my way through this book (yet again) and I absolutely love the book. The author explains the material really well, giving the motivations behind the various concepts and going into a good discussion of how they relate to programming. It won't be long before I read it all again.

This book is put together from a series of blog posts, and the material is also available as an [6]online set of lectures.

1. [https://www.amazon.co.uk/Art-Immutable-Architecture-Management-Distributed/dp/1484259548/ref=cm\\_cr\\_arp\\_d\\_product\\_top?ie=UTF8](https://www.amazon.co.uk/Art-Immutable-Architecture-Management-Distributed/dp/1484259548/ref=cm_cr_arp_d_product_top?ie=UTF8)
2. [https://en.wikipedia.org/wiki/Conflict-free\\_replicated\\_data\\_type](https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type)
3. <https://github.com/michalporeba/pof>
4. [https://www.amazon.co.uk/Staff-Engineer-Leadership-beyond-management/dp/1736417916/ref=sr\\_1\\_1?dchild=1&keywords=staff+engineer&qid=1624803547&s=books&sr=1-1](https://www.amazon.co.uk/Staff-Engineer-Leadership-beyond-management/dp/1736417916/ref=sr_1_1?dchild=1&keywords=staff+engineer&qid=1624803547&s=books&sr=1-1)
5. <https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/>
6. [https://www.youtube.com/watch?v=I8LbkfSSR58&list=PLbgaMIhjbEnaH\\_LTkxLI7FMa2HsnaW\\_M\\_](https://www.youtube.com/watch?v=I8LbkfSSR58&list=PLbgaMIhjbEnaH_LTkxLI7FMa2HsnaW_M_)

## 15.6 September

### 15.6.1 Develop C# as if it were a dynamic language? (2021-09-13 07:20)

I've always been a massive fan of Edit and Continue, which used to be implemented as a method inside the CLR's debugging APIs which allowed you to patch an assembly to change the metadata and IL instructions. In the past it was hard to figure out how to use it, but now Roslyn lets us look into the compiler source, it is much easier to see how the metadata and IL patch files are generated.

Anyway, this whole feature has been rebranded as "Hot Reload" and the API is available as standard C# library method in order to support "dotnet watch". I did a lightning talk at work at how this works under the covers, and slides are [1]here. The Github repository also contains some examples about the new experience.

---

1. <https://github.com/clivetong/Play/blob/master/EnCForTheWin/HotReloadAndEditAndContinue.pptx>

### 15.6.2 And a summer of reading (2021-09-15 06:49)

It was great that the lockdown came to an end over the summer, and it was great to use the time away to read through some of the books that had queued up.

[1]Reinforcement Learning: An Introduction by Richard S Sutton and Andrew G Barto

This is quite a large academic book, but a great read if you have the time. It starts simple and works its way through the various models for reinforcement learnings, covering all of the variations. It sets out a model of policies and value functions, and then looks at the various algorithms that can be used to allow a computer to learn for itself by playing multiple scenarios. I liked the way it started with solvable Markov Decision Processes, and then generalized to the point where the algorithm is actually learning the value functions (which are implemented via a neural network). I may not have had the time to understand everything in detail, but it gave a great overview of how the magic happens.

[2]A Philosophy of Software Design by John Ousterhout

To me this book was full of great advice and well reasoned arguments as to why the advice should be followed. I came across the book when reading a blog post that criticized "Clean Code", which was an always recommended book in the past, but which had a few things that always seemed wrong to me (and that book doesn't really give arguments why the advice is good).I think that's why I think this book is so good - the author discusses trade-offs and doesn't come across as advocating some perfect solution.

### [3]Web Development with Blazor by Jimmy Engstrom

This book introduces you to Blazor by developing client-side and server-side Blazor applications that implement a blogging platform. Personally I found the text a little hard to follow as the type setting often made it hard to follow what the text was actually describing. However, it was a good basic introduction to Blazor and the author can talk about his experiences developing Blazor application.

### [4]Androids: The Team that build the Android Operating System by Chet Hasse

This book covers the history of Android, from the days before it was acquired by Google though to just after the first few public releases. I liked the history and the explanation of how some of the features like Intents came about.

### [5]Don't be evil The Case Against Big Tech by Rana Foroohar

I feel uneasy about how big tech is taking people's data and using it for strange purposes. This book outlines why big tech is dangerous, from the ability to influence elections to the use of personal information in different scenarios. The book is well worth a read for understanding how we are at a turning point where more laws may be needed to protect the democracy that we have.

### [6]Software Architecture Patterns by Mark Richards

This was a free download from O'Reilly and gives some details about 5 architecture styles, from layered architectures to Microservices.

### [7]Software Engineering at Google: Lessons Learned from Programming Over Time

An amazing book rich, with advice about team processes and implementation at a company like Google. I guarantee that you will see something from your daily engineering job in a different light after reading this book.

1. [https://www.amazon.co.uk/Reinforcement-Learning-Introduction-Richard-Sutton/dp/0262039249/ref=sr\\_1\\_1?crid=4C17BZ4U0JXE&dchild=1&keywords=reinforcement+learning+](https://www.amazon.co.uk/Reinforcement-Learning-Introduction-Richard-Sutton/dp/0262039249/ref=sr_1_1?crid=4C17BZ4U0JXE&dchild=1&keywords=reinforcement+learning+)
  2. [https://www.amazon.co.uk/Philosophy-Software-Design-2nd/dp/173210221X/ref=sr\\_1\\_1?crid=1SCC670FIKD3T&dchild=1&keywords=a+philosophy+of+software+design%2C+2nd+edi](https://www.amazon.co.uk/Philosophy-Software-Design-2nd/dp/173210221X/ref=sr_1_1?crid=1SCC670FIKD3T&dchild=1&keywords=a+philosophy+of+software+design%2C+2nd+edi)
  3. [https://www.amazon.co.uk/Web-Development-Blazor-.NET-hands/dp/1800208723/ref=sr\\_1\\_1?dchild=1&keywords=web+development+with+blazor&qid=1631446346&sr=8-1](https://www.amazon.co.uk/Web-Development-Blazor-.NET-hands/dp/1800208723/ref=sr_1_1?dchild=1&keywords=web+development+with+blazor&qid=1631446346&sr=8-1)
  4. [https://www.amazon.co.uk/Androids-Built-Android-Operating-System/dp/1737354810/ref=sr\\_1\\_1?crid=Z7V39ESKLNZC&dchild=1&keywords=androids+the+team+that+built+the+a](https://www.amazon.co.uk/Androids-Built-Android-Operating-System/dp/1737354810/ref=sr_1_1?crid=Z7V39ESKLNZC&dchild=1&keywords=androids+the+team+that+built+the+a)
  5. [https://www.amazon.co.uk/Dont-Be-Evil-Case-Against/dp/0141991089/ref=sr\\_1\\_1?crid=3NL9MYVS0X05F&dchild=1&keywords=dont+be+evil+the+case+against+big+tech+by+rana+](https://www.amazon.co.uk/Dont-Be-Evil-Case-Against/dp/0141991089/ref=sr_1_1?crid=3NL9MYVS0X05F&dchild=1&keywords=dont+be+evil+the+case+against+big+tech+by+rana+)
  6. <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/>
  7. [https://www.amazon.co.uk/Software-Engineering-Google-Lessons-Programming/dp/B08VKJXVHK/ref=sr\\_1\\_1?dchild=1&keywords=software+engineering+at+google&qid=163144665](https://www.amazon.co.uk/Software-Engineering-Google-Lessons-Programming/dp/B08VKJXVHK/ref=sr_1_1?dchild=1&keywords=software+engineering+at+google&qid=163144665)
- 

## 15.7 October

### 15.7.1 Some Kafka and some algorithms (2021-10-18 06:02)

[1]Mastering Kafka Streams and ksqlDB: Building real-time data systems by example by Mitch Seymour

I'm fascinated by the duality between streams and tables, the mixing of data at rest and data that flows through a pipeline. This was a great read to understand how these notions are realised inside Kafka. The book consists of a number of chapters that introduce the concepts and then use the concepts in a set of examples, which really brings out the understanding.  
[2]The examples are also available on GitHub so you can try them out yourself.

The book starts with a rapid introduction to Kafka, going over topics and partitions, clusters and brokers. It then goes through the concepts behind Kafka streams, looking at how they relate to scalability, reliability and maintainability. There is some discussion of processor topologies and there is a look at the differences between the high level DSL and the lower level processor API. This chapter also looks at the stream table duality, looking at KStream and KTable. The next two chapters look at stateless and stateful processing, where the examples concern processing a twitter stream and a video game leaderboard. This is followed by a chapter on Windows and Time, and then a chapter on advanced state management which includes a description of rebalancing. The last chapter in this section of the book, looks in

more detail at the Processor Api.

Section two of the book looks at ksqlDB. In this section we use an extended Sql to express our queries. The first chapter looks at how ksqlDB differs from a standard Sql database, and is followed by a chapter on connectors, which is the interface you can use to get data from other systems and move it into this world. There are then two chapters on stream processing using ksql.

The last section of the book contains a single chapter on Testing, Monitoring and Deployment.

This book is a really good read, and gives you a really good mental model of what is happening as Kafka runs.

### [3]Algorithms and Data Structures in Action by Marcello La Rocca

The book does a good job in the introduction of explaining why data structures matter. It then has three sections: improving over basic data structures, multidimensional queries, planar graphs and minimal crossing numbers. It's the choice of data structures that I really enjoyed - they are often extensions of the basic data structures we've all come across before, or more modern data structures applicable to large scale computing.

The basic data structures section covers priority heaps (d-way heaps), treaps, bloom filters, disjoint sets, tries and radix tries, and finally studies how to implement an LRUI cache.

The multidimensional queries section looks at nearest neighbours, multidimensional data indexing using K-d trees, approximate nearest neighbours for image retrieval, applications of nearest neighbour, clustering and map reduce and how it can be used to implement clustering.

The third section, on planar graphs, looks at minimum distance paths, graph embeddings, and uses this to study three ways of getting approximate solutions: gradient descent, simulated annealing and genetic algorithms. I thought this third section was really good, using the earlier problem to inspire the need for the approximate search algorithms.

The book contains working code that shows you how to implement the algorithms, which makes the explanations really clear, and all of the algorithms that the author chose are really interesting.

#### [4]ADAPTIVE OPTIMIZATION FOR SELF: RECONCILING HIGH PERFORMANCE WITH EXPLORATORY PROGRAMMING by Urs Holze

This is a Phd thesis on the implementation of adaptive (JIT) compilation for the SELF language. It's full of great ideas like polymorphic inline caches and dynamic de-optimization and on-stack replacement, which where all quite novel at the time. The thesis isn't too long, and contains great explanations and measurements to give you a great idea of the improvements.

1. [https://www.amazon.co.uk/Mastering-Kafka-Streams-ksqlDB-real-time/dp/1492062499/ref=sr\\_1\\_1?dchild=1&keywords=mastering+kafka+streams+and+ksqlDB&qid=1634470702&s](https://www.amazon.co.uk/Mastering-Kafka-Streams-ksqlDB-real-time/dp/1492062499/ref=sr_1_1?dchild=1&keywords=mastering+kafka+streams+and+ksqlDB&qid=1634470702&s)
  2. <https://github.com/mitch-seymour/mastering-kafka-streams-and-ksqlDB>
  3. [https://www.amazon.co.uk/Advanced-Algorithms-Structures-Marcello-Rocca/dp/1617295485/ref=sr\\_1\\_1?dchild=1&keywords=advanced+algorithms+and+data+structures&qid=16](https://www.amazon.co.uk/Advanced-Algorithms-Structures-Marcello-Rocca/dp/1617295485/ref=sr_1_1?dchild=1&keywords=advanced+algorithms+and+data+structures&qid=16)
  4. <http://i.stanford.edu/pub/cstr/reports/cs/tr/94/1520/CS-TR-94-1520.pdf>
- 

## 15.8 November

### 15.8.1 And some holiday reading (2021-11-07 07:47)

I just had a week of holiday, and it was good to catch up with some reading.

#### [1]Networking and Kubernetes: A Layered Approach by James Strong and Vallery Lancey

I absolutely loved this book, which I bought for knowledge about Kubernetes networking, but learned so much more - the book is about networking in general, and then later moves into specialising around Kubernetes. I loved this approach.

There are six chapters. The first is a general introduction to networking, both the history and the development of the OSI model. Chapter two moves into Linux networking, with sections on packet handling inside the kernel using netfilter/conntrack and high level routing via iptables, IPVS and eBPF. It also covers all the network troubleshooting tools like ping, traceroute and

openssl. Chapter three moves on to container networking basics after an introduction to control groups and namespaces. It covers the docker network model and goes in depth on container to container networking on the same host, and when one container is remote. Chapter four starts the dive into Kubernetes networking and looks at how the kubelet and kube-proxy handle the networking between the pods. Chapter five looks at the abstractions in Kubernetes, including EndPoints, NodePorts, ClusterIP, LoadBalancing and moving on to service meshes. Chapter six looks at the networking provided by the various cloud providers in their managed Kubernetes offerings.

The book links to a Github repository with lots examples that you can run in vagrant, and the text is full of examples of someone using the command line to illustrate the issues. Brilliant!

[2]Production Kubernetes: Building Successful Application Platforms by Josh Rosso, Rich Lander, Alexander Brand and John Harris

This is a really good book on the practical use of Kubernetes as a platform on which teams can run their applications. It is written by consultants and their experience really shines through. I learned lots about Kubernetes in the earlier chapters which talk about most of the features of Kubernetes from storage to service meshes. The later chapters then discuss the various ways that you can set things up to let other teams run their applications on top of Kubernetes and all of the associated things you need to think about from single- to multi-tenancy on the cluster to collection of logs and observability.

[3]Pro BASH Programming: Scripting the GNU/Linux Shell by Chris F.A Johnson and Jayant Varma

This book goes into BASH in a lot of detail and shows off lots of its advanced features,

[4]Terraform Up and Running: Writing Infrastructure as Code by Yevginiy Brikman

I've read this book before, but wanted to re-read as I'm using Terraform a lot more at work now. It's a good introduction to Terraform, starting at the very beginning, and with enough practical examples to explain the points the text makes.

[5]Team Topologies: Organising Business and Technology Teams for Fast Flow by Matthew Skelton and Manuel Pais

A very colourful book that discusses why you don't just want a set of similarly configured development teams in your company. The book introduces the four key types of team: stream-aligned, enabling, complicated subsystem and platform teams.

[6]Software Mistakes and Tradeoffs: How to make good programming decisions by Tomasz Lelek and Jon Skeet

This book consists of a set of 13 independent chapters that look at various software problems, and then looks at the tradeoffs in the possible solutions. Some chapters are on things like exceptions versus other patterns for handling errors, simplicity of an API against the cost of maintenance or consistency and atomicity in distributed systems, The chapters are interesting, and cover lower level and higher level issues.

I found the book quite hard to read, but it was worth the effort for some great gems of knowledge.

And finally a couple of type checking papers related to Haskell. [7]The first by Richard Eisenberg on making existentials easier to use in Haskell programs. [8]The second is on impredicativity and Haskell, making universally quantified types first class.

1. [https://www.amazon.co.uk/Kubernetes-Networking-Deep-James-Strong/dp/1492081655/ref=sr\\_1\\_1?crid=3VFI1FAM9NS78&keywords=networking+and+kubernetes&qid=1636200640&s](https://www.amazon.co.uk/Kubernetes-Networking-Deep-James-Strong/dp/1492081655/ref=sr_1_1?crid=3VFI1FAM9NS78&keywords=networking+and+kubernetes&qid=1636200640&s)
2. [https://www.amazon.co.uk/Production-Kubernetes-Successful-Application-Platforms/dp/1492092304/ref=sr\\_1\\_1?crid=31U21AR3Z90DR&keywords=production+kubernetes&qid=1](https://www.amazon.co.uk/Production-Kubernetes-Successful-Application-Platforms/dp/1492092304/ref=sr_1_1?crid=31U21AR3Z90DR&keywords=production+kubernetes&qid=1)
3. [https://www.amazon.co.uk/Pro-Bash-Programming-Scripting-Linux/dp/1484201221/ref=sr\\_1\\_3?crid=VK7UA02ZB003&keywords=pro+bash+programming&qid=1636202434&sprefix=pr](https://www.amazon.co.uk/Pro-Bash-Programming-Scripting-Linux/dp/1484201221/ref=sr_1_3?crid=VK7UA02ZB003&keywords=pro+bash+programming&qid=1636202434&sprefix=pr)
4. [https://www.amazon.co.uk/Terraform-Running-Writing-Infrastructure-Code/dp/1492046906/ref=sr\\_1\\_1?crid=1M2PRMJ62NJWU&keywords=terraform+up+and+running&qid=1636202](https://www.amazon.co.uk/Terraform-Running-Writing-Infrastructure-Code/dp/1492046906/ref=sr_1_1?crid=1M2PRMJ62NJWU&keywords=terraform+up+and+running&qid=1636202)
5. [https://www.amazon.co.uk/Team-Topologies-Organizing-Business-Technology/dp/1942788819/ref=sr\\_1\\_1?keywords=team+topologies&qid=1636202730&sprefix=team+topologfie](https://www.amazon.co.uk/Team-Topologies-Organizing-Business-Technology/dp/1942788819/ref=sr_1_1?keywords=team+topologies&qid=1636202730&sprefix=team+topologfie)
6. [https://www.amazon.co.uk/Software-Mistakes-Tradeoffs-Programming-Decisions/dp/1617299200/ref=sr\\_1\\_1?keywords=software+mistakes+and+tradeoffs&qid=1636203021&spref](https://www.amazon.co.uk/Software-Mistakes-Tradeoffs-Programming-Decisions/dp/1617299200/ref=sr_1_1?keywords=software+mistakes+and+tradeoffs&qid=1636203021&spref)
7. <https://richarde.dev/papers/2021/exists/exists.pdf>
8. <https://www.microsoft.com/en-us/research/publication/a-quick-look-at-impredicativity/>



# 2022

## 16.1 January

### 16.1.1 Static and Abstract? (2022-01-03 07:10)

Another lightning talk at work recently, on the experimental static abstract which has been added to the language to make it easier to define generic operations over the operators on the various numeric types in the language. I prepared no slides for this talk, but [1] instead used a file of the various examples which I uncommented part by part. This was made easier than in C # comment opening expressions /\* don't nest.

This whole addition works through interfaces and it will be interesting seeing how much more gets added to interfaces in the future - now that the CLR can be changed between releases, and hence we don't have to implement every language feature using code generation (levelling).

Another recent change to the CLR that catch my eye was [2] dynamic profile guided optimisation. Use different tiers of compilation, the first level being just a fast and unoptimised version of the code generator. Then run the code for a while and identify what it would be good to optimise more thoroughly, and once this has been identified sufficiently well, re-compile to more optimised code. It's a good idea, and one that Java has had for a long time. It offers the chance of fast start up, and highly optimised code as the application reaches a steady state - you may well identify lots of opportunities for inlining for example that you'd not think worthwhile from a more static view of the program.

1. <https://github.com/clivetong/Play/blob/master/StaticAbstract/StaticAbstract/Program.cs>
2. <https://gist.github.com/EgorBo/dc181796683da3d905a5295bfd3dd95b>

## 16.2 February

### 16.2.1 Dynamic and profile guided (2022-02-27 05:57)

Another [1]lightning talk at work, this time on Dynamic Profile Guided Optimization. The idea that you optimize your running program specifically for the data to which it is being applied, has always felt to me like the right way to do things. It's all very well optimizing for the general case, or collecting data on runs across some scenarios and using that to guide the compilation, but surely the best solution is to optimize hard for the data you've seen so far during the run, with the likely hypothesis that subsequent data will be like that too. .NET seems to have taken ages to catch up with other platforms like Java that have had this technology for some time. With the new tiered compilation model, this fits in quite nicely and with some initial work on OSR making it possible to optimize running functions, there must be many gains to be made in the future.

---

1. <https://github.com/clivetong/Play/tree/master/dynamiccpgo#readme>

### 16.2.2 Python for the win (2022-02-28 06:54)

I've been using Python a lot for personal programming projects. The REPL makes it fun to explore a problem space and the number of libraries is truly astounding.

[1]CPython Internals: Your Guide to the Python 3 Interpreter by Antony Shaw

I got interested in the internals of Python, after coming across the proposal by Sam Gross in [2]Multithreaded Python without the GIL to change the reference counting to avoid some of the locking that is currently required, and change the core to allow more than one thread to run Python at the same time. [3]The GIL has always been something that made Python look a lot less useful as a runtime, though admittedly, as the next book shows, people have had to come up with a mass of clever ideas to work around it.

This book was a great practical introduction to the internals of CPython. The first chapter gets you to check out the sources, and then build them using a number of development environments. The next chapters cover the grammar, the lexing and parsing and the resulting syntax trees, and then move on to the implementation of the compiler. The book then goes into some detail about the evaluation loop, including the stack frames and thread states, and then dives into chapters on memory management, parallelism and concurrency, and objects and types. The last chapters cover the standard library, the test suite and benchmarking/profiling/tracing.

The book goes into the right amount of detail, at times stepping us through the actions of the runtime so we understand what is happening. It also covers how you can use C modules from Python.

[4]High Performance Python: Practical Performant Programming for Humans by Micha Gorelick and Ian Ozsvald

This is a lovely book, full of techniques to improve the performance of Python code. There's an initial chapter giving an overview of the Python system, followed by a chapter on how to profile to find the bottlenecks. The authors then look at some common datatypes in the next chapters - list/tuples, dictionaries/sets and iterators/generators. Matrices and vectors are important datatypes in machine learning, and there is a chapter about the various libraries and tricks to get good performance. The next chapters cover the various ways to compile Python in to C code, including Cython and PyPy. Three chapters follow on scaling up to use the multiple cores on the machine - an in-depth look at AsyncIO, the multiprocessing module and clusters and job queues. It is quite amazing the ways people have developed to get around the GIL to make applications scale as you add more cores.

The book then has a chapter on how to use less RAM, including some techniques for approximating results to use less memory. The last chapter is a series of case studies on optimising Python applications in the real world.

This was a thoroughly interesting book, giving insights into the implementation as well as ways to work around the limitations. Some of the advice is general and is valid for all programming languages and runtimes.

1. [https://www.amazon.co.uk/CPython-Internals-Guide-Python-Interpreter/dp/1775093344/ref=sr\\_1\\_1?crid=B8RW6173AK81&keywords=cpython+internals&qid=1645957188&sprefix](https://www.amazon.co.uk/CPython-Internals-Guide-Python-Interpreter/dp/1775093344/ref=sr_1_1?crid=B8RW6173AK81&keywords=cpython+internals&qid=1645957188&sprefix)
2. <https://docs.google.com/document/d/18CXhDb1ygxg-YXNBJNzfzZsDFosB5e6BfnXLlejd910/edit>
3. <https://realpython.com/python-gil/>
4. [https://www.amazon.co.uk/High-Performance-Python-Performant-Programming/dp/1492055026/ref=sr\\_1\\_1?crid=3P1E93W3K9K3I&keywords=high+performance+python&qid=1645957](https://www.amazon.co.uk/High-Performance-Python-Performant-Programming/dp/1492055026/ref=sr_1_1?crid=3P1E93W3K9K3I&keywords=high+performance+python&qid=1645957)

## 16.3 March

### 16.3.1 Getting to grips with Quantum Computing (2022-03-14 06:55)

[1]Learn Quantum Computing with Python and Q # by Sarah Kaiser and Christopher Grande

This is an absolutely fantastic book.

Quantum computers might well be the next step in speed and computing power, and I wanted an introduction that would teach me how it all links together. This book comes at the subject from a very practical perspective, and as I've always been a great believer that you don't really understand something until you implement it yourself, and since I can't afford to build a quantum computer, building a simulator like you do in the early chapters of this book is a great way to see what is happening.

The book starts with an introductory chapter on quantum computers and covers how they different from classical computers. Then things start getting interesting. The book looks at Qubits, and how they are the building block of quantum algorithms. We quickly start writing a simulator in Python. The whole time the authors push the linear algebra side of things, showing us the various transforms (using Real numbers to start with). As initial targets of the first version of the simulator, we look at quantum key distribution, and then get into non-local games which requires us to use multiple qubits. Part 1 finishes with simulation of quantum teleportation and entanglement. We start using NumPy and move to using the [2]QuTiP library as things become more complicated to simulate.

Part 2 of the book moves on to using [3]Q # as the implementation language - you can find many other examples [4]here. The book does a good job of covering the Q # language, and then talks through some quantum algorithms like Deutsch-Jozsa to convert an oracle into a quantum operation. Solving various games is used as the motivator for some of the examples - I think that was a fun way to set up a problem, but it did take a little getting used to.

Part 3 of the book looks at some applications of a Quantum computer. Applications in chemistry by simulating the Hamiltonian for a molecule, applications in search by way of Grover's algorithm, and then integer factoring using Shor's algorithm. The authors walk through the algorithms leaving you with a good understanding of how they work.

I'll say it again - this is a really great book for getting up to speed on Quantum computing, and I enjoyed it very much.

1. [https://www.amazon.co.uk/Learn-Quantum-Computing-Python-hands/dp/1617296139/ref=sxts\\_rp\\_s1\\_0?cv\\_ct\\_cx=learn+quantum+computing+with+python+and+q%23&keywords=learn](https://www.amazon.co.uk/Learn-Quantum-Computing-Python-hands/dp/1617296139/ref=sxts_rp_s1_0?cv_ct_cx=learn+quantum+computing+with+python+and+q%23&keywords=learn)
  2. <https://qutip.org/>
  3. <https://docs.microsoft.com/en-gb/azure/quantum/>
  4. <https://github.com/microsoft/Quantum/tree/main/samples>
- 

### 16.3.2 Microsoft Refresh (2022-03-20 06:37)

I was interested in the details behind Microsoft's move into cloud computing, and was hoping that these two books would give me some of the story. Unfortunately, neither did.

#### [1]Hit Refresh by Satya Nadella

This book didn't really work for me as an autobiography, and didn't work as a story of how Microsoft was rebooted. None of the discussion about the future, via Artificial Intelligence and Quantum Computing really went into anything like the detail I was looking for either.

#### [2]The Microsoft Story by Dan Good

The book did a good job of describing the Microsoft Story in some detail, and I must admit that I hadn't heard some of the stories before. Again, not a bad read, but it stops short of the era of Microsoft that I wanted details about.

1. [https://www.amazon.co.uk/Hit-Refresh-Memoir-Microsofts-CEO/dp/0008247692/ref=sr\\_1\\_1?keywords=hit+refresh+by+satya+nadella&qid=1647703879&sprefix=hit+refres%2Cap](https://www.amazon.co.uk/Hit-Refresh-Memoir-Microsofts-CEO/dp/0008247692/ref=sr_1_1?keywords=hit+refresh+by+satya+nadella&qid=1647703879&sprefix=hit+refres%2Cap)
  2. [https://www.amazon.co.uk/dp/1400223903/ref=sspa\\_dk\\_detail\\_1?psc=1&pd\\_rd\\_i=1400223903&pd\\_rd\\_w=aWzqm&pf\\_rd\\_p=828203ef-618e-4303-a028-460d6b615038&pd\\_rd\\_wg=y7BoR&p](https://www.amazon.co.uk/dp/1400223903/ref=sspa_dk_detail_1?psc=1&pd_rd_i=1400223903&pd_rd_w=aWzqm&pf_rd_p=828203ef-618e-4303-a028-460d6b615038&pd_rd_wg=y7BoR&p)
- 

### 16.3.3 Secure your containers (2022-03-21 06:51)

#### [1]Container Security by Liz Rice

I really enjoyed this book which takes you through container security. It starts with a good introduction on container security threats, including security boundaries and the issue of

multi-tenancy. After that, we look at some of the security mechanisms that Linux offers - systems calls and permissions including capabilities and features like setuid and setgid. This is followed by a good chapter on cgroups and how they can be used to control resource usage for a group of processes. The author then goes through the means that Docker uses to isolate its containers - the various Linux namespaces, the how the root is changed to isolate the container's filesystem from the outside world.

There's a great chapter on virtual machines, with a discussion of the difference between Type I and Type II VMs and how they work. This discusses how they differ from containers. This is followed by a chapter on container images - in particular what they contain in the various layers. We then look at what vulnerabilities images can contain, and how we can use scanning to detect them. Linux mechanisms to tighten container isolation are then covered - tools like Seccomp, AppArmour, SELinux and gVisor, and then moving away from containers to Kata Containers, FireCracker and Unikernels.

Chapters follow on breaking container isolation which includes running rootless containers, and the use of the -privileged flag. Then a chapter on container network security, one on TLS, how to pass secrets securely to containers, and container runtime protection using profiles.

This is quite a short book which is packed with loads of information. I really enjoyed the "build your own docker" chapters to build up knowledge about the way containers work, and the explanations are really great.

1. [https://www.amazon.co.uk/Container-Security-Fundamental-Containerized-Applications/dp/1492056707/ref=sr\\_1\\_1?crid=1H3GT26V9M9XX&keywords=container+security&qid=1](https://www.amazon.co.uk/Container-Security-Fundamental-Containerized-Applications/dp/1492056707/ref=sr_1_1?crid=1H3GT26V9M9XX&keywords=container+security&qid=1)
- 

#### **16.3.4 What are the chances of that? (2022-03-23 06:55)**

I put together [1]a lightning talk on Quantum Computing. I've recently got into Quantum Computing and find it a fascinating subject. There was, of course, too much material to fit in to the talk - I was aiming to try and cover all of the misconceptions about Quantum Computing, but in the end there was just too much to cover to make this really the subject of the talk.

1. <https://github.com/clivetong/Play/blob/master/StateOfQuantum/README.md>
-

### **16.3.5 Ceph in detail (2022-03-28 06:21)**

[1]Mastering Ceph: Infrastructure storage solutions with the latest Ceph release by Nick Fisk

We've started using Ceph, or more specifically [2]Rook-Ceph as a storage solution for some Kubernetes work that we are doing. We started treating Ceph itself as a bit of a black box, but as we've moved more towards the performance side of things, it has become apparent that we need to have a better understanding of what is happening under the covers.

This book is a little old, but does give a good understanding of Ceph. There are general overview chapters about Ceph, and what its unique selling points are, and there are some Vagrant scripts that allow you to do a Ceph deployment yourself, in a couple of different scenarios. I loved the way that the book gets into the details by getting you to run various command line commands to understand what is going on. The second section of the book is about operating and tuning Ceph, and gave us many insights as to what was happening beneath the covers, and Section 3 on troubleshooting is going to come in useful as we deploy our K8s application to more customer sites.

For us, some parts of the book weren't particularly relevant, as we won't be using CephFS for example, but the book is definitely worth a read if you want to see across the whole of the Ceph system. We still have [3]many questions about how Ceph is driven via Rook in the K8s world, but this was a great way to understand the storage architecture.

1. [https://www.amazon.co.uk/Mastering-Ceph-Infrastructure-storage-solutions/dp/1789610702/ref=sr\\_1\\_1?crid=3BXFI3TABTZG1&keywords=mastering+ceph&qid=1648390874&spre](https://www.amazon.co.uk/Mastering-Ceph-Infrastructure-storage-solutions/dp/1789610702/ref=sr_1_1?crid=3BXFI3TABTZG1&keywords=mastering+ceph&qid=1648390874&spre)
  2. <https://github.com/rook/rook>
  3. <https://rook-io.slack.com/archives/CK9CF5H2R/p1645513239976989>
- 

## **16.4 April**

### **16.4.1 Software Architecture: It's hard! (2022-04-04 06:27)**

[1]Software Architecture: The Hard Parts - Modern Trade-off Analysis for Distributed Architectures by Neal Ford, Mark Richard, Pramod Sadalage and Zhamak Dehghani

This is a good book about the hard architectural decisions that a team face when breaking a monolith into a micro-services architecture. It covers a lot of very interesting topics like

architectural fitness functions and is full of lots of practical advice on how to weigh up the pros and cons. I'm not sure I'm the greatest fan of the storyline which follows a team as it splits up a support application from monolith to microservices, but all of the material that hangs off the chapters of this story is really good. The book covers everything from choice of database type to service granularity to the different types of transactional sagas to choices between choreography and orchestration. It's a great read.

1. [https://www.amazon.co.uk/Software-Architecture-Modern-Tradeoff-Analysis/dp/1492086894/ref=sr\\_1\\_1?keywords=software+architecture+the+hard+parts&qid=1648981026&sp](https://www.amazon.co.uk/Software-Architecture-Modern-Tradeoff-Analysis/dp/1492086894/ref=sr_1_1?keywords=software+architecture+the+hard+parts&qid=1648981026&sp)

---

## 16.5 May

### 16.5.1 Another set of books (2022-05-16 06:25)

[1]React Hooks In Action: With Suspense and Concurrent Mode by John Larsen

I'd been away from React for a while and noticed that React 18 has just been released. This reminded me of the talk about Concurrent mode (the ability for a render to be interrupted and then restarted if a higher priority task comes along) and Suspense (seamlessly handle data loading notification in the GUI) a while back when I last did React in my day job.

This book is a great guide to modern React. It discusses what hooks are and how they work, and then walks through an example application that uses a number of the standard React hooks. The application feels really realistic and the author explains the needs for the hooks and how to use them really well - useState, useReducer (part of the eternal question of how to handle component state across the application), useEffect and useRef. There is an initial chapter on managing application state which introduces useCallback and then useMemo.

This section then ends with chapters on managing state with the Context API, creating your own hooks (by putting together existing hooks, and at the same time respecting the Rules of Hooks), and then a discussion of some of the available third party hooks (and React Query).

The shorter part 2 of the book covers Suspense and Concurrent Mode. The former is introduced in a chapter that covers lazy loading and error boundaries, and then there is a chapter on data fetching and how it interacts with Suspense. The last chapter looks at Concurrent Mode and the future of React.

The book is great - it is just the right length and covers the material in a way that answers most of the first level questions. Highly recommended!

[2]Learning GraphQL - Declarative Data Fetching for Modern Web Apps by Eve Porcello and Alex Banks

Using REST to fetch data from a complicated data model feels increasingly like hard work for the client. There are often lots of times when you fetch an object and then need to issue requests for each of the children, wiring the navigation of the object graph by doing a fetch for each edge. It has often felt to me like a single request with more specifics about the navigation of the graph to the desired properties was the way to go and things like OData and GraphQL offer this.

This book is a short introduction to GraphQL. It starts by describing where GraphQL fits in, with a discussion of the issues with REST. There's a very brief introduction to graph theory and then a chapter on the GraphQL language - this covers fragments, mutations and subscriptions (GraphQL is not just for reading data but allows applications to write data, and also allows you to keep listening and be informed when the data changes).

There is a chapter on schemas, and then an example of how to implement a schema using Apollo Server, an open source GraphQL library. The book finishes with chapters on GraphQL clients (particularly Apollo Client) with some focus on their caches. The last chapter covers GraphQL in the real world, discussing things like timeouts and security, as well as community resources you can access.

The book is short, but provides all the information that you need.

1. [https://www.amazon.co.uk/React-Hooks-Action-Suspense-Concurrent/dp/1617297631/ref=sr\\_1\\_1?crid=3E3GAJYUXEQ2U&keywords=react+hooks+in+action&qid=1652602881&srefi](https://www.amazon.co.uk/React-Hooks-Action-Suspense-Concurrent/dp/1617297631/ref=sr_1_1?crid=3E3GAJYUXEQ2U&keywords=react+hooks+in+action&qid=1652602881&srefi)
2. [https://www.amazon.co.uk/Learning-GraphQL-Declarative-Fetching-Modern/dp/1492030716/ref=sr\\_1\\_1?crid=2YAZ0J0QXE897&keywords=learning+GraphQL&qid=1652602948&sref](https://www.amazon.co.uk/Learning-GraphQL-Declarative-Fetching-Modern/dp/1492030716/ref=sr_1_1?crid=2YAZ0J0QXE897&keywords=learning+GraphQL&qid=1652602948&sref)

---

### 16.5.2 And yet more (2022-05-18 06:17)

[1]Programming Microsoft Azure Service Fabric by Haishi Bai

I've been doing a lot of Kubernetes lately, and since that has a Google heritage from systems like [2]Borg, I thought it was probably time to have another quick look at Service Fabric which was/is the base layer for many Microsoft services. Unsurprisingly, the concepts in Service Fabric are every much like those you find in other systems, though it felt to me that there was more of an emphasis on sharding and actor patterns - I think that some of these ideas have been taken across to Kubernetes with [3]Microsoft's DAPR. In Service Fabric the services can be stateful or stateless, and there are the usual mechanisms for rolling out changes across a cluster, concepts of scalability and performance, and availability and reliability.

The later chapters on Linux containers and advanced concepts (serverless and artificial intelligence) felt like standard expected things these days, but were probably innovative when the first version of the book was written.

[4]Linux Kernel Programming: A Comprehensive guide to kernel internals, writing kernel modules and kernel synchronization by Kaiwan N Billimoria

This is a big thick book on the Linux kernel and how to write modules to plug into the kernel. As a way to write these kernel modules, it covers the memory management inside the kernel and the way it handles threads in great detail. For the former, you'll get to understand the mappings of virtual memory and the page tables, on the way to understanding where your modules should allocate their memory. For the latter, you'll get a great understanding of the Linux tasks and how they are scheduled, and the locks you will need to use to make your code race free.

The book is really practical, with instructions on how to set up virtual box to safely set up many of the kernel programming exercises.

I absolutely loved this book, and will be spending time working on the examples to increase my understanding.

[5]Designing Microservices Platforms with NATS by Chanaka Fernando

We've been using NATS as a message bus at work in an application that I have inherited, so I wanted to learn a little more about it. This book covers the NATS application in its early chapters but also covers the ideas behind Microservice architectures too. It was an interesting read.

1. [https://www.amazon.co.uk/Programming-Microsoft-Service-Developer-Reference/dp/1509307095/ref=sr\\_1\\_1?crid=2GBXANRD7G43K&keywords=Programming+Microsoft+Azure+Serv](https://www.amazon.co.uk/Programming-Microsoft-Service-Developer-Reference/dp/1509307095/ref=sr_1_1?crid=2GBXANRD7G43K&keywords=Programming+Microsoft+Azure+Serv)
  2. [https://en.wikipedia.org/wiki/Borg\\_\(cluster\\_manager\)](https://en.wikipedia.org/wiki/Borg_(cluster_manager))
  3. <https://dapr.io/>
  4. [https://www.amazon.co.uk/Linux-Kernel-Development-Cookbook-programming/dp/178995343X/ref=sr\\_1\\_1?crid=15J3RCW7FBGB0&keywords=Linux+Kernel+Programming+by+Kaiwan+N](https://www.amazon.co.uk/Linux-Kernel-Development-Cookbook-programming/dp/178995343X/ref=sr_1_1?crid=15J3RCW7FBGB0&keywords=Linux+Kernel+Programming+by+Kaiwan+N)
  5. [https://www.amazon.co.uk/Designing-Microservices-Platforms-NATS-microservices-ebook/dp/B09CQ4T6QK/ref=sr\\_1\\_1?crid=1H4T6YFK6J3ZE&keywords=Designing+Microservices](https://www.amazon.co.uk/Designing-Microservices-Platforms-NATS-microservices-ebook/dp/B09CQ4T6QK/ref=sr_1_1?crid=1H4T6YFK6J3ZE&keywords=Designing+Microservices)
- 

### 16.5.3 Distributed system distilled (2022-05-23 06:36)

[1]Understanding Distributed Systems: What every developer should know about large distributed applications by Roberto Vitillo

This is, by far, the best book I have ever read on distributed systems. It's about 250 pages of information starting with communication via HTTPS and discoverability, coordination via distributed notions of time, replication and distributed transactions and their non-transactional equivalents, scalability via functional decomposition and partitioning, resiliency via an analysis of failure modes, both upstream and downstream, and even has a final chapter on testing and operations which converts monitoring and alerting. It goes into detail about all of the items, putting together just the right level of information about the concepts - things like retry amplification are covered in a couple of paragraphs of great explanation, and ideas like chain replication where just explained brilliantly.

An absolutely fabulous book that every developer should read!

[I should also mention that if you are interested in Logic and application, [2]there's a great multi-hour episode of the [3]Type Theory Forall podcast that talks about the context of Goedel's incompleteness theorems and a lot more]

1. [https://www.amazon.co.uk/Understanding-Distributed-Systems-Second-applications/dp/1838430210/ref=sr\\_1\\_1?crid=27IV7Y2YLZQM5&keywords=understanding+distributed+sy](https://www.amazon.co.uk/Understanding-Distributed-Systems-Second-applications/dp/1838430210/ref=sr_1_1?crid=27IV7Y2YLZQM5&keywords=understanding+distributed+sy)
  2. [https://www.typetheoryforall.com/2022/05/19/18-Godel-Incompleteness-Theorems-\(Cody-Roux\).html#93109f49](https://www.typetheoryforall.com/2022/05/19/18-Godel-Incompleteness-Theorems-(Cody-Roux).html#93109f49)
  3. <https://www.typetheoryforall.com/>
-

## 16.6 July

### 16.6.1 Some C# odds and ends (2022-07-04 06:38)

[1]I've been taking some Microsoft Exams lately, but have finally had time to look through a long set of blog posts that I have accumulated. Amongst them were a few interesting posts related to C #, so I thought I'd list them here.

[2]This is the design document for on-stack replacement, which has been added to the CLR in recent releases (and is still being improved). On-stack replacement has enabled better tier compilation - you start with a basic compilation of a given function, watch for how it is being used (for example, noticing the common paths through it), and then spent time compiling a more specific optimised version for the arguments that are being passed at runtime. This DPGO (dynamic profile guided optimisation) is great, and it is easy to make new invocations of a method use the more specific version, but if you have on stack instances that are, say, looping, you'd also like to transition them to using the new code. OSR lets you do this, changing the stack frames of executing versions of the old version of the code, to make them use the new version. This has enabled the CLR to use tiered compilation in more scenarios, and allowed methods to specialize for the data that the current run of the program is being passed (compared to PGO - profile guided optimization - which uses examples to specialize, though let's hope you profiled using the right examples for your current run). It's a great technology, and one that the JVM has had for a long time.

C # has emphasised low level performance and the avoidance of garbage collection in recent releases. In particular, stack allocated structs were added a while back. [3]This proposal attempts to tidy up the ref struct and other added struct types, some of which are used in the implementation of Spans.

This post discusses [4]the improvements in the C # library support of regular expressions and this post feels like a blast from the past as [5]it discusses using COM from C #. While we are talking about runtimes,[6] this post discusses how Go and the CLR differ, and [7]this paper discusses some recent attempts to figure out how much garbage collection in a language costs you.

Some other good posts included:[8] a list of the common machine learning algorithms

1. <https://www.linkedin.com/in/clivetong/>
2. <https://github.com/dotnet/runtime/blob/main/docs/design/features/OnStackReplacement.md>
3. <https://github.com/dotnet/csharplang/blob/main/proposals/csharp-7.2/span-safety.md>
4. <https://devblogs.microsoft.com/dotnet/regular-expression-improvements-in-dotnet-7/>
5. <https://devblogs.microsoft.com/ifdef-windows/the-journey-of-moving-from-cpp-winrt-to-csharp-in-the-microsoft-store/>

6. <https://medium.com/servicetitan-engineering/go-vs-c-part-3-compiler-runtime-type-system-modules-and-everything-else-faa423dddb34>
  7. <https://users.cecs.anu.edu.au/~steveb/pubs/papers/lbo-ispass-2022.pdf>
  8. <https://read.deeplearning.ai/the-batch/issue-146/>
- 

### **16.6.2 Inside Facebook, and API security (2022-07-11 06:20)**

Two more books off the pile.

[1]An Ugly Truth: Inside Facebook's Battle for Domination by Sheera Frenkel and Cecilia Kang

This gives a journalist's view of the fall of Facebook, from the darling of the tech industry to a company mired in controversy over various scandals. It's an interesting read, and the many interviews allow the authors to give very in-depth details on the various scandals and how Facebook tried to handle them - from spin to political meetings of various kinds.

[2]Advanced Api Security: OAuth 2.0 and Beyond by Prabath Siriwardena

This book covers OAuth 2 and its related profiles, which allow you to expose Enterprise APIs in a safe way so that they can be targeted by web applications, single page applications native mobile and browser-less applications. The author covers the whole subject area well with code that demonstrates some of the parts of the protocols and designs. There are also many good appendixes where the author covers the evolution of identity delegation on the web, as well as many other facets of this technology.

1. [https://www.amazon.co.uk/Ugly-Truth-Inside-Facebooks-Domination/dp/0349144060/ref=tmm\\_pap\\_swatch\\_0?\\_encoding=UTF8&qid=1657455048&sr=8-1](https://www.amazon.co.uk/Ugly-Truth-Inside-Facebooks-Domination/dp/0349144060/ref=tmm_pap_swatch_0?_encoding=UTF8&qid=1657455048&sr=8-1)
  2. [https://www.amazon.co.uk/Advanced-API-Security-Definitive-Guide/dp/1484220498/ref=sr\\_1\\_1?crid=32HL0SG6RMA7F&keywords=advanced+api+security&qid=1657455146&sprefi](https://www.amazon.co.uk/Advanced-API-Security-Definitive-Guide/dp/1484220498/ref=sr_1_1?crid=32HL0SG6RMA7F&keywords=advanced+api+security&qid=1657455146&sprefi)
-

## 16.7 October

### 16.7.1 And some more links (2022-10-24 07:13)

Things have calmed down a little at work, so it's time to go through a list of links that I have accumulated over the last few months. Here are some of the interesting ones.

- [1]Preparing for a system design interview
- [2]Tracking in in-app browsers
- [3]When and why does a component re-render in React?
- [4]The .NET 7 preview 7 release, including links to [5]Source Generators for P/Invoke and [6]low level struct improvements
- [7]Some views on TDD which I agree with
- [8]Writing a hypervisor from scratch, a great tutorial
- [9]Scaling Git's garbage collection
- [10]How to clone a running VM in seconds
- [11]An article on separation logic (for proving program properties when address aliasing can happen)
- [12]The infinite tower of meta-circular interpreters
- [13]20 dynamic programming interview questions
- [14]A simplified C++ memory model for x86
- [15]A note on LXC
- [16]Inspecting WebViews in MacOS
- [17]Database storage engines under the hood
- [18]An ongoing CLR issue with Ref locals
- [19]Stubs and mocks break encapsulation
- [20]Ten years of TypeScript
- [21]How Wine works
- [22]Support swap on Linux for Kubernetes
- [23]Benchmarking legacy x86 instructions

1. <https://www.educative.io/blog/how-to-prepare-system-design-interview>
2. <https://krausefx.com/blog/ios-privacy-instagram-and-facebook-can-track-anything-you-do-on-any-website-in-their-in-app-browser>
3. <https://www.joshwcomeau.com/react/why-react-re-renders/>
4. <https://devblogs.microsoft.com/dotnet/announcing-dotnet-7-preview-7/>
5. <https://github.com/dotnet/runtime/issues/60595>
6. <https://github.com/dotnet/csharplang/blob/main/proposals/csharp-11.0/low-level-struct-improvements.md>
7. <https://buttondown.email/hillelwayne/archive/i-have-complicated-feelings-about-tdd-8403/>
8. <https://rayanfam.com/topics/hypervisor-from-scratch-part-1/>
9. <https://github.blog/2022-09-13-scaling-gits-garbage-collection/>
10. <https://codesandbox.io/post/how-we-clone-a-running-vm-in-2-seconds>
11. <https://cacm.acm.org/magazines/2019/2/234356-separation-logic/fulltext>
12. <http://www.cofault.com/2022/08/3-lisp-infinite-tower-of-meta-circular.html>

13. <https://faun.pub/20-dynamic-programming-interview-questions-with-solutions-2a144b1a1e07>
14. <http://databasearchitects.blogspot.com/2020/10/c-concurrency-model-on-x86-for-dummies.html>
15. <https://lwn.net/Articles/907613/>
16. <https://blog.jim-nielsen.com/2022/inspecting-web-views-in-macos/>
17. <https://shashankbaravani.medium.com/database-storage-engines-under-the-hood-705418dc0e35>
18. <https://github.com/dotnet/runtime/issues/76929>
19. <https://blog.ploeh.dk/2022/10/17/stubs-and-mocks-break-encapsulation/>
20. <https://devblogs.microsoft.com/typescript/ten-years-of-typescript/>
21. <https://werat.dev/blog/how-wine-works-101/>
22. <https://github.com/kubernetes/enhancements/tree/master/keps/sig-node/2400-node-swap>
23. <https://www.acepace.net/2019-07-27-xlatb/>

---

Navus (2023-06-19 11:31:57)

Don't give up Clive, keep posting! Your blog is amazing!



# 2023

## 17.1 July

### 17.1.1 What is it if it isn't SQL? (2023-07-03 07:25)

We held our yearly internal tech conference at work, and I managed to put together a talk on NoSQL. I've been interested in this area for a long time, particularly the theoretical side of the arguments why a relational model isn't a good fit for some problems. Doing the talk meant that I had time to do some more reading, and put the ideas into more of a coherent story.

You can get the [1]slides as markdown for [2]Reveal JS here. The slides link out to a load of resources that can elucidate the points I was trying to make, and in some cases link into O'Reilly's learning platform as that is something that we are experimenting with at work. I should also say that being able to edit the slides in markdown made it really easy to get the material together - whenever I had a thought I could just edit the slides on GitHub.

1. <https://github.com/clivetong/Talks/blob/main/QuickDiveIntoNoSQL/README.md>
  2. <https://revealjs.com/>
- 

## 17.2 September

### 17.2.1 Hello World (again) (2023-09-04 07:09)

Ok, so I've been busy and haven't blogged for a while. That means that I have several hundred blog posts to comment on, and have a stack of 20 books that I've read. I will get to work through and comment on the good ones, but I did have to mention [1]this blog post which is more or less a perfect blog post to me. It seeks to understand something by demonstrating a small issue and then shows how that problem is fixed, describing the solution in a way that helps you fix up your mental model to see how this solution works. And all in an article that

can be digested in a single sitting. Brilliant!

1. <https://andreigatej.dev/blog/the-underlying-mechanisms-of-reacts-concurrent-mode/>
-



BLOGBOOKER

BlogBook v1.2,  
 $\text{\LaTeX} 2_{\varepsilon}$  & GNU/Linux.  
<https://www.blogbooker.com>

Edited: November 21, 2023

