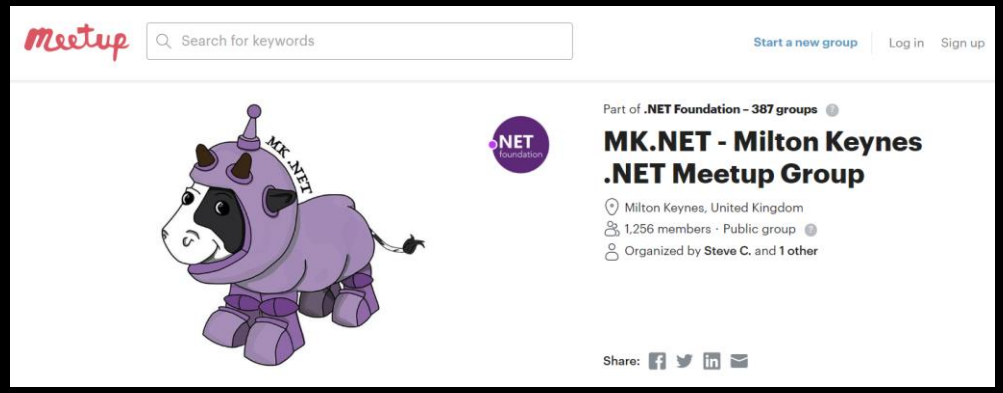
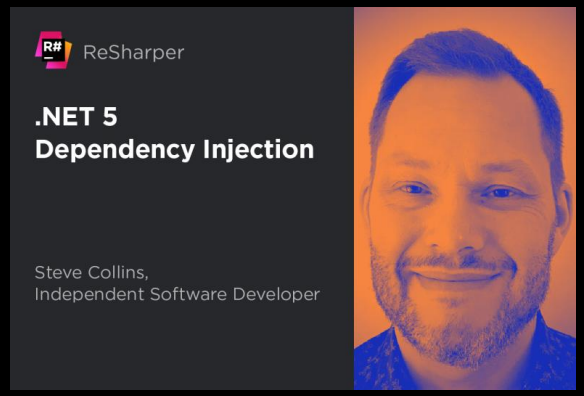
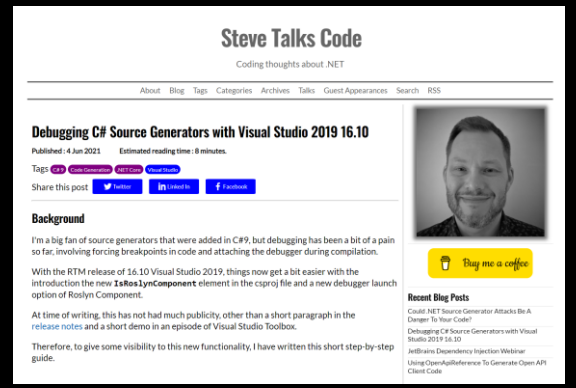

Steve Collins
and the
[Source Code]
Generation
Game

Hello!

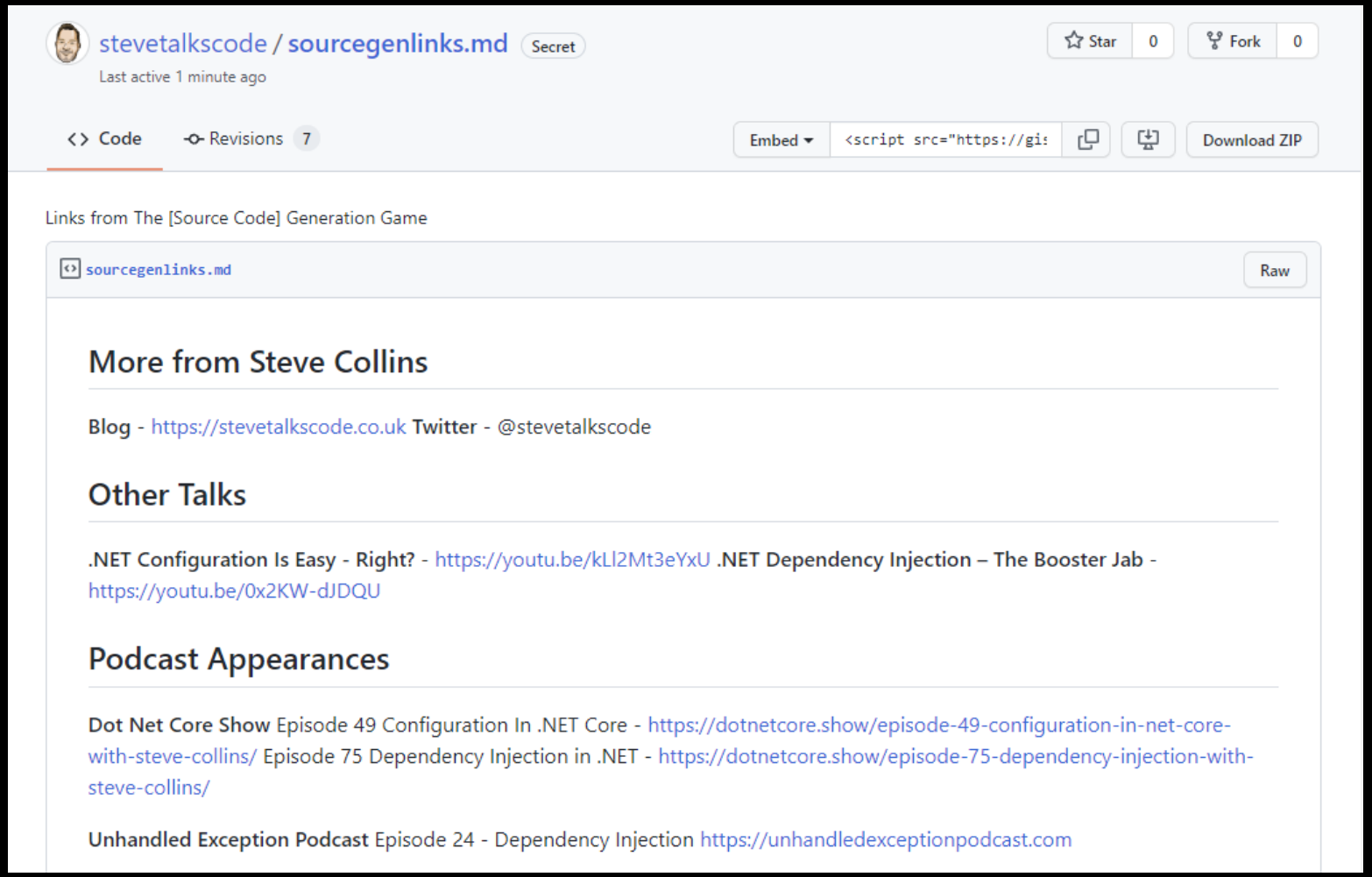
- Steve Collins (he/him) aka ‘Steve Talks Code’
- Based in West Sussex, UK
- Blog : https://stevetalkscode.co.uk
- Twitter : @stevetalkscode
- https://www.meetup.com/Milton-Keynes-NET-Meetup-Group







Links from talk - https://bit.ly/sourcegengame



stevetalkscode / sourcegenlinks.md Secret

Last active 1 minute ago

Star 0 Fork 0

Code Revisions 7 Embed <script src="https://gi: Download ZIP

Links from The [Source Code] Generation Game

sourcegenlinks.md Raw

More from Steve Collins

Blog - https://stevetalkscode.co.uk **Twitter** - @stevetalkscode

Other Talks

.NET Configuration Is Easy - Right? - https://youtu.be/kLl2Mt3eYxU **.NET Dependency Injection – The Booster Jab** - https://youtu.be/0x2KW-dJDQU

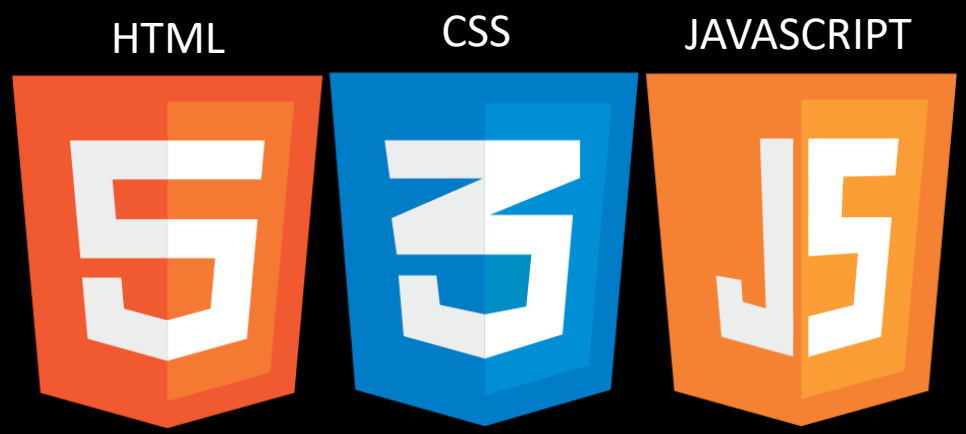
Podcast Appearances

Dot Net Core Show Episode 49 Configuration In .NET Core - https://dotnetcore.show/episode-49-configuration-in-net-core-with-steve-collins/ Episode 75 Dependency Injection in .NET - https://dotnetcore.show/episode-75-dependency-injection-with-steve-collins/

Unhandled Exception Podcast Episode 24 - Dependency Injection https://unhandledexceptionpodcast.com

What is NOT Source Code Generation?

What is NOT Source Code Generation?



Different from code rendering

Code that dynamically creates HTML and JavaScript to send to some output receiver (screen, consuming code) such as WebForms, MVC, Razor Pages

What is NOT Source Code Generation?

Transpilers are grey area

They DO generate code

Main purpose is to convert from one language/markup to another

What is NOT Source Code Generation?

{ JSON}

. [RegEx] *

Dynamically create executable code at runtime

Cannot see the source code

What is ~~DOT~~ Source Code Generation?

{ JSON}

Dynamically create executable code at runtime

SPOILER ALERT!

Cannot see the source code

.[RegEx]*

So, what is Source Code Generation?

What is Source Code Generation?

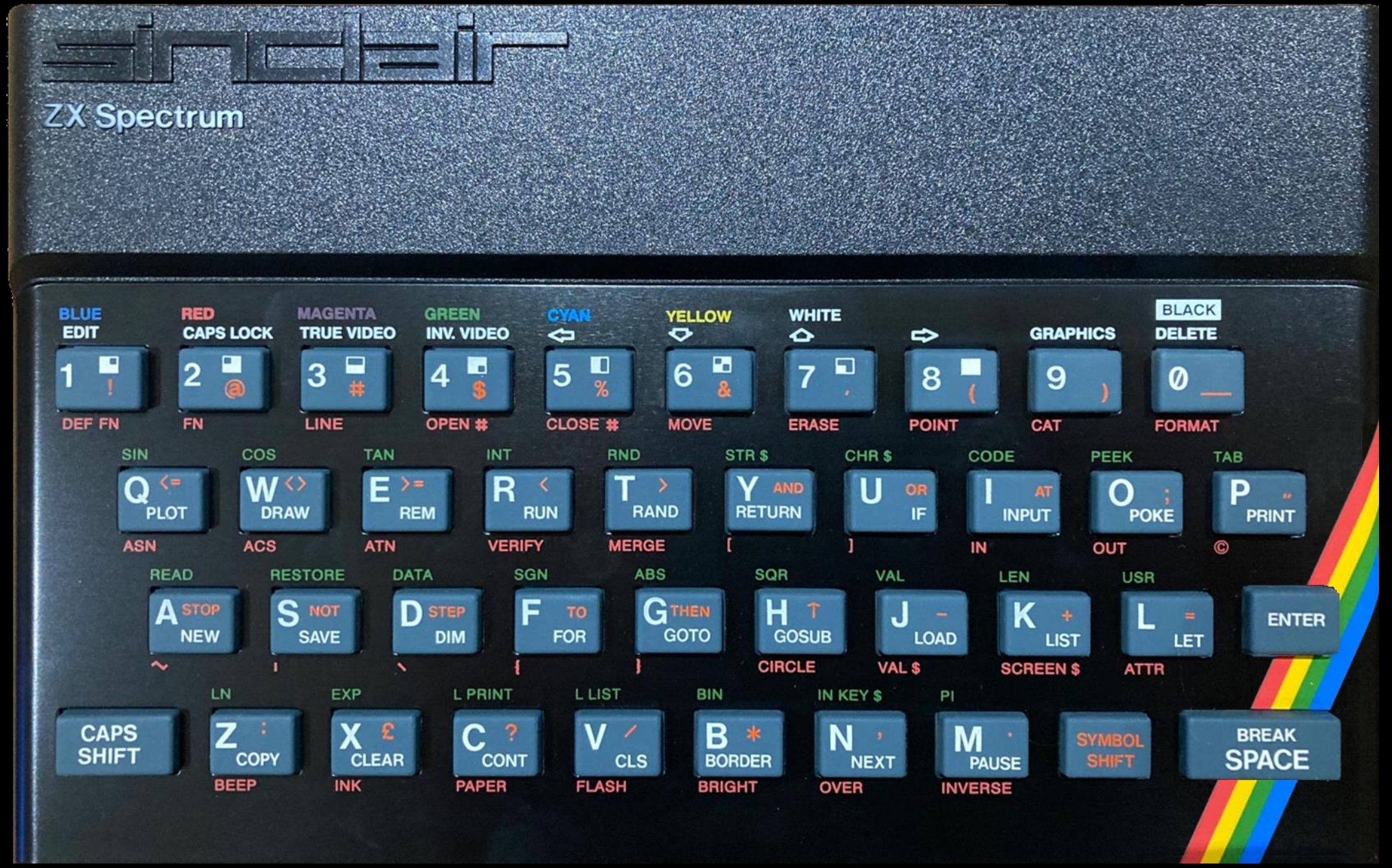
Source Generators

Creating code that will become a baked in part of a program at compile time rather than at runtime

PROGRAMING TALES

GENERATING SOURCE CODE SINCE THE 1980's


1988


sinclair
ZX Spectrum
BLUE EDIT
RED CAPS LOCK
MAGENTA TRUE VIDEO
GREEN INV. VIDEO
CYAN
YELLOW
WHITE
GRAPHICS
BLACK DELETE
1 ! DEF FN
2 @ FN
3 # LINE
4 $ OPEN #
5 % CLOSE #
6 & MOVE
7 ' ERASE
8 (POINT
9) CAT
0 _ FORMAT
SIN Q <= PLOT ASN
COS W <> DRAW ACS
TAN E >= REM ATN
INT R < RUN VERIFY
RND T > RAND MERGE
STR $ Y AND RETURN
CHR $ U OR IF
CODE I AT INPUT IN
PEEK O ; POKE OUT
TAB P " PRINT ©
READ A STOP NEW ~
RESTORE S NOT SAVE |
DATA D STEP DIM \
SGN F TO FOR {
ABS G THEN GOTO }
SQR H ↑ GOSUB CIRCLE
VAL J - LOAD VAL $
LEN K + LIST SCREEN $
USR L = LET ATTR
ENTER
CAPS SHIFT
LN Z : COPY BEEP
EXP X £ CLEAR INK
L PRINT C ? CONT PAPER
L LIST V / CLS FLASH
BIN B * BORDER BRIGHT
IN KEY $ N , NEXT OVER
PI M . PAUSE INVERSE
SYMBOL SHIFT
BREAK SPACE


The COMPLETE
All you want to know
Teach yourself Bas
Saving and Loading
avoiding cassette
In six monthly par
FREE INSIDE ARTIC TO
YOU
SPECTR
Byte High, No Limit
PROGRAMMING
MACHINE CODE BREAKOUT
A Tony Baker Exclusive!
August 1983
sinc
us
75p
STRETCH Y
TO I
Queen says: 'Arise, Sir Clive'
Middle Earth re-visited
Aids to faster programming reviewed
MORE PAGES THAN EVER!
60p
Y
COM
AUGUST 1982
Can you survive ZX D
Dragon reviewed
Vic duck shoot
BBC techniques
Spectrum sound
Atom file handling
Ecological modelling
FEBRUARY/MARCH 1984
SIN
PRO
THE MAGAZINE F
CHARACTER GENER
TAPE HEAD RENEW
ZX-81 SOUND BOAR
AUTOMATIC RESTAR
RAM RE-SET
SPECIAL FEATURE
Your Xmas guide to add-ons
AN ARGUS
Z
COMI
Britain's Biggest Ma
Over 130 pages of information and programs for ZX Spectrum, ZX81 and ZX80 computers
BIGGEST EVER ISSUE
Comprehensive checklists for Spectrum software
Round up your ZX81 for the 1K Corral
Reviews – a hard look at the software mark
PLUS
Lots of chall
programs to
SIN
MORE SOFTWARE REVIEWS
128K+2 CODING ★ JOY
MOUSE WORK ★ CHEETAH SAMP
DUCK MOVIE COMPO! ★ HI
URIDIUM ★ PRODIG
YOUR SINCLAIR
May 1987 £1.50
YS SPECIAL
GAMES WITH PERSONALITY
PLUS
Interviews with Eddie Kidd and Geoff Capes
OCEAN'S ROAD RACE
EXCLUSIVE
With Your Sinclair
WIN! WIN! WIN!
Win a Walkman, 20 Road Runners and 50 copies of Thing Bounces Back.
FREE Pull-Out poster!
TAI PAN • WORLD GAMES • ARKANOID • SABOTEUR II • UCHI MATA
DRAGON'S LAIR II • QUARTET • RENEGADE • THRUST II • SCORPION
HINTS'N'TIPS ON OVER 15 NEW GAMES • ADVENTURE SOLUTION SPECIAL

PROGRAM PRINTOUT

GRAPHICS INSTRUCTIONS

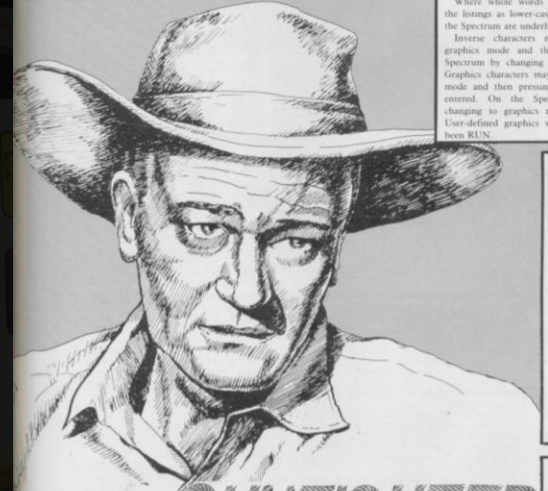
Instructions for graphics characters are printed in lower-case letters in our listings. They are enclosed by brackets and separated by colons to distinguish them and the brackets and colons should not be entered.

Inverse characters are represented by the letter "i" and graphics characters by "g". Thus an inverse W would be represented by "iw", a graphics W by "gw", and an inverse graphics W by "igw".

Spaces are represented by "sp" and inverse spaces by "isp". Whenever any character is to be used more than once, the number of times it is to be used is shown before it, together with a multiplication sign. Thus "6*isp" means six inverse spaces and "(g4:4*i4:g)" would be entered as a graphic four, followed by an inverse four repeated four times, followed by a graphics three. Control codes appear with cc before the appropriate key. They are obtained by pressing Caps Shift while in Extended mode.

Where whole words are to be written in inverse letters they appear in the listings as lower-case letters. Letters to be entered in graphics mode on the Spectrum are underlined.

Inverse characters may be entered on the ZX-81 by changing to graphics mode and then typing the appropriate characters and on the Spectrum by changing to inverse video and typing the appropriate letters. Graphics characters may be entered on the ZX-81 by changing to graphics mode and then pressing symbol shift while the appropriate characters are entered. On the Spectrum graphics characters may be obtained by changing to graphics mode and then pressing the appropriate character. User-defined graphics will appear as normal letters until the program has been RUN.



GUNFIGHTER

JOHN WAYNE lives again in **Gunfighter** by Mark Hinchcliffe of Aberdeen. You must eliminate the Boot Hill Gang, who are holed up on a farm outside town. The gang members appear from behind cover and you must train your sights on them before they get you.

Gunfighter runs on the 48K Spectrum and contains two machine code routines. One scans the keyboard and allows you to fire on the move, the other produces the sound effects.

The program uses our special abbreviations for graphics characters, so please read the instructions above before entering the listing.

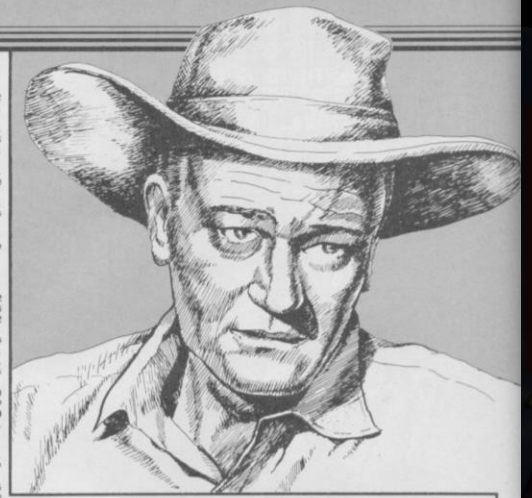
```
  1 REM *******GUNFIGHT*******
by Mark Hinchcliffe
  5 CLEAR 38990
  10 BORDER 0: PAPER 6: CLS
  30 PRINT  INK 1; PAPER 5;AT 5,
10;"PLEASE WAIT";AT 6,11;"A MOME
NT"
  40 GO TO 5000
  100 POKE 40000,16: POKE 40001,1
0: PRINT  OVER 1;AT 10,16;CHR$ 1
44
  110 RANDOMIZE USR 39000
  120 GO SUB (con+10)*2000
  124 REM Fire routine
  125 IF f=1 THEN  GO SUB 1000: G
O TO 110
  130 IF PEEK 40010=1 THEN  LET y
=PEEK 40000: LET x=PEEK 40001: G
O SUB 500: LET f=1: PRINT  OVER
1; PAPER 6; INK 2;AT x,y;CHR$ 15
0: RANDOMIZE USR 39165: GO TO 14
0
  135 GO TO 110
  140 LET shots=shots-1
  150 IF shots<12 THEN  PRINT  PA
PER 7;AT 20,16+shots);"(sp)"
  160 IF shots>11 THEN  PRINT  PA
PER 7;AT 21,(shots-6);"(sp)"
  170 GO SUB 500: GO SUB 1000
  180 GO TO 110
  500 PLOT 135,16: DRAW  OVER 1;(
(PEEK 40000)*8)-131,155-((PEEK 4
0001)*8)
  510 RETURN
  999 REM Gunfighter hit?
  1000 IF x=xx AND y=yy AND con=3
THEN  GO SUB 3000: GO TO 1040
  1005 IF x=xx+1 AND y=yy AND con=
3 AND d=149 THEN  GO SUB 3000: G
O TO 1040
  1010 PRINT  OVER 1; INK 0; PAPER
 6;AT x,y;CHR$ 150: LET f=0: POK
E 40010,0
  1015 IF hit=10 THEN  GO TO 8000
  1020 IF shots=0 AND con=3 THEN
GO SUB 3000: GO TO 8600
  1025 IF shots=0 THEN  GO TO 8600
  1030 RETURN
  1040 LET sc=sc+((INT (300/t))+1)
*100: LET con=0: LET t=0: LET hi
t=hit+1: PRINT #1; PAPER 7; INK
0;AT 1,(hit+5);CHR$ 146: PRINT
```

```
PAPER 7;AT 21,26;sc: GO TO 1010
1999 REM Gunfighter timing
2000 RANDOMIZE : LET del=INT (RN
D*100)+150/lev: LET con=1
2005 LET r=INT (RND*12)+1: LET x
=a(r,1): LET yy=a(r,2): LET c=a
(r,3): LET d=a(r,4)
2010 LET t=t+1: IF t=del THEN  L
ET con=2: LET t=0: GO SUB 3000:
BEEP .05,xx
2015 RETURN
2020 LET del=INT (RND*(60/lev))+
90/lev: LET con=3
2029 REM Sheriff hit?
2030 LET t=t+1: IF t=del THEN  L
ET con=0: LET t=0: RANDOMIZE USR
 39165: GO SUB 3000: LET h=INT (
RND*(4-lev))+1
2040 IF h=1 THEN  FOR n=7 TO 0 S
TEP -1: OUT 254,n: PAUSE 2: NEXT
 n: LET lives=lives-1: PRINT #1;
 PAPER 7;AT 1,(26+lives);"(sp)":
 PRINT  PAPER 6;AT 0,8;"(sp)": L
ET h=0
2050 IF lives=0 THEN  GO TO 8300
2060 RETURN
2999 REM Print gunfighter
3000 PRINT  OVER 1; PAPER 6;AT x
,yy;CHR$ c;AT (xx+1),yy;CHR$ d
3010 RETURN
4999 REM UDG'S
5000 FOR n=0 TO 95
```

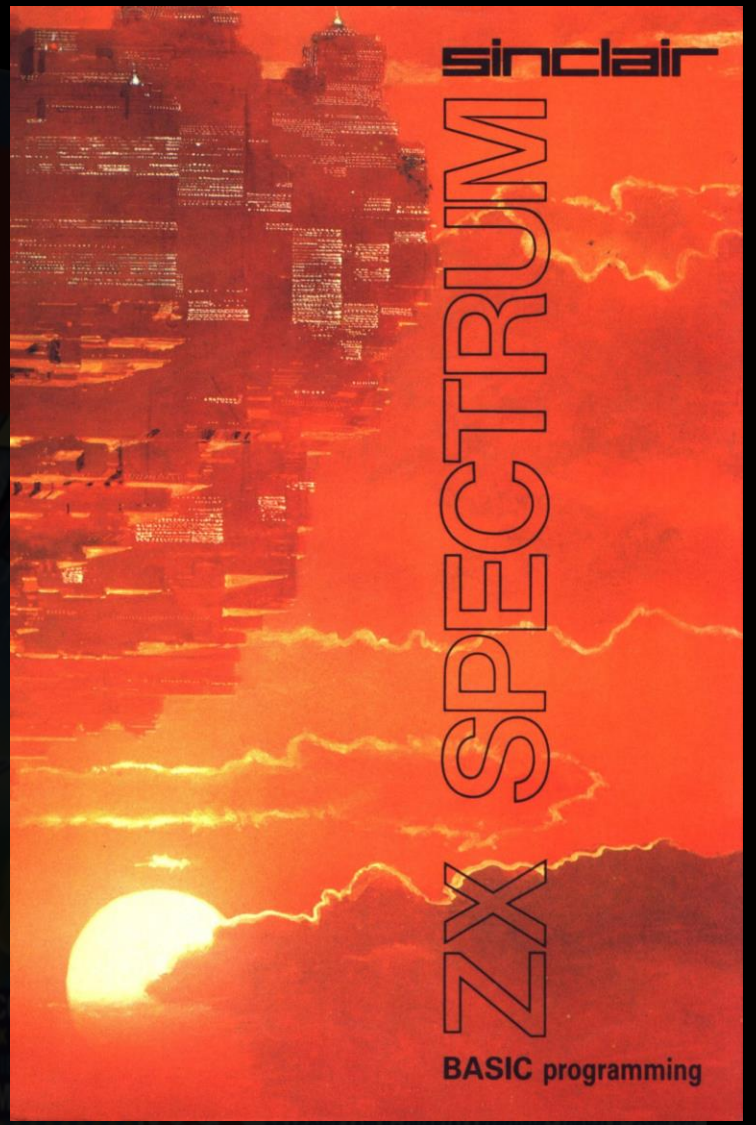
continued on page 86

continued from page 85

```
5010 READ a
5020 POKE (65368+n),a
5030 DATA 24,24,0,195,195,0,24,2
4,0,24,60,60,60,60,60,60,126,24,24,
126,24,24,60,126,0
5040 DATA 16,16,56,84,84,40,40,1
00,24,60,24,24,60,90,153,153,153
,60,60,102,102,102,102,231,132,1
00,127,62,124,124,158,1
5050 DATA 60,66,165,153,153,165,
66,60,56,70,161,149,161,171,69,0
2,66,36,36,36,36,37,35,34
5060 DATA 36,36,36,36,36,36,36,1
29,69,170,170,146,146,146,146,14
6
5070 NEXT n
5099 REM Gunsight control/Gunsho
t
5100 FOR n=0 TO 107
5110 READ a
5120 POKE (39000+n),a
5130 DATA 237,75,64,156,62,247,2
11,255,219,254,230,16,194,119,15
2,58,64,156,38,0,100,202,143,152
,61,50,64,156,195,143,152
5140 DATA 62,239,211,255,219,254
,230,4,194,143,152,58,64,156,38,
31,100,202,143,152,60,50,64,156,
62,239,211,255,219,254,230,0,194
,170
5150 DATA 152,58,65,156,38,0,100
,202,194,152,61,50,65,156,195,19
4,152,62,239,211,255,219,254,230
,16,194,194,152,50,65,156,38,19,
100
5160 DATA 202,194,152,60,50,65,1
56,62,239,211,255,219,254,230,1,
194,210,152,62,1,50,74,156,237,9
1,64,156,62,22,215,120,215,121
5170 DATA 215,62,21,215,62,1,215
,62,144,215,62,22,215,122,215,12
3,215,62,21,215,62,1,215,62,16,2
15,62,0,215,62,144,215,201
5180 DATA 33,0,0,14,0,22,1,126,2
30,24,211,254,65,16,254,35,21,32
,244,12,32,239,201
5190 NEXT n
5299 REM Gunfighter data
5300 DIM a(12,4)
5310 FOR n=1 TO 12: FOR m=1 TO 4
5320 READ a(n,m)
5330 NEXT m: NEXT n
5340 DATA 11,1,140,149,13,6,148,
32,13,9,148,149,17,12,148,149,10
,25,148,32,16,30,148,32,9,22,148
,32,4,29,147,32,2,22,147,32,2,8,
147,32,16,21,148,32,13,12,148,14
9
5399 REM Instructions
5400 CLS : BEEP .2,15
5410 PRINT AT 1,11;"GUNFIGHT"
5420 PRINT AT 5,1;"You are the l
ocal sheriff and";AT 6,1;"your t
own is being terrorized";AT 7,1;
"by the Boot Hill Gang.";AT
8,1;"You have decided to go to t
heir";AT 9,1;"ranch for a showdo
wn"
5430 PRINT AT 11,1;"Your gunsigh
t is controlled by";AT 12,1;"the
 cursor keys (5,6,7&8) for";AT 1
3,1;"left,down,up and right";AT
14,1;"respectively. The";"0";"key
fires";AT 15,1;"your gun"
5440 PRINT  INK 5; PAPER 1;AT 18
,4;"PRESS ANY KEY TO START"
5450 IF INKEY$="" THEN  GO TO 54
50
5460 BEEP .2,15: CLS
5499 REM Draw scene
5500 FOR n=0 TO 31: PRINT AT 17,
n;CHR$ 155;AT 18,n;CHR$ 155: NEX
T n: PRINT AT 17,11;CHR$ 138;CHR
$ 32;CHR$ 133;AT 18,11;CHR$ 138;
CHR$ 32;CHR$ 133
5510 FOR n=0 TO 21: PRINT AT 2,n
;CHR$ 155: NEXT n: PRINT AT 2,8;
CHR$ 32
5515 FOR n=1 TO 17: READ a,b,c:
PRINT AT a,b;CHR$ c: NEXT n
```

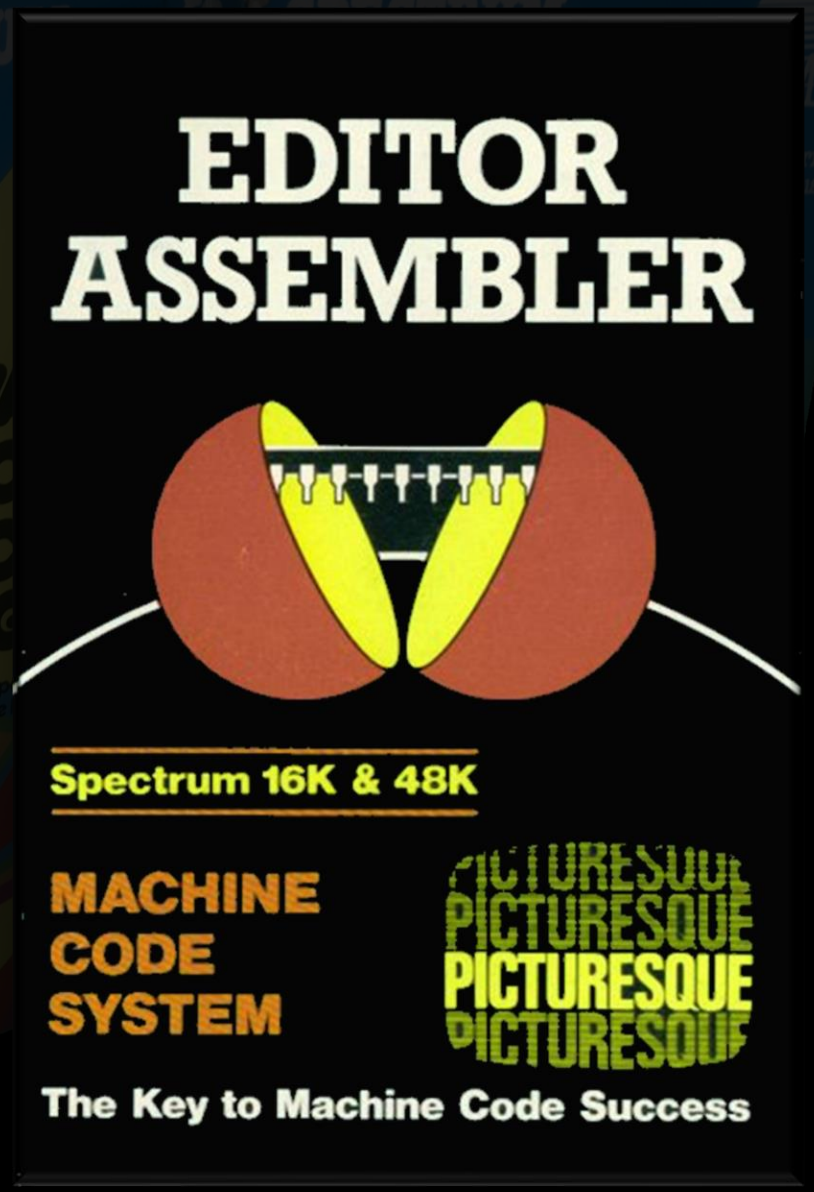


```
5520 DATA 12,25,151,12,22,151,0,
23,153,0,27,152,0,29,154,1,23,15
4,1,25,152,1,27,153,2,24,152
5525 DATA 2,25,153,2,27,154,2,28
,152,3,24,153,3,25,154,3,28,153,
4,24,154,4,28,154
5530 PLOT 95,56: DRAW -15,0: DRA
W 0,28: DRAW -9,0: DRAW 0,-28: D
RAW -31,0: DRAW 0,40: DRAW 55,0:
 DRAW 0,-40
5540 PLOT 50,63: DRAW -12,0: DRA
W 0,10: DRAW 12,0: DRAW 0,-10
5550 PLOT 40,56: DRAW -24,16: DR
AW 0,40: DRAW 12,4: DRAW 12,-19
5560 PLOT 28,116: DRAW 55,0: DRA
W 12,-19
5570 PLOT 200,76: DRAW -32,0: DR
AW 0,11: DRAW 32,0: DRAW 0,-11
5580 PLOT 176,76: DRAW -12,0: DR
AW 0,11: DRAW 12,-8: DRAW 32,0:
DRAW -12,8: DRAW -32,0
5590 PLOT 196,87: DRAW 0,8: PLOT
 170,82: DRAW -12,-10: PLOT 164,
72: DRAW -6,4
6010 FOR n=0 TO 127: POKE (23168
+n),56: NEXT n
6020 PRINT  PAPER 7;AT 20,1;"AMM
O";AT 20,20;"LEVEL";AT 21,20;"SC
ORE"
6030 PRINT #1; PAPER 7; INK 0;AT
 1,1;"HITS";AT 0,20;"HIGH";AT 1,
20;"LIVES"
6040 LET hs=0
6099 REM Start
6100 PRINT #1; PAPER 7;AT 1,6;"(
10*sp)"; FOR n=20 TO 21: FOR m=6
TO 17: PRINT  INK 1; PAPER 7;AT
n,m;CHR$ 145: NEXT m: NEXT n
6200 PRINT  INK 1; PAPER 5;AT 5,
7;"(2*sp)SELECT LEVEL(2*sp)";AT
6,9;"(2*sp)(1 TO 3)(2*sp)"; BEEP
.1,10
6210 IF PEEK 23557<>5 THEN  GO T
O 6210
6220 IF PEEK 23560<49 OR PEEK 23
560>51 THEN  GO TO 6210
6230 LET lev=(PEEK 23560)-48
6240 PRINT  PAPER 7;AT 20,26;lev
; PAPER 6;AT 5,7;"(16*sp)";AT 6,
9;"(12*sp)"
6500 LET xx=0: LET yy=0: LET h=0
: LET t=0: LET f=0: LET wo=0: LE
T hit=0: LET sc=0: LET shots=24:
 LET lives=3: LET con=0: PRINT J
1; INK 2; PAPER 7;AT 1,26;CHR$ 1
47;CHR$ 147;CHR$ 147; INK 0;AT 0
,26;hs: PRINT  PAPER 7;AT 21,26;
"0(5*sp)"
6600 FOR n=0 TO 7: READ a,b: BEE
P a,b: NEXT n
6610 DATA -2,16,.2,21,.2,16,.2,2
1,.5,16,.4,12,.45,14,.45,9
6700 GO TO 100
7999 REM End routines
8000 PRINT  FLASH 1; INK 1; PAPE
R 5;AT 5,11;"WELL DONE";AT 6,8;"
!!!Youve won!!!"; FOR n=0 TO 10:
 BEEP .05,n: BEEP .05,(n+3): NEX
T n: PAUSE 100
8010 LET sc=sc+1000+(shots*200)+
(lives*500): GO TO 9010
8300 PRINT AT 5,11;"You have";AT
6,8;"been shot dead": GO TO 890
0
8600 PRINT AT 5,10;"You have no"
;AT 6,9;"bullets left": GO TO 89
00
8900 FOR n=0 TO 10: READ a,b: BE
EP a,b: NEXT n
8910 DATA .45,0,.3,0,.15,0,.45,0
,.3,3,.15,2,.3,2,.15,0,.3,0,.15,
-1,.45,0
9010 PRINT  PAPER 7;AT 21,26;sc
9020 IF sc>hs THEN  LET hs=sc: P
RINT #1; PAPER 7; INK 0;AT 0,26;
hs
9030 PRINT  INK 1; PAPER 5;AT 5,
9;"PRESS ANY KEY";AT 6,8;"  TO C
ONTINUE  "
9040 IF INKEY$="" THEN  GO TO 90
40
9050 PRINT  PAPER 6;AT 5,9;"(13*
sp)";AT 6,8;"(15*sp)": RESTORE 8
910: RESTORE 6610: PRINT  OVER 1
; PAPER 6;AT PEEK 40001,PEEK 400
00;CHR$ 144: POKE 40010,0: GO TO
 6100
```

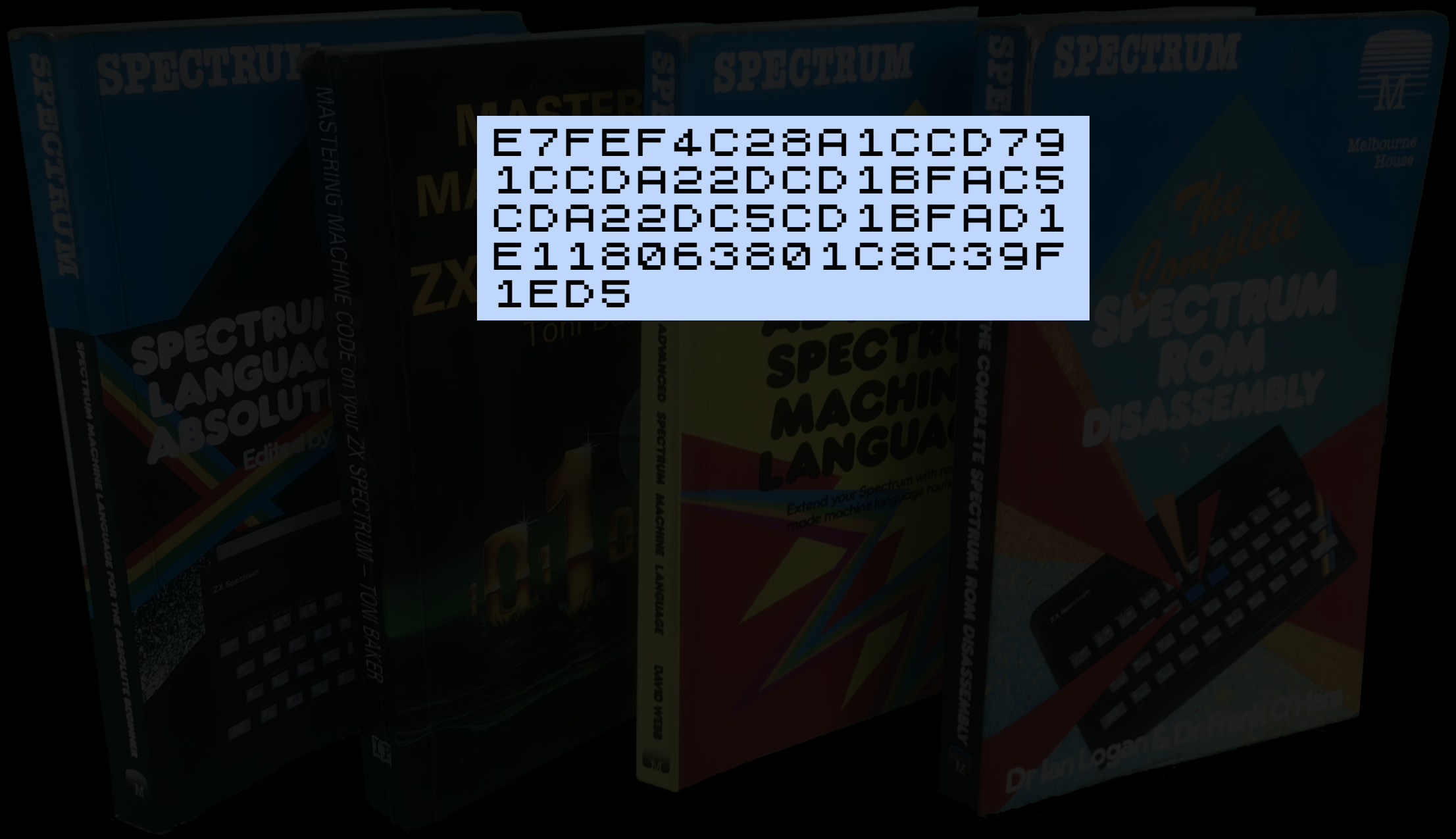

SPECTRUM
SPECTRUM MACHINE LANGUAGE FOR THE ABSOLUTE BEGINNER
MASTERING MACHINE CODE on your ZX SPECTRUM – TONI BAKER
MASTER
MACHINE
on your
ZX SPECT
Toni Bak
SPECTRUM
ADVANCED SPECTRUM MACHINE LANGUAGE DAVID WEBB
ADVANC
SPECTRU
MACHIN
LANGUAG
Extend your Spectrum with
made machine language
SPECTRUM
THE COMPLETE SPECTRUM ROM DISASSEMBLY
Melbourne House
The Complete
SPECTRUM
ROM
DISASSEMBLY
Dr Ian Logan & Dr Frank O'Hara

Mem location	Compiled bytes	Line No	Jump Label	Mnemonic Command	Comments
FA00	E7	0200		RST 20H	Set next statement
FA01	FEF4	0210		CP 244	Is it POKE ?
FA03	C28A1C	0220		JP NZ,RPT_C	If not, Nonsense in Basic
FA00	CD791C	0230		CALL EX2NM	Get the two parameters
FA09	CDA22D	0240		CALL GETBC	Fetch length into BC
FA0C	CD1BFA	0250		CALLEROR?	Is it valid ?
FA0F	C5	0260		PUSH BC	Save it
FA10	CDA22D	0270		CALL GETBC	Fetch start into BC
FA13	C5	0280		PUSH BC	Save it
FA14	CD1BFA	0290		CALL EROR?	Is it valid ?
FA17	01	0300		POP DE	DE = start address
FA18	E1	0310		POPHL	HL = No of bytes to read
FA19	1800	0320		JR RUN	Start the actual routine
FA1B	3801	0330	EROR	JR C,ERR00	Error if number too big
FA1D	C8	0340		RET Z	Return if positive number
FA1E	C39F1E	0350	ERROR	JP RPT B	Report Integer too big
FA21	D5	0370	RUN	PUSH DE	Save start address

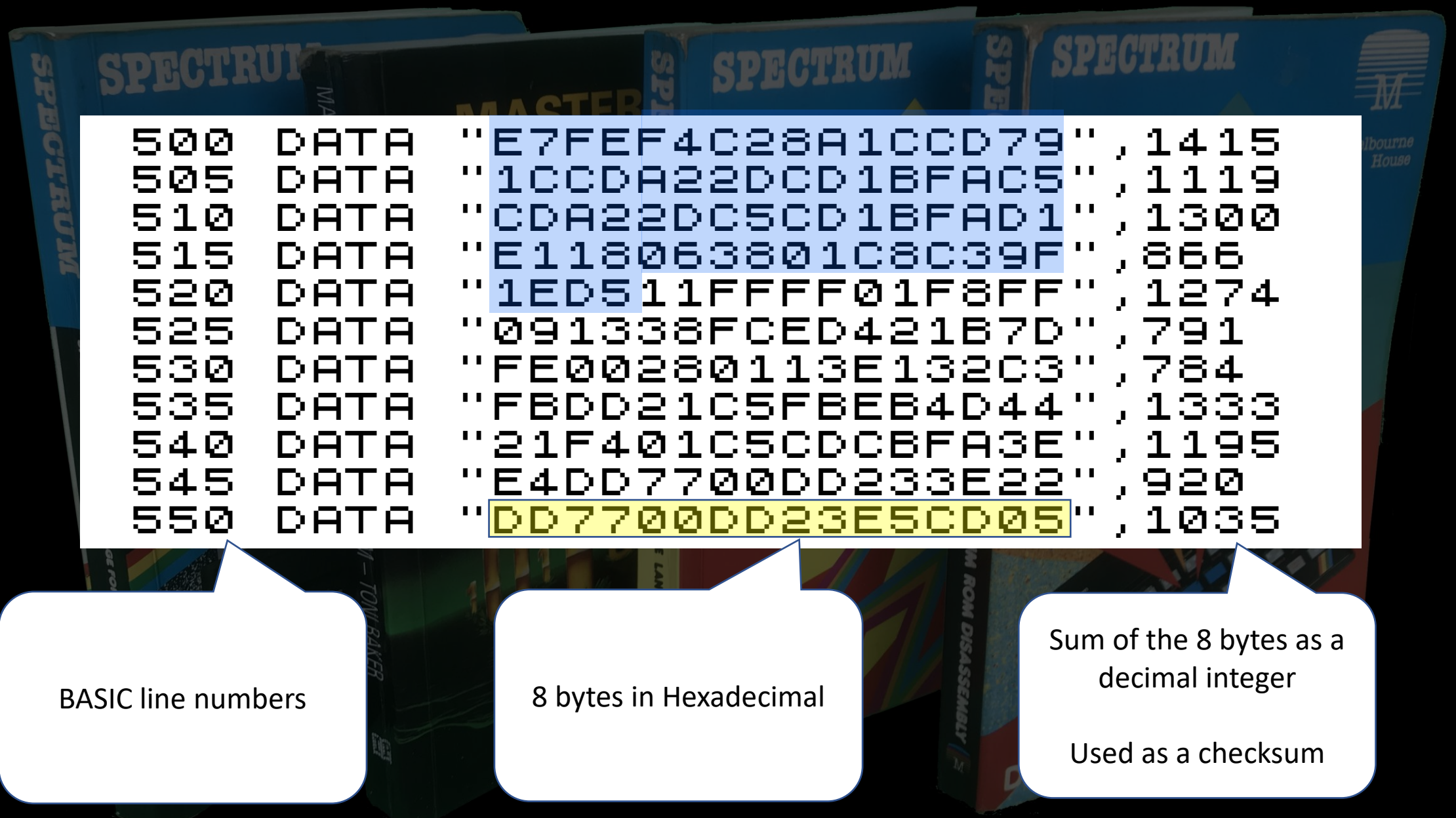


Compiled bytes

```
E7
FEF4
C28A1C
CD791C
CDA22D
CD1BFA
C5
CDA22D
C5
CD1BFA
01
E1
1800
3801
C8
C39F1E
D5
```




```
E7FEF4C28A1CCD79
1CCDA22DCD1BFAC5
CDA22DC5CD1BFAD1
E1180638 01C8C39F
1ED5
```


500 DATA "E7FEF4C28A1CCD79",1415
505 DATA "1CCDA22DCD1BFAC5",1119
510 DATA "CDA22DC5CD1BFAD1",1300
515 DATA "E11806380 1C8C39F",866
520 DATA "1ED511FFFF01F8FF",1274
525 DATA "091338FCED421B7D",791
530 DATA "FE00280113E132C3",784
535 DATA "FBDD21C5FBEB4D44",1333
540 DATA "21F401C5CDCBFA3E",1195
545 DATA "E4DD7700DD233E22",920
550 DATA "DD7700DD23E5CD05",1035
BASIC line numbers
8 bytes in Hexadecimal
Sum of the 8 bytes as a decimal integer
Used as a checksum

WHAT HAS ALL THIS GOT TO DO WITH SOURCE CODE GENERATION?

DATABANKER

by Tom Baker

Tom Baker hmmmmm, that's a familiar name. Doctor Who? Naaahhh. *Fastape* routine printed in the October 1987 issue? Yeahhhh! I remember now, Tom's routine was the best of the fast loaders we received. Well, Tom's come up tops again with his incredibly (and at the same time, very concise), valuable program to create data statements and lines of your own code.

This program is of value to anyone who has a huge chunk of code in memory and can't be bothered to put it into data statements. So here's Tom's program to do all that for you. Wowsers!

Method

To get this working, simply type in the listing and save it with SAVE "databanker" LINE 10 and you're all set, (goooooo!).

Banking

Right, to get you lot out there to learn how to work this bijou prog, we shall go through an (imaginary) working example. Sitting comfortably? Then I shall begin.

You have some code at 60000 which is 120 bytes long and you want to put it into data statements. Do you:
a) Scream, shout and pull your hair out — you can't stand DATA?
b) Use Tom Baker's *Data Banker* program?
c) Go "yibble, yibble" — you've had enough of machine code?

The correct answer is, of course, 'b'. Now edit line 50 of the program. It will look like this:

```
50 LET f=0: LET addr=0: LET len=60
```

Change the line so that it reads:

```
50 LET f=0: LET addr=60000: LET len=120
```

Easy innit (peeps)? Now RUN the routine and data lines will appear at lines 8995 onwards in this format:

DATA "(*address), no. of bytes of line (max=4), data bytes separated by spaces, checksum"

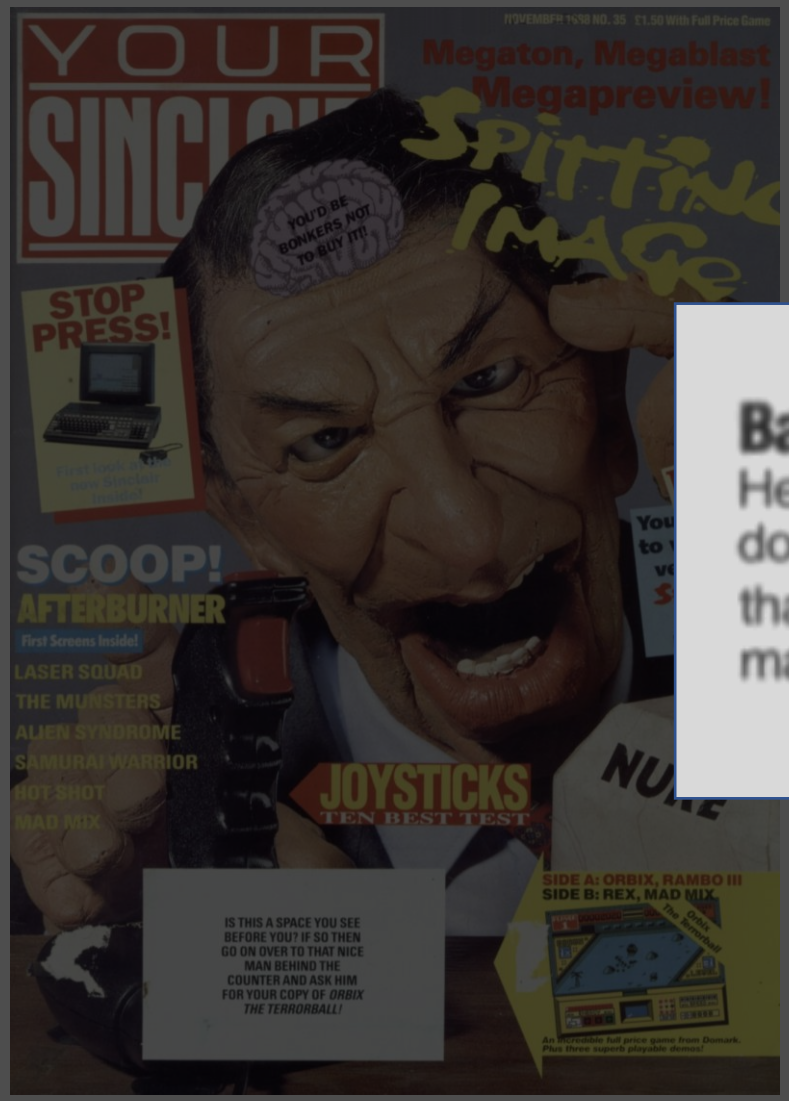
The program itself is in two sections. The first section (10-220) is the part which assembles the code into DATA, and the second part (1000+) reads the data and makes sense of it. Geddit? Good, now no complaints.

Basic Program

Here it is, quite short for what it does, and I'm quite surprised that Tom used Basic intead of machine code, but there you are.

```
 0>REM   TBB, 1986 BY TOM BA
KER

  10 GO TO 50
  20 LET P1=INT (N/256): LET P2=
N-(P1*256): POKE ADR,P2: POKE AD
R+1,P1: RETURN
  30 LET VA=VA-1: IF PEEK VA<>13
 THEN  GO TO 30
  40 RETURN
  50 LET F=0: LET ADDR=0: LET LE
N=60
  60 RESTORE : DATA 1,0,0,33,0,0
,195,85,22: FOR A=65527 TO 65535
: READ B: POKE A,B: NEXT A
  70 LET A$=" DATA "": IF F=0 T
HEN  LET A$=" DATA ""#"+STR$ ADD
R+","
  80 LET F=F+1: LET B=40: IF LEN
<41 THEN  LET B=LEN
  90 LET B$=STR$ B+",": IF B<10
THEN  LET B$="0"+B$
 100 LET A$=A$+B$
 110 LET C=0: FOR A=1 TO B: LET
N=PEEK ADDR: LET ADDR=ADDR+1: LE
T C=C+N: LET A$=A$+STR$ N: IF A<
>B THEN  LET A$=A$+" "
 120 NEXT A: LET A$=A$+","+STR$
C+""""+CHR$ 13
 130 LET VA=PEEK 23627+PEEK 2362
8*256
 140 GO SUB 30: GO SUB 30: GO SU
B 30
 150 LET NL=VA+1
 160 LET VA=VA-1: IF PEEK (VA-1)
<>13 THEN  GO TO 160
 170 LET DL=(PEEK VA*256+PEEK (V
A+1))+5
 180 LET ADR=0: LET N=DL: GO SUB
 20: LET C$=CHR$ P1+CHR$ P2: LET
 ADR=0: LET N=LEN A$: GO SUB 20:
 LET A$=C$+CHR$ P2+CHR$ P1+A$
 190 LET N=LEN A$: LET ADR=65528
: GO SUB 20: LET N=NL: LET ADR=6
5531: GO SUB 20: LET U$=USR 6552
7
 200 FOR A=1 TO LEN A$: POKE NL,
CODE A$(A): LET NL=NL+1: NEXT A
```

```
 210 PRINT AT 0,0;"............
.................";AT 0,0;"LIN
E ";DL;" COMPLETED": LET LEN=LEN
-40: IF LEN>0 THEN  GO TO 70
 220 STOP
1000 LET LINE=9000: RESTORE LINE
1010 READ A$
1015 IF A$(1)<>"#" THEN  GO TO 1
050
1020 LET B$="": LET A=2
1030 LET B$=B$+A$(A): LET A=A+1:
 IF A$(A)<>"," THEN  GO TO 1030
1040 LET AD=VAL B$: LET A$=A$(A+
1 TO )
1050 LET C=0: LET A=4: FOR L=1 T
O VAL A$( TO 2)
1060 LET B$=""
1070 LET B$=B$+A$(A): LET A=A+1:
 IF A$(A)<>" " AND A$(A)<>"," TH
EN  GO TO 1070
1080 LET A=A+1: POKE AD,VAL B$:
LET C=C+VAL B$: LET AD=AD+1: NEX
T L
1090 IF C<>VAL A$(A TO ) THEN  C
LS : BEEP 1,10: PRINT AT 0,0; IN
K 9;"DATA ERROR WITHIN LINE ";LI
NE;".....PLEASE CORRECT": STOP
1100 LET LINE=LINE+5: READ A$: I
F A$(1)<>"#" THEN  GO TO 1015
1110 STOP
8995 REM --START OF DATA--
9999 DATA ";END OF DATA;"
```



DATABANKER

by Tom Baker

Tom Baker hmmmmm, that's a familiar name. Doctor Who? Naaahhh. *Fastape* routine printed in the October 1987 issue? Yeahhhh! I remember now, Tom's routine was the best of the fast loaders we received. Well, Tom's come up tops again with his incredibly (and at the same time, very concise), valuable program to create data

(imaginary) working example. Sitting comfortably? Then I shall begin.

You have some code at 60000 which is 120 bytes long and you want to put it into data statements. Do you:
a) Scream, shout and pull your hair out — you can't stand DATA?
b) Use Tom Baker's *Data Banker* program?
c) Go "yibble, yibble" — you've

The program itself is in two sections. The first section (10-220) is the part which assembles the code into DATA, and the second part (1000+) reads the data and makes sense of it. Geddit? Good, now no complaints.

Basic Program

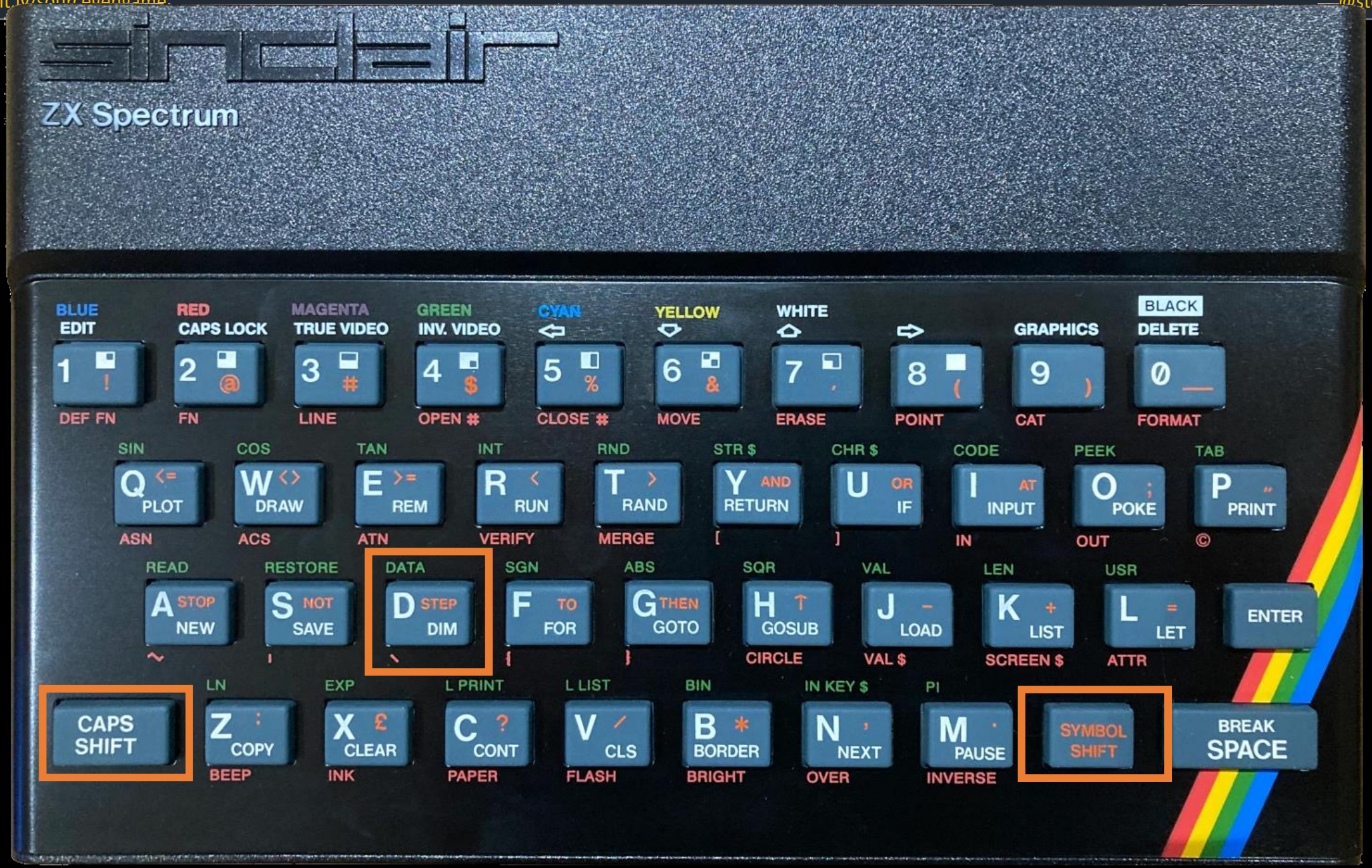
Here it is, quite short for what it does, and I'm quite surprised that Tom used Basic intead of machine code, but there you are.

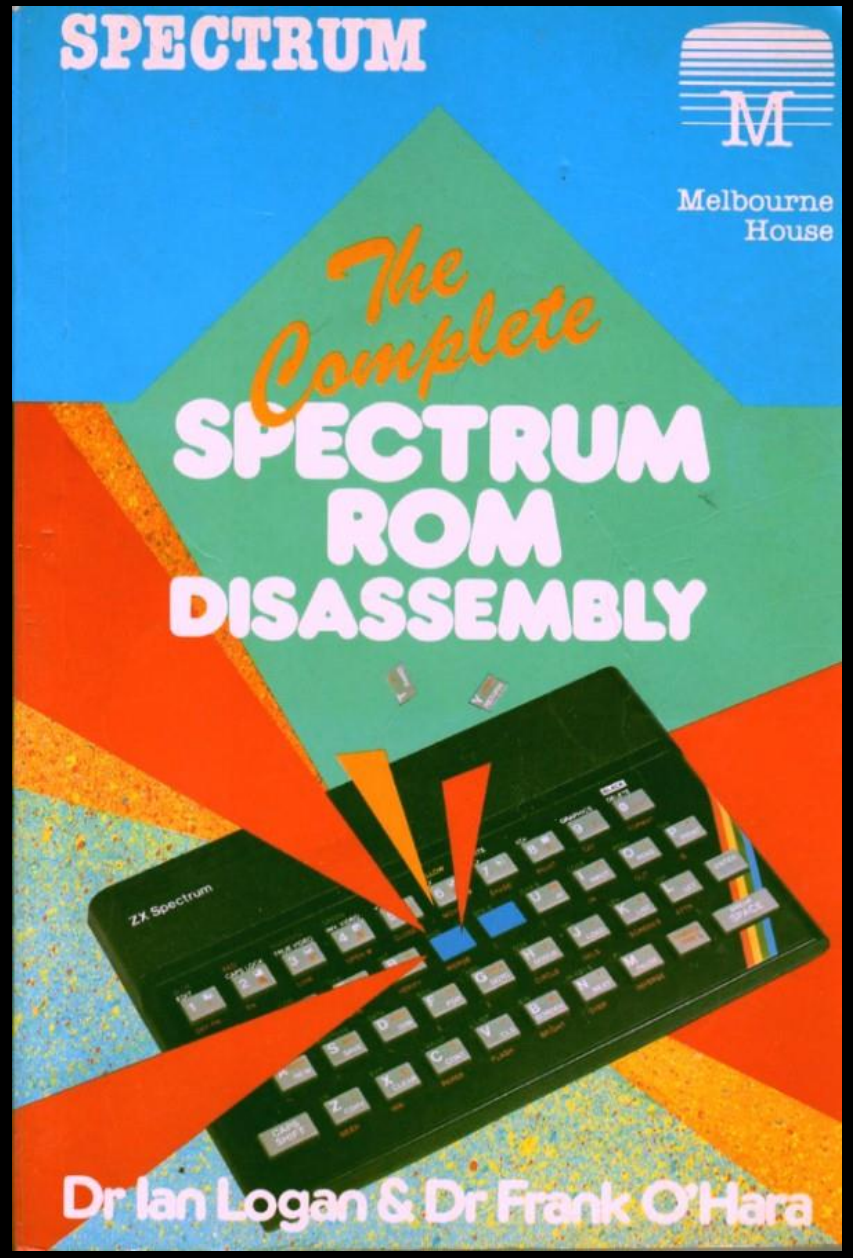
Basic Program
Here it is, quite short for what it does, and I'm quite surprised that Tom used Basic intead of machine code, but there you are.

```
0 REM  TBB, 1986 BY TOM BA
KER

 10 GO TO 50
 20 LET P1=INT (N/256): LET P2=
N-(P1*256): POKE ADR,P2: POKE AD
R+1,P1: RETURN
 30 LET VA=VA-1: IF PEEK VA<>13
 THEN  GO TO 30
 40 RETURN
 50 LET F=0: LET ADDR=0: LET LE
N=60
 60 RESTORE : DATA 1,0,0,33,0,0
,195,05,22: FOR A=65527 TO 65535
: READ B: POKE A,B: NEXT A
 70 LET A$="" DATA """: IF F=0 T
HEN  LET A$=" DATA """+STR$ ADD
R+","
 80 LET F=F+1: LET B=40: IF LEN
<41 THEN  LET B=LEN
 90 LET B$=STR$ B+",": IF B<10
THEN  LET B$="0"+B$
 100 LET A$=A$+B$
 110 LET C=0: FOR A=1 TO B: LET
N=PEEK ADDR: LET ADDR=ADDR+1: LE
T C=C+N: LET A$=A$+STR$ N: IF A<
>B THEN  LET A$=A$+","
 120 NEXT A: LET A$=A$+",":+STR$
C+"""+CHR$ 13
 130 LET VA=PEEK 23627+PEEK 2362
8*256
 140 GO SUB 30: GO SUB 30: GO SU
B 30
 150 LET NL=VA+1
 160 LET VA=VA-1: IF PEEK (VA-1)
<13 THEN  GO TO 160
 170 LET DL=(PEEK VA*256+PEEK (V
A+1))+5
 180 LET ADR=0: LET N=DL: GO SUB
 20: LET C$=CHR$ P1+CHR$ P2: LET
 ADR=0: LET N=LEN A$: GO SUB 20:
 LET A$=C$+CHR$ P2+CHR$ P1+A$
 190 LET N=LEN A$: LET ADR=65528
: GO SUB 20: LET N=NL: LET ADR=6
5531: GO SUB 20: LET US=USR 6552
7
 200 FOR A=1 TO LEN A$: POKE NL,
CODE A$(A): LET NL=NL+1: NEXT A
```

```
 210 PRINT AT 0,0;"                "
;"                "'"AT 0,0;"LIN
E ";10,"" COMPLETED": LET LEN=LEN
-40: IF LEN>0 THEN  GO TO 70
 220 STOP
1000 LET LINE=9000: RESTORE LINE
1010 READ A$
1015 IF A$(1)<>"*" THEN  GO TO 1
050
1020 LET B$="": LET A=2
1030 LET B$=B$+A$(A): LET A=A+1
: IF A$(A)<>"," THEN  GO TO 1030
1040 LET AD=VAL B$: LET A$=A$(A+
1 TO )
1050 LET C=0: LET A=A+ FOR L=1 T
O VAL A$: TO 2)
1060 LET B$=""
1070 LET B$=B$+A$(A): LET A=A+1:
 IF A$(A)<>"," AND A$(A)<>"," TH
EN  GO TO 1070
1080 LET A=A+1: POKE AD,VAL B$:
LET C=C+VAL B$: LET AD=AD+1: NEX
T L
1090 IF C<>VAL A$(A TO ) THEN  C
LS : BEEP 1,10: PRINT AT 0,0; IN
K 9;"DATA ERROR WITHIN LINE ";LI
NE;"'''''PLEASE CORRECT": STOP
1100 LET LINE=LINE+5: READ A$: I
F A$(1)<>"*" THEN  GO TO 1015
1110 STOP
9995 REM --START OF DATA--
9999 DATA "END OF DATA!"
```

BUT SURELY, THAT'S EASY?


sinclair
ZX Spectrum
BLUE EDIT
RED CAPS LOCK
MAGENTA TRUE VIDEO
GREEN INV. VIDEO
CYAN
YELLOW
WHITE
GRAPHICS
BLACK DELETE
1 ! DEF FN
2 @ FN
3 # LINE
4 $ OPEN #
5 % CLOSE #
6 & MOVE
7 ' ERASE
8 (POINT
9) CAT
0 _ FORMAT
SIN COS TAN INT RND STR $ CHR $ CODE PEEK TAB
Q <= PLOT ASN
W <> DRAW ACS
E >= REM ATN
R < RUN VERIFY
T > RAND MERGE
Y AND RETURN [
U OR IF]
I AT INPUT IN
O ; POKE OUT
P " PRINT ©
READ RESTORE DATA SGN ABS SQR VAL LEN USR
A STOP NEW ~
S NOT SAVE |
D STEP DIM \
F TO FOR {
G THEN GOTO }
H ↑ GOSUB CIRCLE
J - LOAD VAL $
K + LIST SCREEN $
L = LET ATTR
ENTER
LN EXP L PRINT L LIST BIN IN KEY $ PI
CAPS SHIFT
Z : COPY BEEP
X £ CLEAR INK
C ? CONT PAPER
V / CLS FLASH
B * BORDER BRIGHT
N , NEXT OVER
M . PAUSE INVERSE
SYMBOL SHIFT
BREAK SPACE



0095	'?'	R	N	'D'	I	N	K	E
009D	Y	'$'	P	'I'	F	'N'	P	O
00A5	I	N	'T'	S	C	R	E	E
00AD	N	'$'	A	T	T	'R'	A	'T'
00B5	T	A	'B'	V	A	L	'$'	C
00BD	O	D	'E'	V	'A'	L'	L	E
00C5	'N'	S	I	'N'	C	O	'S'	T
00CD	A	'N'	A	S	'N'	A	C	'S'
00D5	A	T	'N'	L	'N'	E	X	'P'
00DD	I	N	'T'	S	Q	'R'	S	G
00E5	'N'	A	B	'S'	P	E	E	'K'
00ED	I	'N'	U	S	'R'	S	T	R
00F5	'$'	C	H	R	'$'	N	O	'T'
00FD	B	I	'N'	O	'R'	A	N	'D'
0105	<	'='	>	'='	<'	>'	L	I
010D	N	'E'	T	H	E	'N'	T	'O'
0115	S	T	E	'P'	D	E	F	
011D	F	'N'	C	A	'T'	F	O	R
0125	M	A	'T'	M	O	V	'E'	E
012D	R	A	S	'E'	O	P	E	N
0135		'#'	C	L	O	S	E	
013D	'#'	M	E	R	G	'E'	V	E
0145	R	I	F	'Y'	B	E	E	'P'
014D	C	I	R	C	L	'E'	I	N
0155	'K'	P	A	P	E	'R'	F	L
015D	A	S	'H'	B	R	I	G	H
0165	'T'	I	N	V	E	R	S	'E'
016D	O	V	E	'R'	O	U	'T'	L
0175	P	R	I	N	'T'	L	L	I
017D	S	'T'	S	T	O	'P'	R	E
0185	A	'D'	D	A	T'	A'	R	E
018D	S	T	O	R	'E'	N	E	'W'
0195	B	O	R	D	E	'R'	C	O
019D	N	T	I	N	U	'E'	D	I
01A5	'M'	R	E	'M'	F	O	'R'	G
01AD	O		T	'O'	G	O		S
01B5	U	'B'	I	N	P	U	'T'	L
01BD	O	A	'D'	L	I	S	'T'	L
01C5	E	'T'	P	A	U	S	'E'	N
01CD	E	X	'T'	P	O	K	'E'	P
01D5	R	I	N	'T'	P	L	O	'T'
01DD	R	U	'N'	S	A	V	'E'	R
01E5	A	N	D	O	M	I	Z	'E'
01ED	I	'F'	C	L	'S'	D	R	A
01F5	'W'	C	L	E	A	'R'	R	E
01FD	T	U	R	'N'	C	O	P	'Y'

?	NOT	LLIST
RND	BIN	STOP
INKEY$	OR	READ
PI	AND	DATA
FN	<=	RESTORE
POINT	>=	NEW
SCREEN$	<	BORDER
ATTR	>	CONTINUE
AT	LINE	DIM
TAB	THEN	REM
VAL$	TO	FOR
CODE	STEP	GOTO
VAL	DEF FN	GOSUB
LEN	CAT	INPUT
SIN	FORMAT	LOAD
COS	MOVE	LIST
TAN	ERASE	LET
ASN	OPEN #	PAUSE
ACS	CLOSE#	NEXT
ATN	MERGE	POKE
LN	VERIFY	PRINT
EXP	BEEP	PLOT
INT	CIRCLE	RUN
SQR	INK	SAVE
SGN	PAPER	RANDOMIZE
ABS	FLASH	IF
PEEK	BRIGHT	CLS
IN	INVERSE	DRAW
USR	OVER	CLEAR
STR$	OUT	RETURN
CHR$	LPRINT	COPY

```
  5 CLEAR 63999
 10 LET start=64000: LET len=460
 20 FOR g=start TO start+len STEP 8
 30 LET cs=0
 40 READ a$,ck
 45 PRINT g;"  ";
 50 IF LEN a$<>16 THEN GO TO 100
 55 FOR i=1 TO 16: IF (a$(i)<"0
" OR a$(i)>"F") AND (a$(i)>"9" O
R a$(i)<"A") THEN GO TO 100
 60 NEXT i
 65 FOR i=1 TO 8: LET a=(16*(CO
DE a$(1)-48+(-7 AND a$(1)>"9")))
: LET a=a+(CODE a$(2)-48+(-7 AND
 a$(2)>"9"))
 70 POKE g+(i-1),a: LET cs=cs+a
: PRINT a$( TO 2);: LET a$=a$(3
TO ): NEXT i
 75 PRINT " ";: IF cs<>ck THEN
GO TO 100
 80 PRINT ck;TAB 30;"OK": NEXT
g
 99 PRINT '"FINISHED": STOP
100 PRINT "ERROR !!": STOP
```

David McCandless,
Program Pitstop,
Your Sinclair,
14, Rathbone Place,
London.
W1P 1DE.

5 March 1989

Dear David,

In the November issue of YS there was a program called "Databanker" which could read memory and produce a line of hex. In the article, you said you were surprised the routine had not been written in machine code. Well, may I present an all singing all dancing version witten in mc.

I enclose an explanation sheet, basic hex loader with data & an assembly listing with full comments. Also enclosed is a tape with the basic loader & data recorded on both side. Just use LOAD "" as normal to load the file.

I hope this up to a standard worth printing in program pitstop.

Introduction

This program is based on the program that appeared in the November edition of YS. The basic principle is the same, ie take a look at memory and create a basic line to read the bytes back later.

While the original maybe short and easy to enter (ie in BASIC), its syntax is rather complicated and needs a dedicated hex loader to read it.

So why is this version special ?

Databanker II has the following :
Simple entry syntax,
Written in machine code, even big amounts of memory only take a couple of seconds,
The result can be loaded into any standard hexloader
Unfortunately it only works in 48K mode

Programming

OK here's the downer. The program is made up of 460 bytes at 64000d. Use the hexloader (fig 1) to enter the data, but it might be wise to save a copy of the loader without the bytes for use with the actual lines produced by the program later.

When the hexloader has finished, save the bytes with

```
SAVE "databank2" CODE 64000, 460
```

Alternatively, there is an assembly listing for those with assemblers (Figure 2)

Using the program

If you want to include the hexloader with the new lines, load it before starting the procedure

First clear ramtop to an area that is below the area you want to read & below the program, 29999 is a fairly safe address.

Load the program with LOAD "databank2" CODE, then in direct mode enter the following:

```
RANDOMIZE USR 64000: POKE start,length
```

Where length can be between 1 & 9000 depending on ramtop & any other data in the program area. This is done by using an internal syntax checker that does not conform to using 1 to 255 like BASIC.

If all goes well, the program drops back to BASIC with the 0 OK message. If the program runs out of memory then it will drop back with either 4 Out of memory or G No room for line.

If this happens increase the value of ramtop with CLEAR or delete a couple of lines. One of these methods will be needed to make room for a save command. The data lines are saved like any other BASIC program.

DATABANKER II

DATABANKER II
© STEVE COLLINS

INSTRUCTIONS

BASED ON AN IDEA SEEN IN

YOUR SINCLAIR

+++PROGRAMMING+++PROGRAMMING+++

PROGRAM PITSTOP

Goodbye, farewell, adieu, adieu, adieu, To you, and you, and you, and you, and *David McCandless*

This is my last Pitstop. Yes, after a year and a bit of tearing open sixty or so envelopes, sifting through billions of tapes and worn Alphacom listings, manhandling my printer in disgust, having stupid gormless pictures taken of myself, I've finally hung up my hex loader, changed my hair style, and left these few pages in the capable hand of none other than **Jon-Boy Davies**. He will now have to deal with my massive backlog of programs. Hah-hah-hah (evil chuckle).

Anyway, first this month is **Steven Collins**, an excellent programmer from West Sussex, who's submitted a revamped version of the *Data Banker* program printed an epoque ago. After Steven is a brilliant redefine keys routine, written by **Peter Zoeteweij**, which is ideal for tagging onto any games you might happen to be working on. And to cap it all for this month and forever is a masterpiece by **Gary Shepardson**, the first part of which was printed last month (see Back Issues).

DATA BANKER 2

by Steven Collins

Steven Collins has just written the single most revolutionary and visually astounding routine in computer programming history and it's under about 2048 bytes.

Banking

The *Data Banker* will compile all your machine code into hex and equip them with Basic data lines for easy and quick use. Not only that but this program features simple entry syntax, blinding speed (huge programs only take a few seconds to compile), and the result can be loaded into any normal hex loader. And it's simple to use.

Just type in this simple program, RUN it and then save the resulting code to tape with SAVE "banker" CODE 64000,460. Incidentally, lines 10 to 100 of this program are actually all that is required to use the results of the *Banker*. So when you get to tabulating your own code, you'll need to reproduce those lines.

Using It

Use this statement to work the program:

RANDOMIZE USR 64000: POKE start, length

The "start" and "length" variables should be replaced with the start address and length of the machine code you want to tabulate respectively.

So if you wanted to put the *Data Banker's* machine code into DATA statements itself, then you'd use:

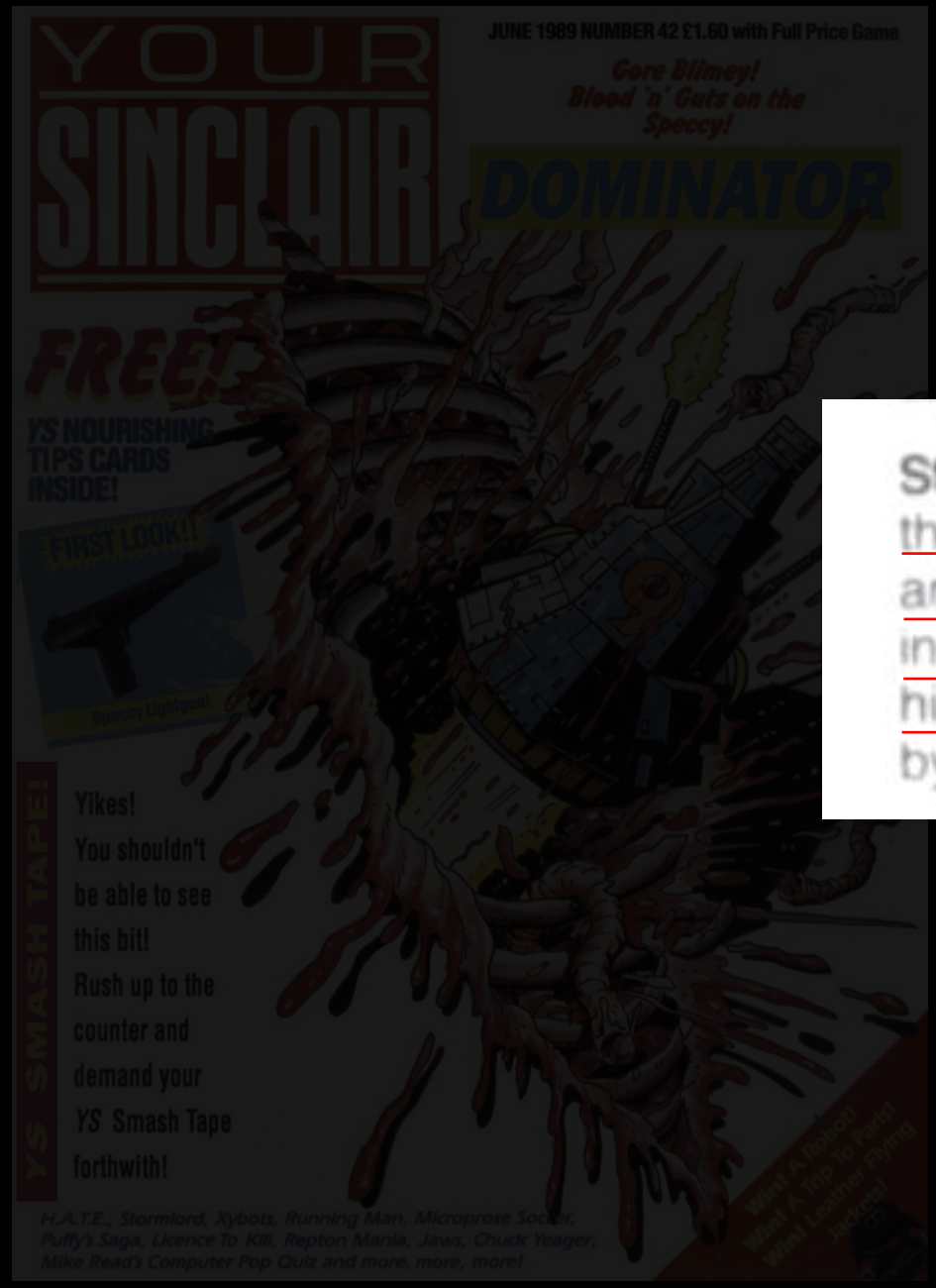
RANDOMIZE USR 64000: POKE 64000,460

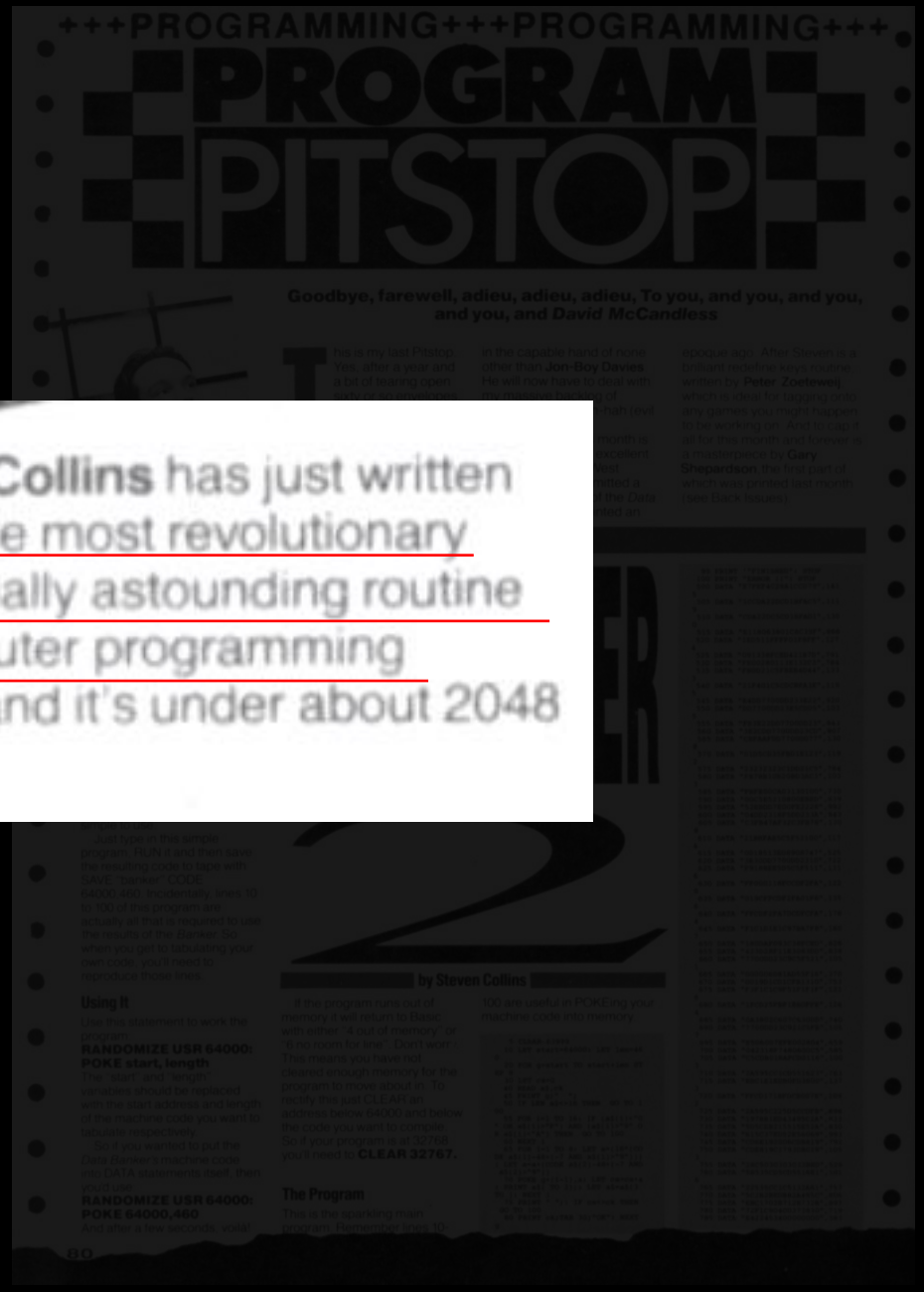
And after a few seconds, voilà!

If the program runs out of memory it will return to Basic with either "4 out of memory" or "6 no room for line". Don't worry. This means you have not cleared enough memory for the program to move about in. To rectify this just CLEAR an address below 64000 and below the code you want to compile. So if your program is at 32768 you'll need to **CLEAR 32767.**

The Program

This is the sparkling main program. Remember lines 10-100 are useful in POKEing your machine code into memory.



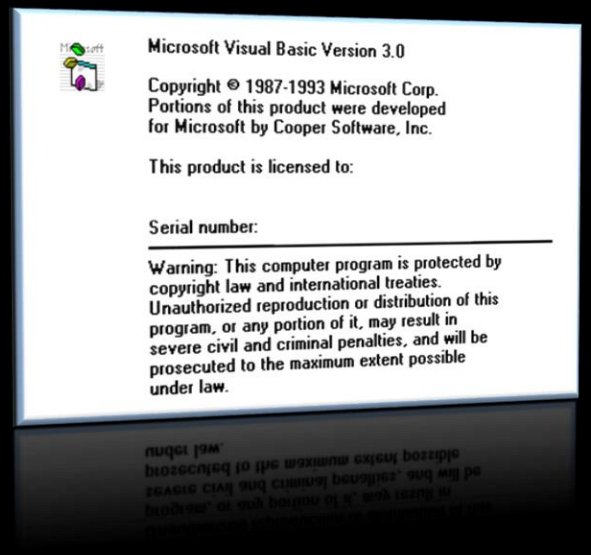


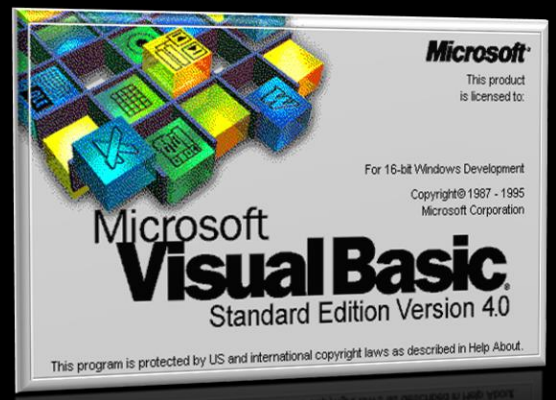
Steven Collins has just written the single most revolutionary and visually astounding routine in computer programming history and it's under about 2048 bytes.

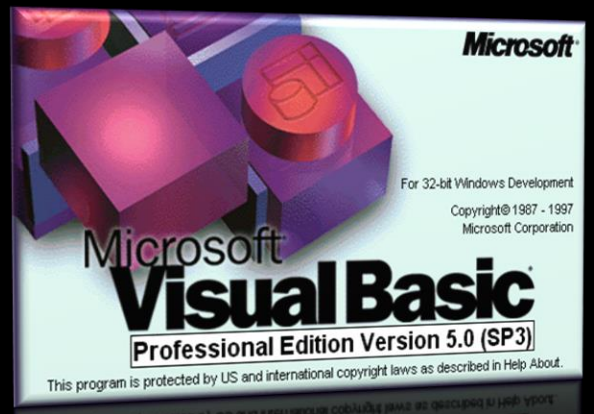
Some source or metadata to read from

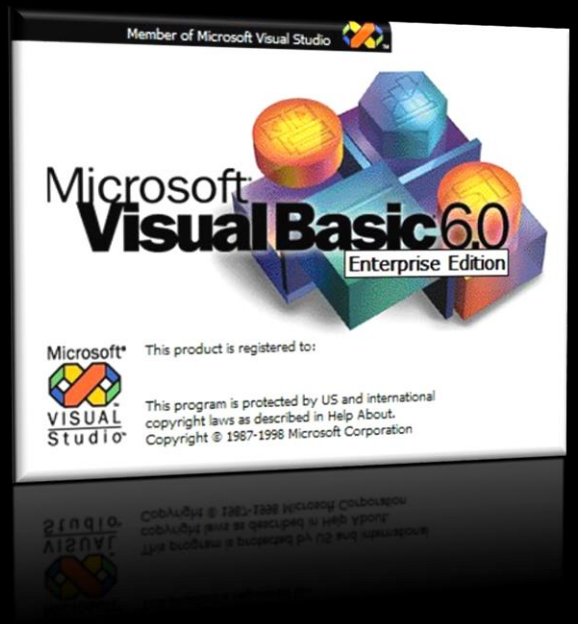
Syntax of target language

Send to the target interpreter/compiler


Microsoft Visual Basic Version 3.0
Copyright © 1987-1993 Microsoft Corp.
Portions of this product were developed
for Microsoft by Cooper Software, Inc.
This product is licensed to:
Serial number:
Warning: This computer program is protected by
copyright law and international treaties.
Unauthorized reproduction or distribution of this
program, or any portion of it, may result in
severe civil and criminal penalties, and will be
prosecuted to the maximum extent possible
under law.


Microsoft
This product
is licensed to:
For 16-bit Windows Development
Copyright© 1987 - 1995
Microsoft Corporation
Microsoft
Visual Basic
Standard Edition Version 4.0
This program is protected by US and international copyright laws as described in Help About.


Microsoft
For 32-bit Windows Development
Copyright© 1987 - 1997
Microsoft Corporation
Microsoft
Visual Basic
Professional Edition Version 5.0 (SP3)
This program is protected by US and international copyright laws as described in Help About.


Member of Microsoft Visual Studio
Microsoft
Visual Basic 6.0
Enterprise Edition
Microsoft
VISUAL
Studio
This product is registered to:
This program is protected by US and international
copyright laws as described in Help About.
Copyright © 1987-1998 Microsoft Corporation

Presentation Layer

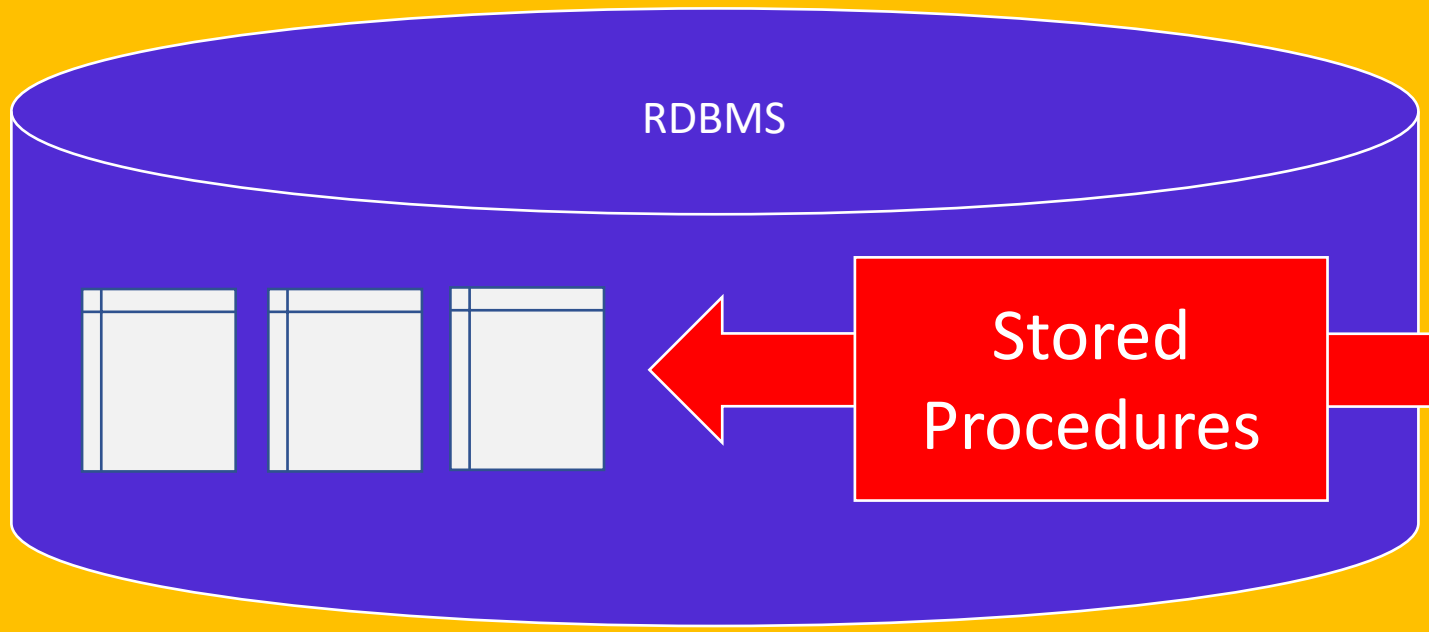
Classic ASP

COM

VB Components

Business Logic

~~ODBC~~ > ~~DAO~~ > ~~RDO~~ > ADO



Data Layer

Business Logic Layer

Utility to Build VB Class to Match

databasejournal.com/features/mssql/article.php/1494191/Utility-to-Build-VB-Class-to-Match-SQL-Stored-Procedures.htm

DATABASE JOURNAL
The Knowledge Center for Database Professionals

MS SQL | Oracle | DB2 | Access | MySQL | PostgreSQL | Sybase | PHP | SQL Etc | SQL Scripts & Samples | Tips | Database Forum | RSS

- » Database Journal Home
- » Database Articles
- » Database Tutorials
  - MS SQL
  - Oracle
  - DB2
  - MS Access
  - MySQL
- » RESOURCES
  - SQL Scripts & Samples
  - Tips
- » Database Forum
- » Slideshows

Free Newsletters:

Enter your Email Address

SUBSCRIBE

News Via RSS Feed

XML RSS

Database Journal | DBA Support | SQLCourse | SQLCourse2 Advertiser Disclosure SEARCH

Featured Database Articles

MS SQL

Posted Oct 9, 2000

Utility to Build VB Class to Match SQL Stored Procedures

By Danny Lesandrini

What is SP2VB?

SP2VB (Stored Proc To VB Class) is a Visual Basic utility that will create the class module code necessary to execute SQL Server Stored Procs with an ADO Command object. In addition to preparing the necessary ADO calls, the utility uses the SQLDMO and SQLNS libraries to query your SQL Server for the names and parameters of all the Stored Procs in your database.

Why the Need for this Utility?

- I hate to type!
- I make mistakes when I type!
- Typing long lists of parameters is boring!

What more need be said?

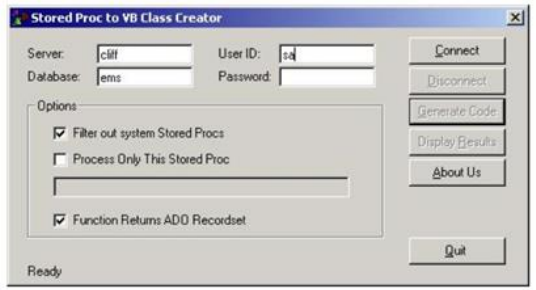
Executing a SQL Server Stored Proc from MS Access or from an ASP page is relatively trivial, but either way, you have to deal with the sometimes long list of input and output parameters. Even when parameters are few, you still want to keep your code consistent accross your entire application, employing a common naming scheme and including at least simple error handling.

Boring? Sometimes! But very necessary and always beneficial.

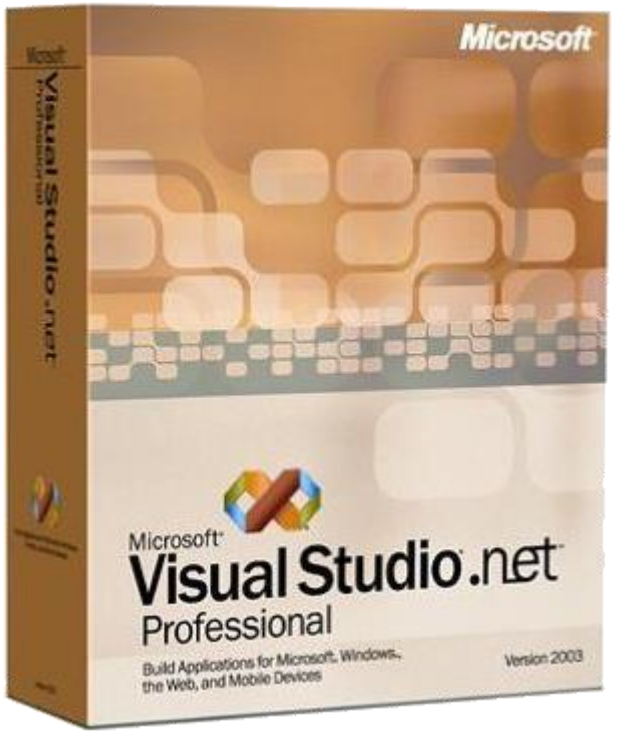
Accordingly, I came up with a plan to automate the task -- a simple code generator. The pages that follow will explain (at relatively high level) what the utility does and how you might modify the code to meet your specific needs. As usual, the source code is available for download so that you can test out the utility and adapt it as you please.

Before we get into the code itself, here's a screen shot of the main form. When we are finished, the tool will be able to perform the following functions:

1. Connect to any SQL Server Database
2. Optionally filter out System Stored Procs (Prefixed with dt_ or sp_)
3. Optionally process only one user-specified procedure
4. Assign the Data Type that the Function returns (Long or ADO Recordset)
5. Display results so user can cut and paste into VB Class module



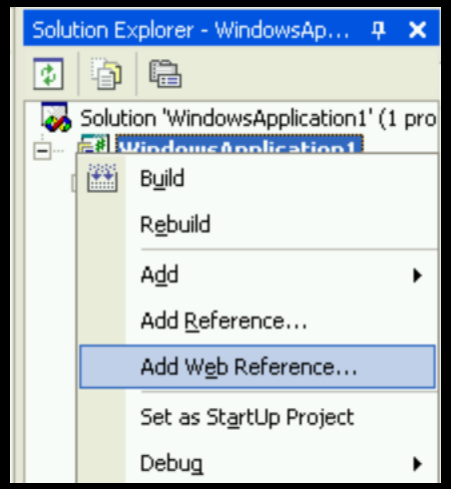
Microsoft®
.net



Web Service Reference

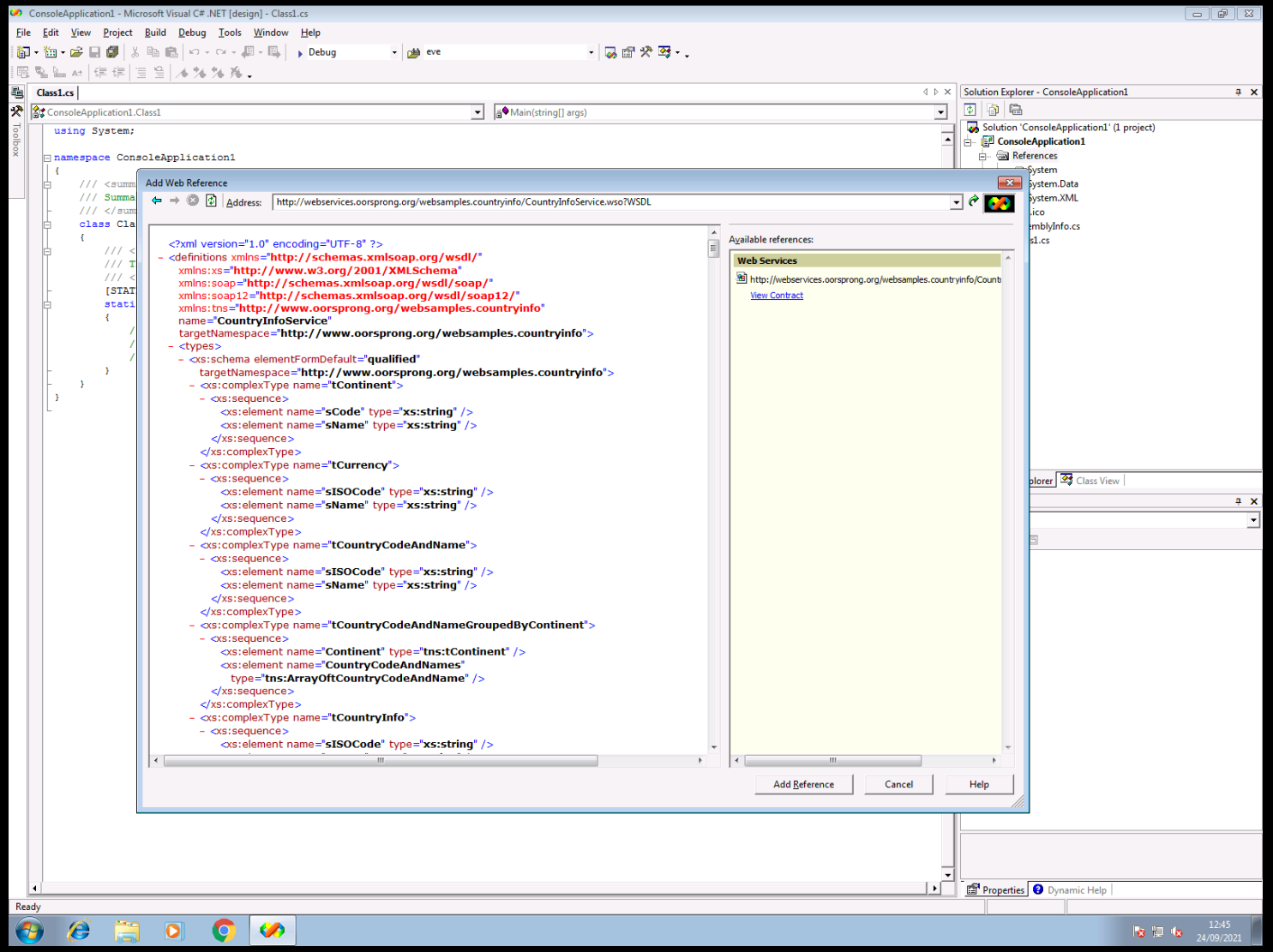
Web Service Reference

- Wizard in Visual Studio to generate C# or VB proxy classes that you use to communicate with XML Soap Services
- Later command line tool (SvcUtil) when WCF introduced
- Interrogates WSDL files or a service project for definition of service and entities

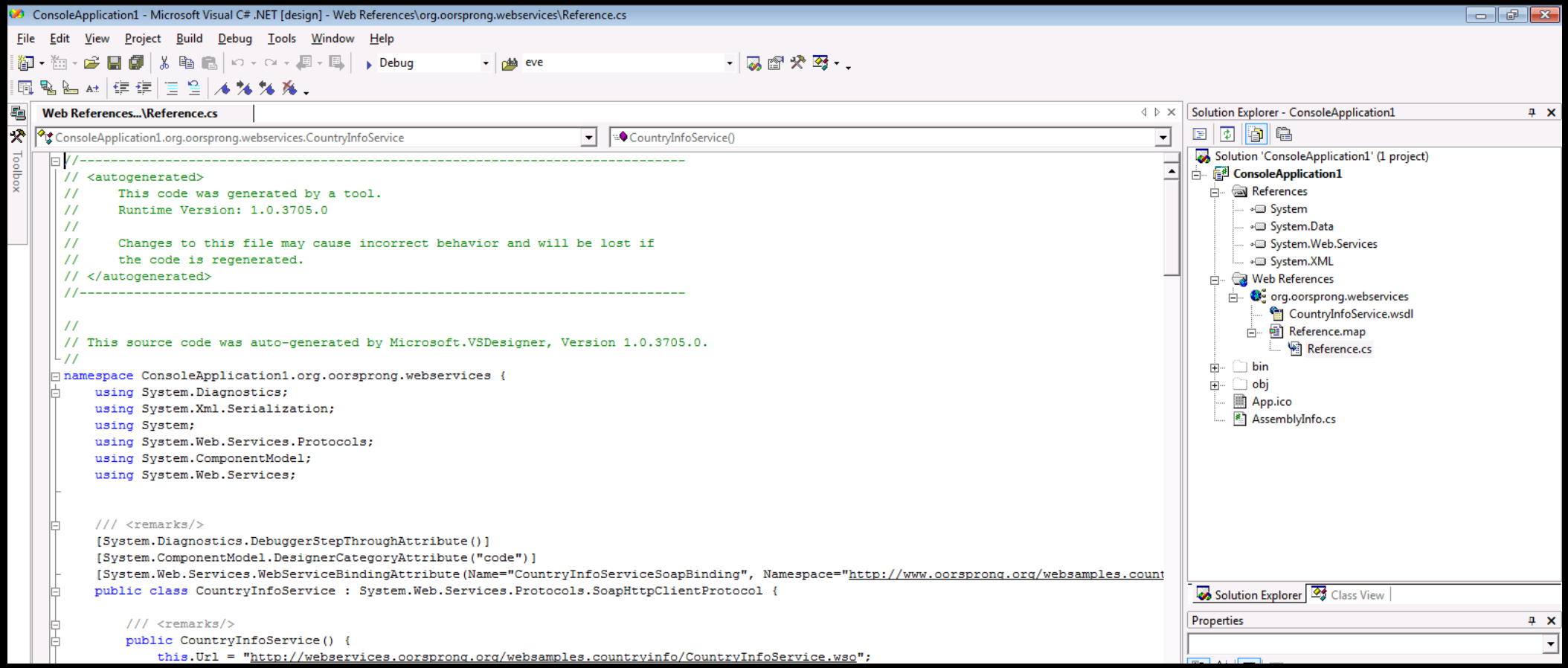



```
Microsoft (R) Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 4.6.1590.0]
Copyright (c) Microsoft Corporation.  All rights reserved.

USES:

 - Generate code from running services or static metadata documents.
 - Export metadata documents from compiled code.
 - Validate compiled service code.
 - Download metadata documents from running services.
 - Pre-generate serialization code.
```

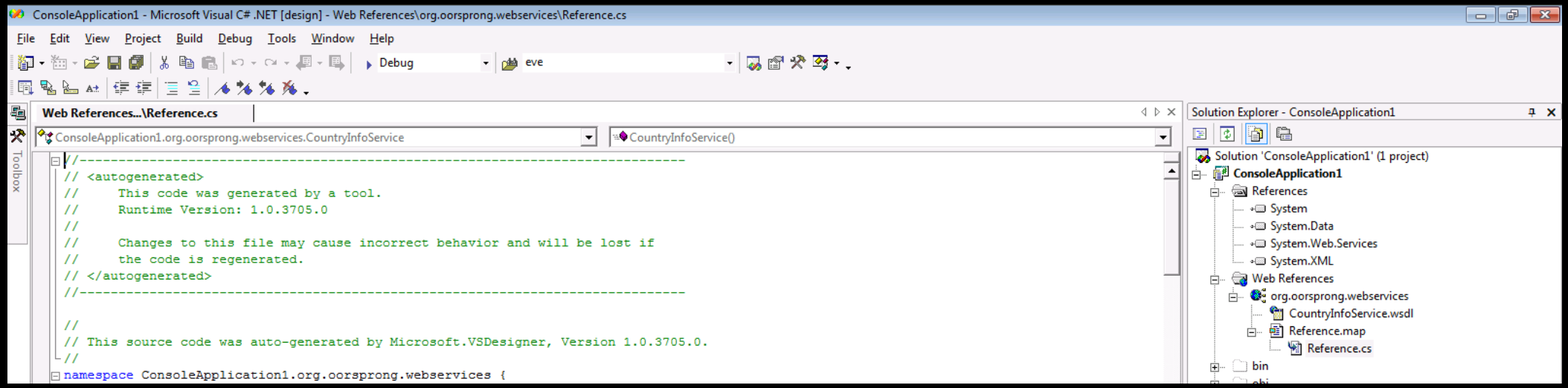
Web Service Reference



Web Service Reference

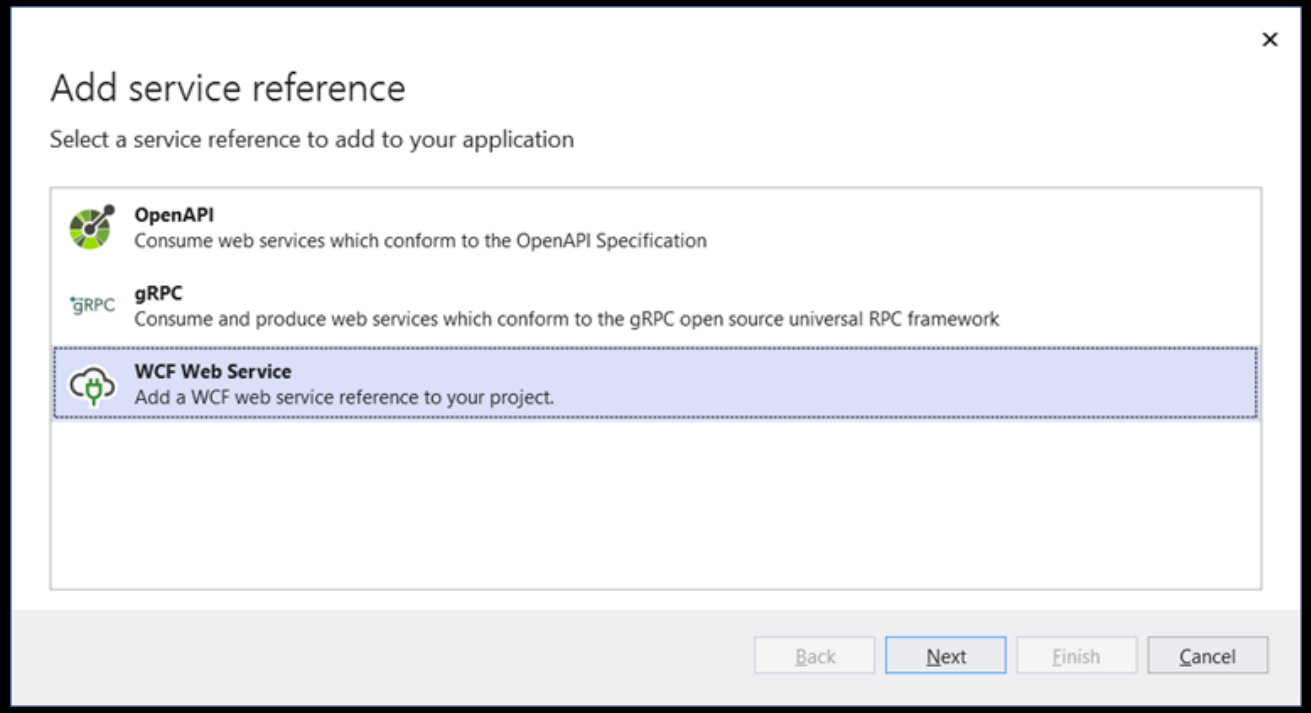


Web Service Reference



- If code manually edited, changes lost if regenerated
- Limited extensibility but some ‘hook’ points
- No control over code quality – warnings in code analysers

Web Service Reference




```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute(tool:"Microsoft.Tools.ServiceModel.Svcutil", version:"2.0.3-preview3.21351.2")]
8 references
public partial class CountryInfoServiceSoapTypeClient :
    System.ServiceModel.ClientBase<ServiceReference1.CountryInfoServiceSoapType>,
    ServiceReference1.CountryInfoServiceSoapType
{
```

Some source or metadata to read from

Syntax of target language

Send to the target interpreter/compiler

Code quality

Immutable

Customisation Hooks

Attributes/Tooling

Influenced changes to C# Language

Influences of Generators on the C# Language

- Partial classes introduced in C# 2
  - Allows for a class to be split into multiple files
  - For classes such as WinForms allows UI part to be split into separate Designer file
  - Means developer can add own code to a generated class without risk of losing it when code generated file is updated

```
namespace WindowsFormsApp1
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
        }

        1 reference
        private void button1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

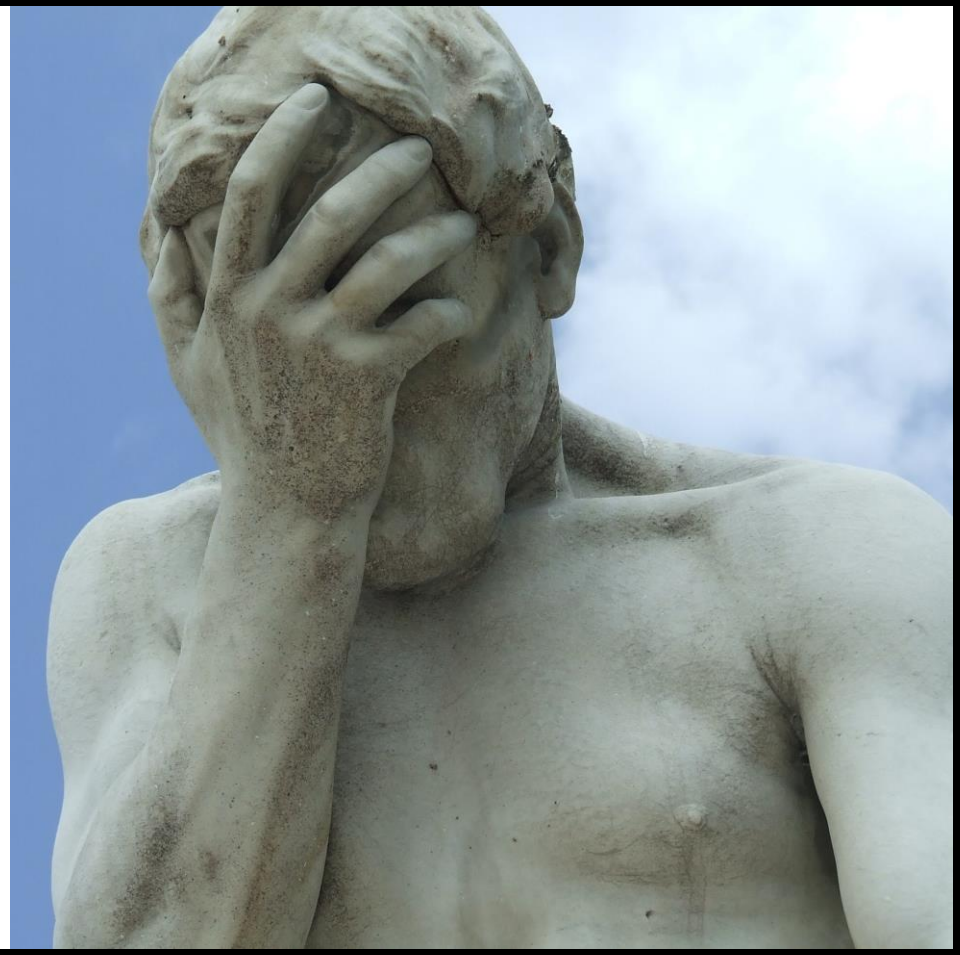
Your Code

```
namespace WindowsFormsApp1
{
    3 references
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        0 references
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows Form Designer generated code

        private System.Windows.Forms.Button button1;
    }
}
```

Generated Code

```
namespace WindowsFormsApp1
{
    3 references
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        0 references
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows Form Designer generated code

        private System.Windows.Forms.Button button1;
    }
}
```

Warning hidden inside #Region



Influences of Generators on the C# Language

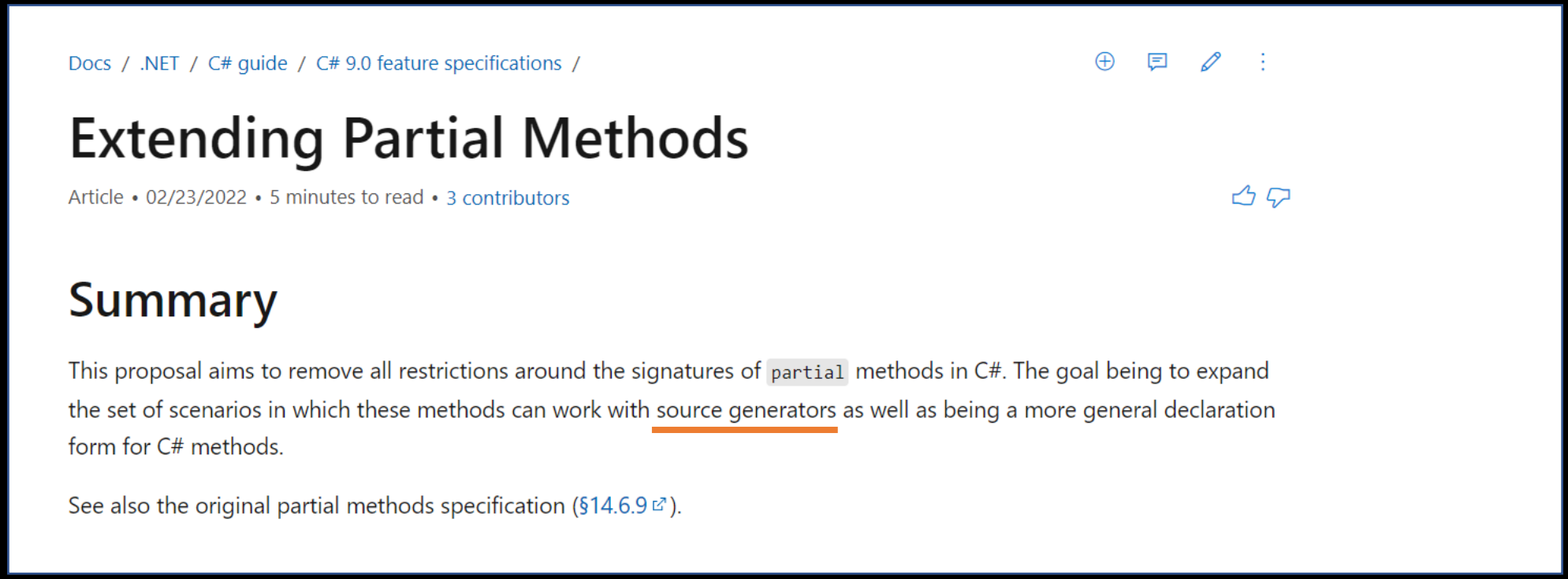
- Partial methods introduced in C# 3 – limited
  - No accessibility modifiers (private only)
  - Only void methods
  - No out parameters

```
public partial class MyClass
{
    2 references
    partial void DoSomething();

}
```

Influences of Generators on the C# Language

- Partial methods extended in C# 9 - Restrictions removed

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-9.0/extending-partial-methods



Docs / .NET / C# guide / C# 9.0 feature specifications /

Extending Partial Methods

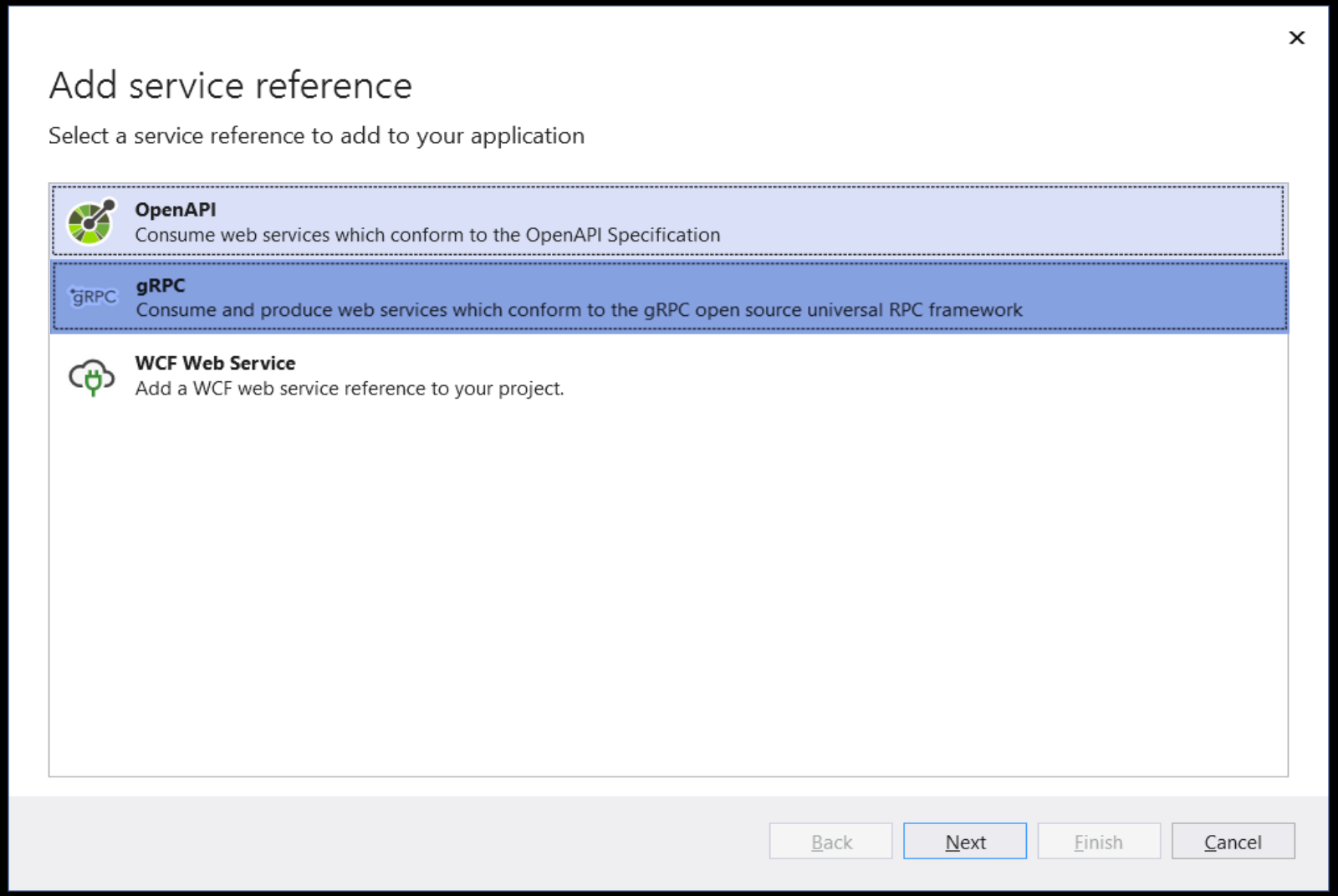
Article • 02/23/2022 • 5 minutes to read • 3 contributors

Summary

This proposal aims to remove all restrictions around the signatures of `partial` methods in C#. The goal being to expand the set of scenarios in which these methods can work with source generators as well as being a more general declaration form for C# methods.

See also the original partial methods specification (§14.6.9).

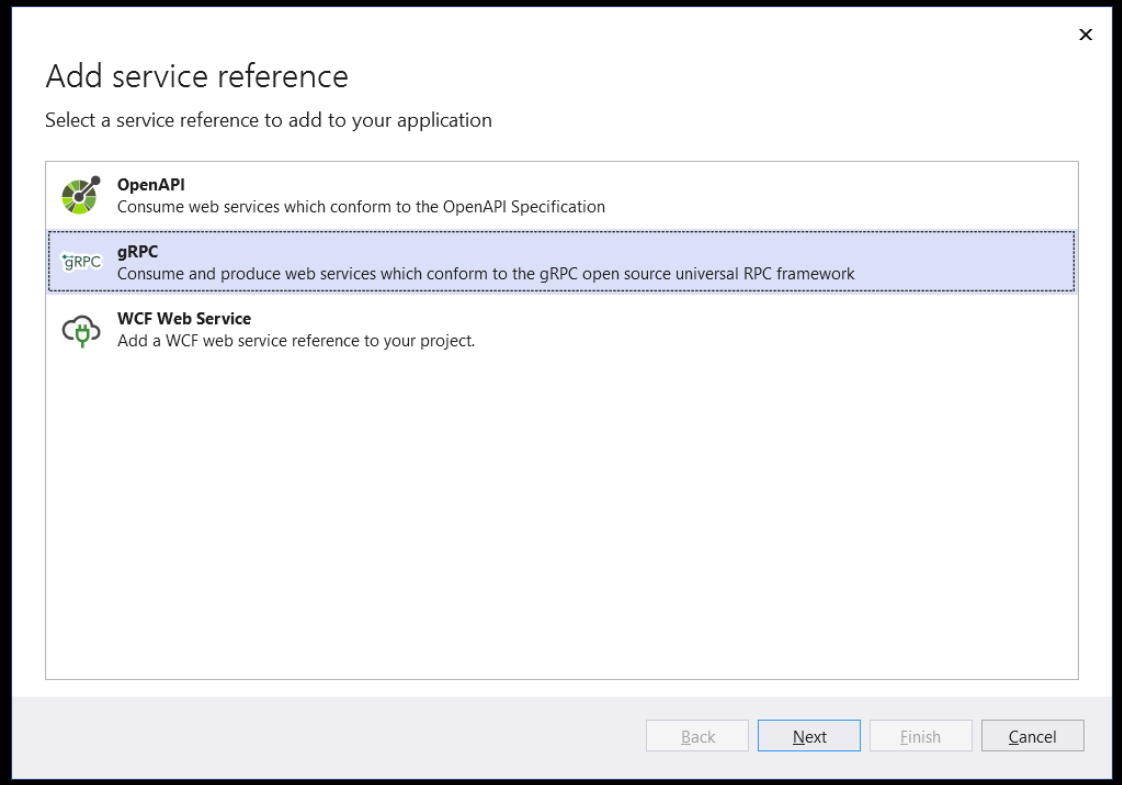
Service References Revisited

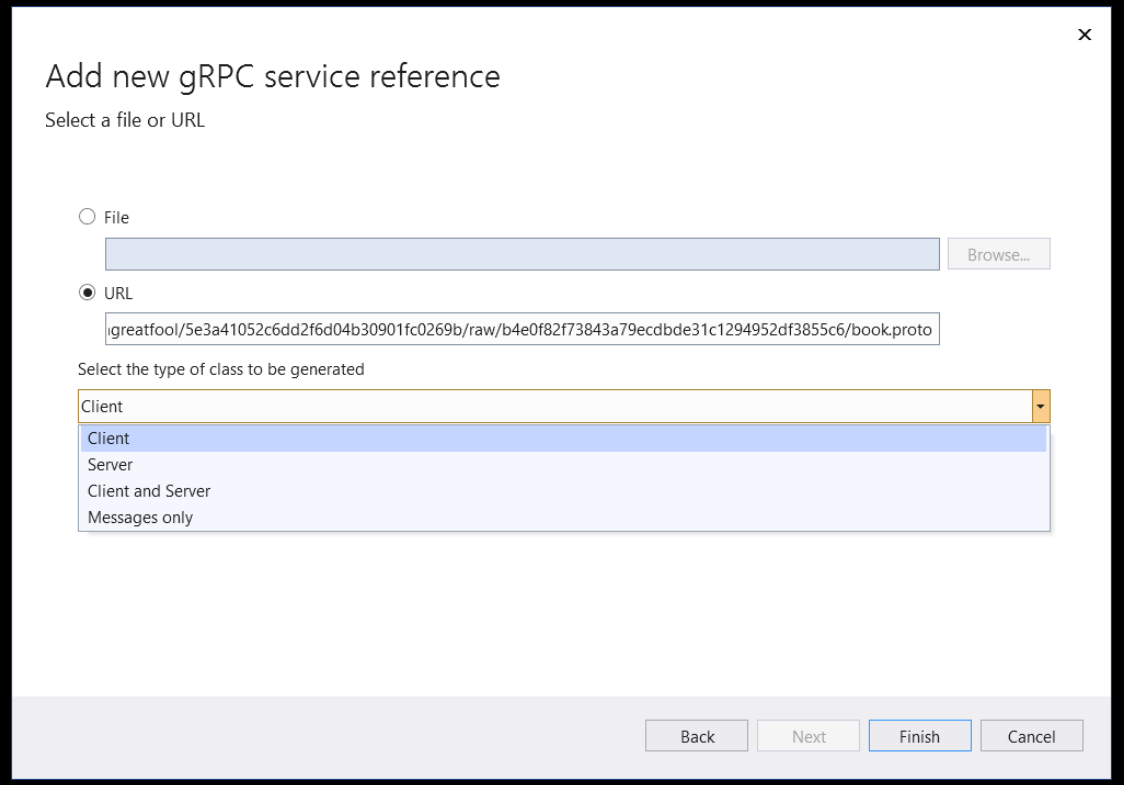


OpenApiReference (.NET 5 >)

- Creates clients for JSON HTTP endpoints
- Generates client proxy based on NSwag
  - for Server, need to use full NSwag tooling
  - https://github.com/RicoSuter/NSwag
- Hooks into the compilation process
- https://stevetalkscode.co.uk/openapireference-commands

Service References Revisited





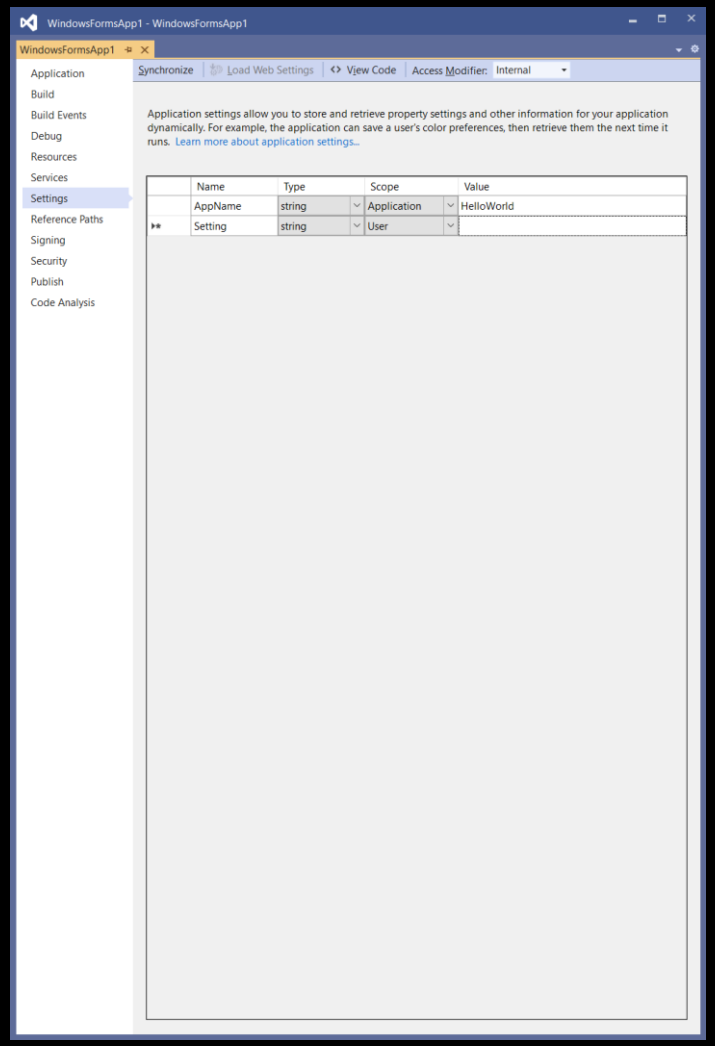
https://docs.microsoft.com/en-us/aspnet/core/grpc/dotnet-grpc?view=aspnetcore-5.0

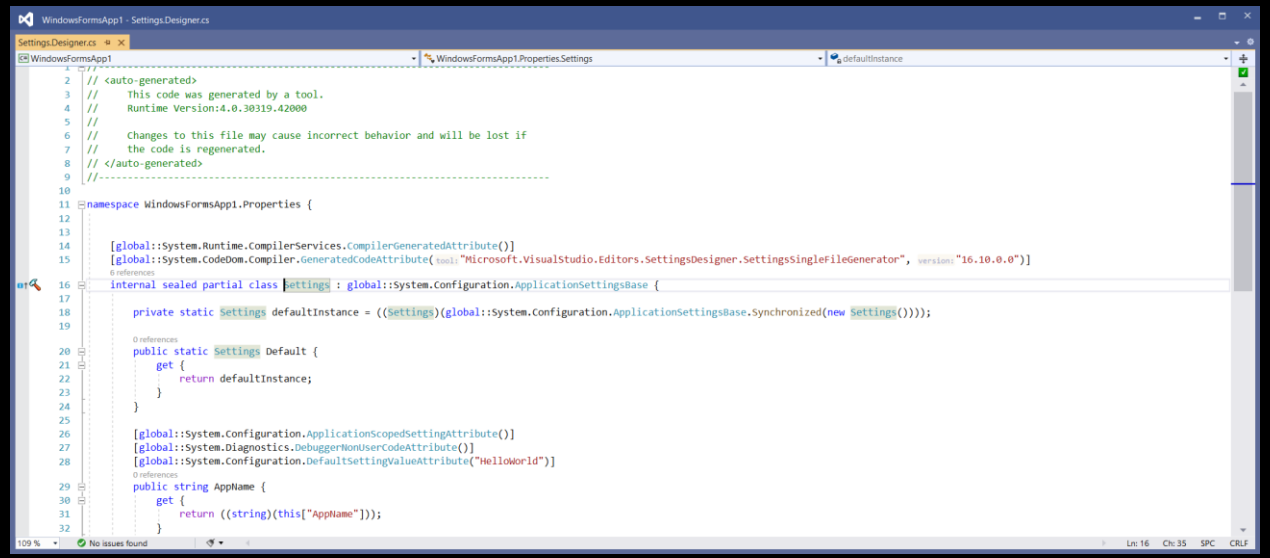
Custom Tool Code Generation

- In Visual Studio, files can have a Custom Tool property set to a code generator

- VS2005 added visual designers for configuration and resources that use Customer Tool Code Generators to generate the following:
  - Settings.settings file – creates a static class representation of configuration files to enable access via model instead of using the ConfigurationManager class
  - Property Resources – creates a static class representation of a RESX resource file

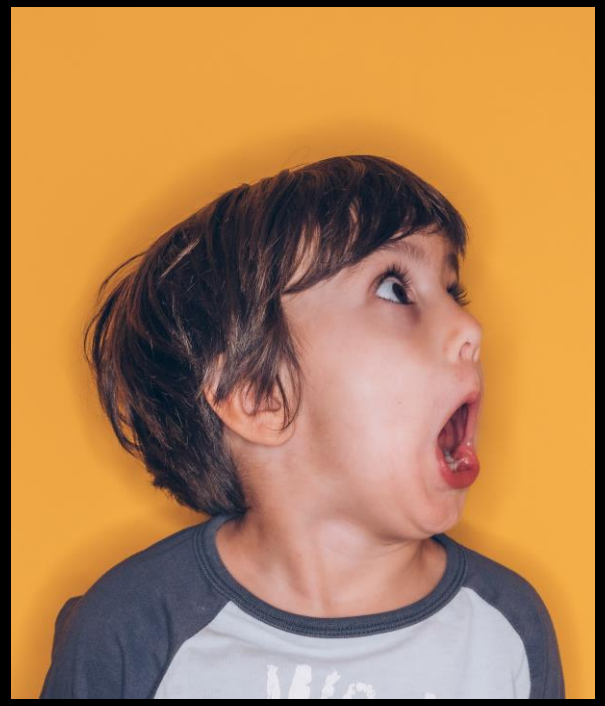
Custom Tool Code Generation





Custom Tool Code Generation

- Can write own code generator, but horrible experience



COM

GUIDs

REGASM

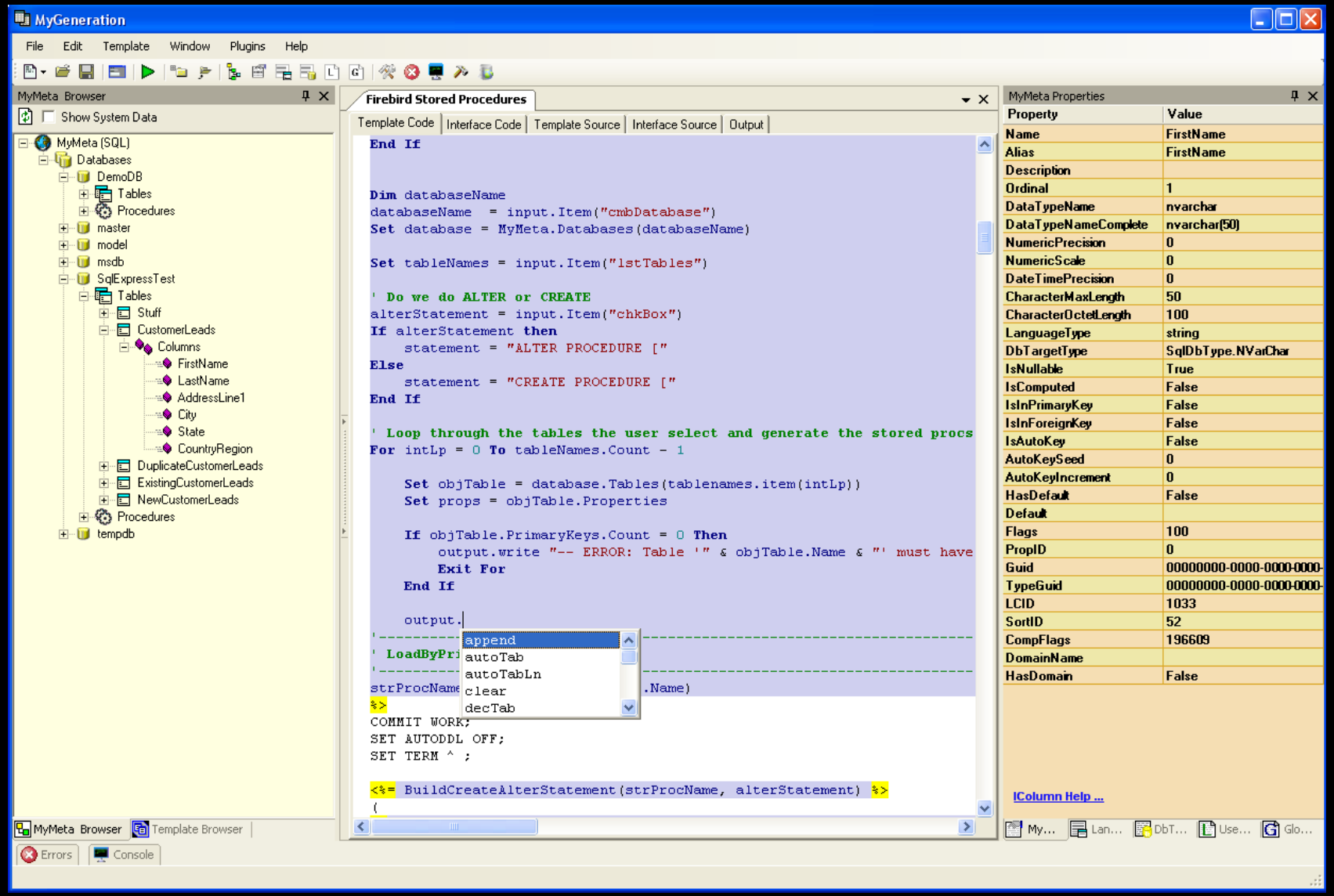
Strong Assembly Naming

Byte Arrays instead of strings

Changes to files need manual code regeneration trigger

Templated Source Code Generation Tools

External Template Tools



T4 – Text Template Transformation Toolkit

- Introduced with Visual Studio 2015
- Template files used with Visual Studio, not the compiler
- Design-time templates – read data from a source and apply that data to the template to result in a file that becomes a source code file
- Visual Studio recognises the template by the Custom Tool property for *.tt files being set to TextTemplatingFileGenerator

T4 – Limitations

- Built for Visual Studio, not the compiler
  - Limited support outside Visual Studio (is supported in Rider!)
- Code executed inside the template must be written using .NET Framework
  - .NET Core / 5 onwards not supported as would need new engine to work with Visual Studio
  - No plans from Microsoft to migrate to this at present
- Can generate output for .NET Core projects since VS2019 16.6

Compile-time and Runtime Code Generators

Reflection

Reflection.Emit

- Generate MSIL at runtime
- Does not cover complete .NET framework
- Security issues

```
ILGenerator generator = methodSum.GetILGenerator();
generator.DeclareLocal(typeof(System.Single));
generator.Emit(OpCodes.Ldarg_1);
generator.Emit(OpCodes.Ldarg_2);
generator.Emit(OpCodes.Add_Ovf);
generator.Emit(OpCodes.Conv_R4);
generator.Emit(OpCodes.Stloc_0);
generator.Emit(OpCodes.Ldloc_0);
generator.Emit(OpCodes.Ret);
```

Regular Expressions

- Translate a regular expression string into code that can check for matches
- Default behaviour is to interpret the expression at runtime and generate MSIL to be executed by the JIT engine
- Option to cache the MSIL using the Compiled option
- Option to compile the MSIL to an assembly file (Framework Only)

https://referencesource.microsoft.com/#system/regex/system/text/regularexpressions/RegexCompiler.cs

Aspect Orientated Programming (AOP)

- IL Weaving – generates Microsoft Intermediate Language, not C# code
- Takes control over the whole compilation process
- Code is inserted at IL level so needs special tooling to debug as no source
- Ability to rewrite your code during compilation

AOP Libraries

- Fody (https://github.com/Fody/Fody)
  - Open source, but ask for financial contribution
- PostSharp (https://www.postsharp.net)
  - Commercial product
- MetaLama
  - Next generation of PostSharp based on Roslyn compiler fork

Expression Trees (C# 3 Onwards)

- Represent code in a tree structure where each node is an expression
- Basis for LINQ providers and working with Dynamic Language Runtime (DLR)
- Usually created from lambda expressions
- Can hand craft expression trees using the API … verbose and hard!
- Not just .NET code – used by Entity Framework to create SQL queries
- Compiled and executed at runtime
- https://docs.microsoft.com/en-us/dotnet/csharp/expression-trees

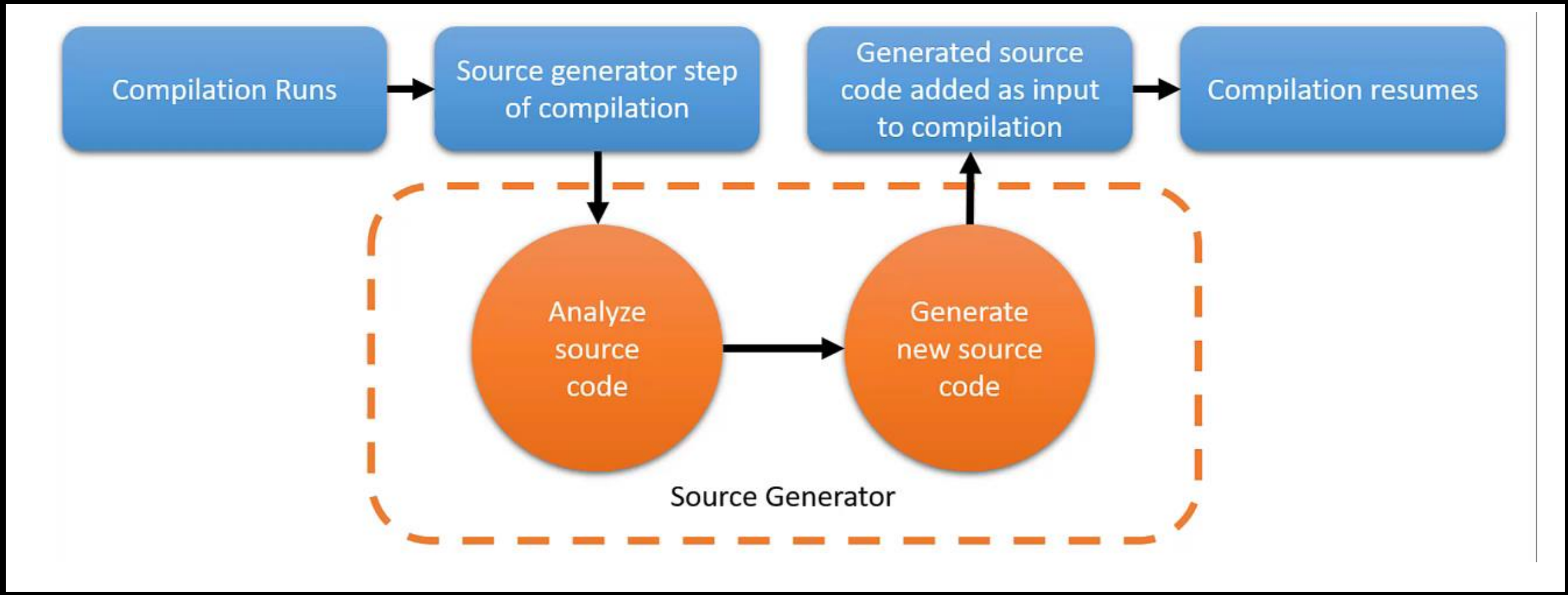
Object Relational Mappers - Entity Framework

Hybrid of mechanisms

- Inspection of SQL metadata to create .NET classes to represent tables (Database First Design)
- Code inspection of .NET classes to generate SQL table creation statements (Code First Design)
- Expression tree engine to generate SQL queries from LINQ
- Reflection to bind SQL data results to .NET entity instances

I Just Want To Generate Some Code!

- Problems with tools described so far.
  - Can be complex to understand and/or write
  - Can be complex to use
  - Some are external tools that run outside Visual Studio
  - Some are bound to and only work with Visual Studio
  - Some are limited to .NET Framework (not Core or 5+)
  - Some are closed source so limited in changing what code gets generated

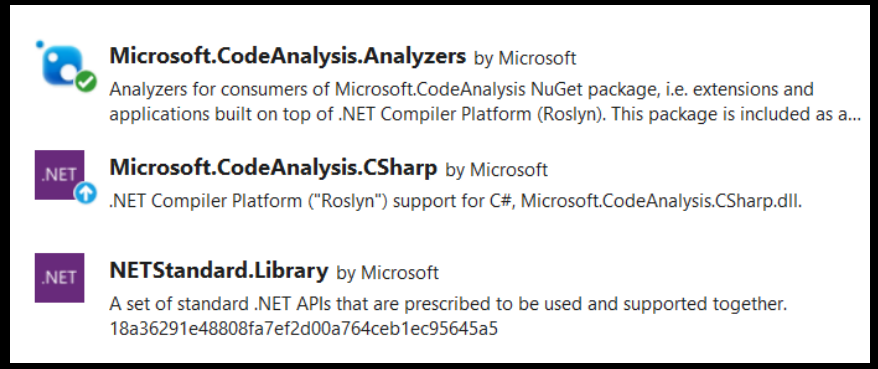


https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/source-generators-overview

Rosyln Source Generators

- Easy to write simple generator … but can get very complex!
- Input can be code syntax tree or additional input files
- The output is just the result of building a string
- No need to use template files (can if you want to, but slows compilation)
- Can reference as project in same solution or use a shared NuGet package

Rosyln Source Generators



.NET Standard 2.0 Only

Useful for debugging in Visual Studio

Generator Attribute

```
using Microsoft.C  nalysis;

namespace Debugg  gSourceGenerators
{
    [Generator]
    public class SimpleCodeGenerator : ISourceGenerator
    {
        public void Initialize(GeneratorInitializationContext context)
        {
        }

        public void Execute(GeneratorExecutionContext context)
        {
        }
    }
}
```

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFrameworks>netstandard2.0</TargetFrameworks>
    <IsRoslynComponent>true</IsRoslynComponent>
    <Configurations>Debug;Release;DebugGenerator</Configurations>
  </PropertyGroup>

  <PropertyGroup Condition="'$(Configuration)|$(TargetFramework)|$(Platform)'=='DebugGenerator|netstandard2.0|AnyCPU'">
    <DefineConstants>TRACE;DEBUG;DEBUGGENERATOR</DefineConstants>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.CodeAnalysis.Analyzers" Version="3.3.2">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="System.CodeDom" PrivateAssets="all" GeneratePathProperty="true" Version="5.0.0" />
    <PackageReference Include="Microsoft.CodeAnalysis.CSharp" Version="3.9.0" />
  </ItemGroup>
</Project>
```

Rosyln Source Generators

- Requires [Generator] attribute
- Two key methods required when implementing ISourceGenerator
  - Initialize - receives a GeneratorInitialzationContext
  - Execute - receives a GeneratorExecutionContext

```
using Microsoft.CodeAnalysis;

namespace DebuggingSourceGenerators
{
    [Generator]
    public class SimpleCodeGenerator : ISourceGenerator
    {
        public void Initialize(GeneratorInitializationContext context)
        {
        }

        public void Execute(GeneratorExecutionContext context)
        {
        }
    }
}
```

Rosyln Source Generators (.NET 5 Version)

- Initialize method
  - Register an interest in listening to syntax changes in the compiler chain
  - Causes the generator to be called every time a compilation takes place
- Execute method
  - Creates and emits the generated source code to be added
  - Build up a string either
    - Based on content of syntax and semantic trees
    - From data in ‘additional files’ such as JSON, CSV etc. using your own parsing logic
    - Using Roslyn APIs to generate source code from code document model
  - Last action of method is to add the source string to the compile context for the compiler to include

Rosyln Source Generators (.NET 5 Version)

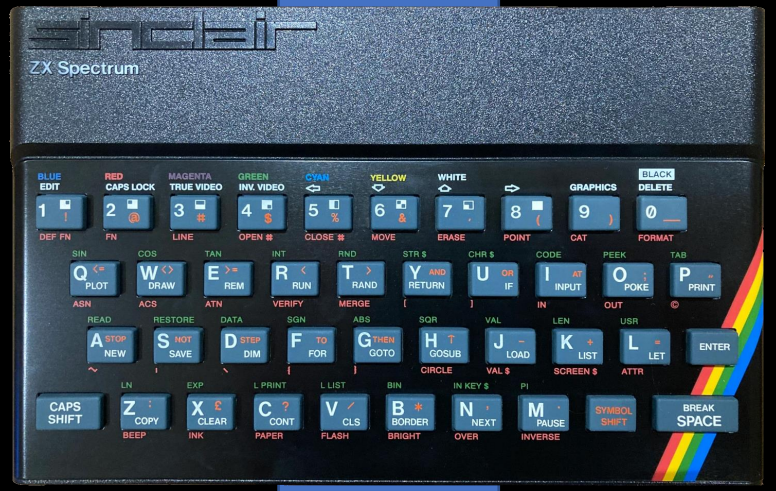
- `ISourceGenerator` in .NET 5 can be slow!
  - Methods called on each generation – potentially every keystroke in IDE!
  - Really slow in very large projects

Rosyln Source Generators (.NET 6 Version)

- Introduction of `IIncrementalGenerator`
  - Only Initialize method – no Execute
  - Build a pipeline to determine if generation required by creating a SyntaxProvider instance
  - Syntax provider uses predicates to act as a filter, so generation only triggered when required
  - When generation is required, a transformer is used to generate the code to be merged into the compilation
  - If syntax has not changed, cached output will be reused

https://github.com/dotnet/roslyn/blob/main/docs/features/incremental-generators.md

MEMORY



SINCLAIR BASIC

IT'S STILL ALL ABOUT SYNTAX

C#

OTHER FILES

.NET 6

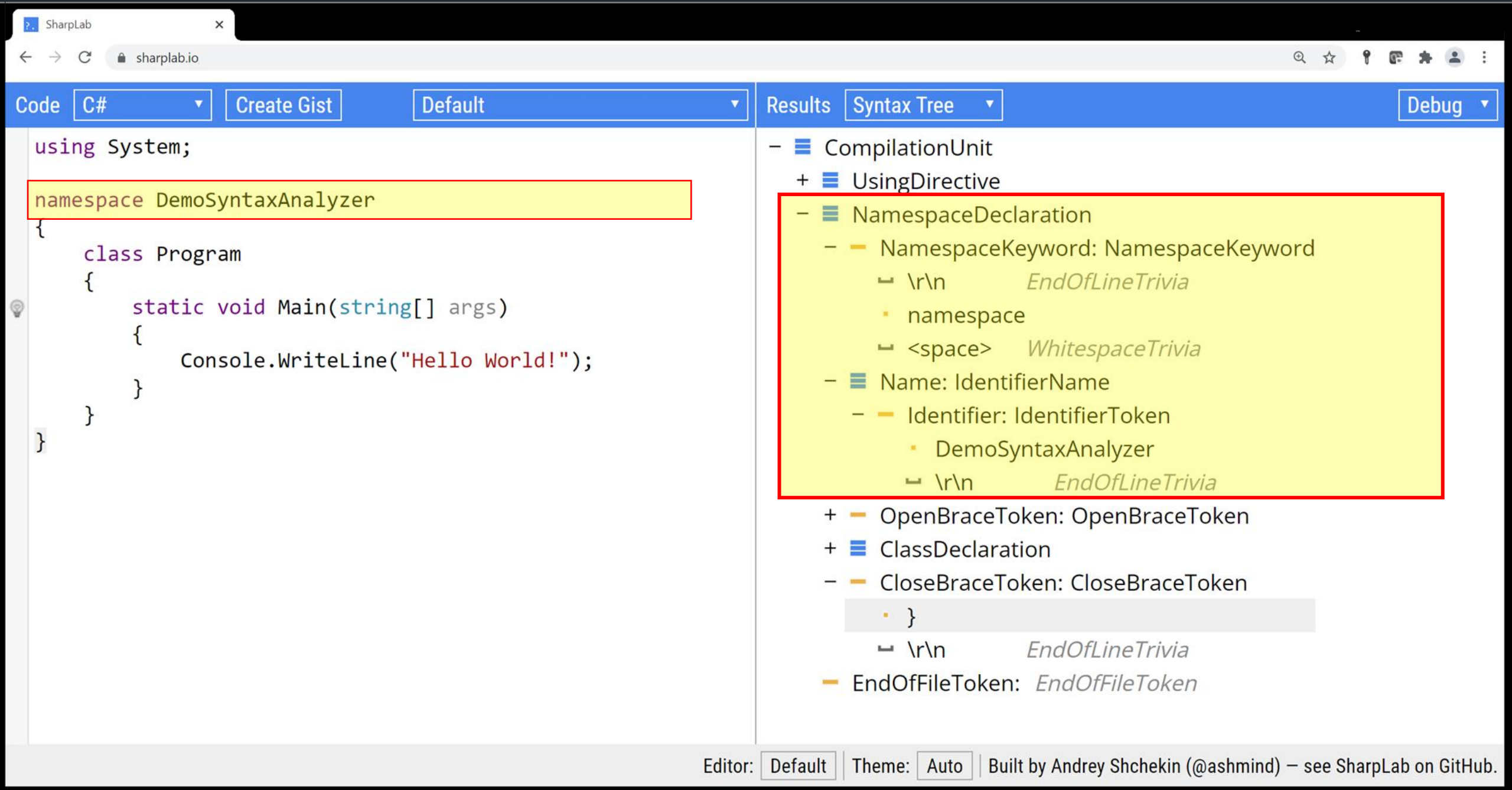
C#

Syntax Trees

Syntax Trivia
Formatting (space/tab/newline)
Comments
Preprocessor directives

Syntax Tokens
Keywords
Identifiers
Literals
Punctuation

Syntax Nodes
Declarations
Statements
Clauses
Expressions


SharpLab
sharplab.io
Code
C#
Create Gist
Default
Results
Syntax Tree
Debug
using System;
namespace DemoSyntaxAnalyzer
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine("Hello World!");
}
}
}
CompilationUnit
UsingDirective
NamespaceDeclaration
NamespaceKeyword: NamespaceKeyword
\r\n EndOfLineTrivia
namespace
<space> WhitespaceTrivia
Name: IdentifierName
Identifier: IdentifierToken
DemoSyntaxAnalyzer
\r\n EndOfLineTrivia
OpenBraceToken: OpenBraceToken
ClassDeclaration
CloseBraceToken: CloseBraceToken
}
\r\n EndOfLineTrivia
EndOfFileToken: EndOfFileToken
Editor: Default
Theme: Auto
Built by Andrey Shchekin (@ashmind) – see SharpLab on GitHub.

Syntax and Semantic Trees

- Syntax analysis is about the structure of code
  - Looking for text to trigger the code generation
- Semantic analysis give that code meaning
  - Generated from syntax tree nodes
  - Similar to using Reflection to inspect an assembly, but at compile time
  - Understands references to namespaces, types, methods, properties
  - Used a lot in Roslyn Analysers
- Powerful, but complex
  - https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/get-started/semantic-analysis

Generate from Additional Files

- Reference to a file that is not a C# source file
- Needs to be included in the project
- Could be whatever you want – JSON, CSV, your own DSL
- The file is watched for updates to trigger a recompilation

Roslyn Source Generators - Why Use Them?

- Part of the compiler chain not tied to Visual Studio
- Generates code from your code as it is compiled, not reading files … but can read ‘Additional Files’ as a source such as CSV, JSON, own DSL, whatever you want if you can parse the file content
- Generate code at compile time instead of using reflection at runtime
- Can view and debug the generated code

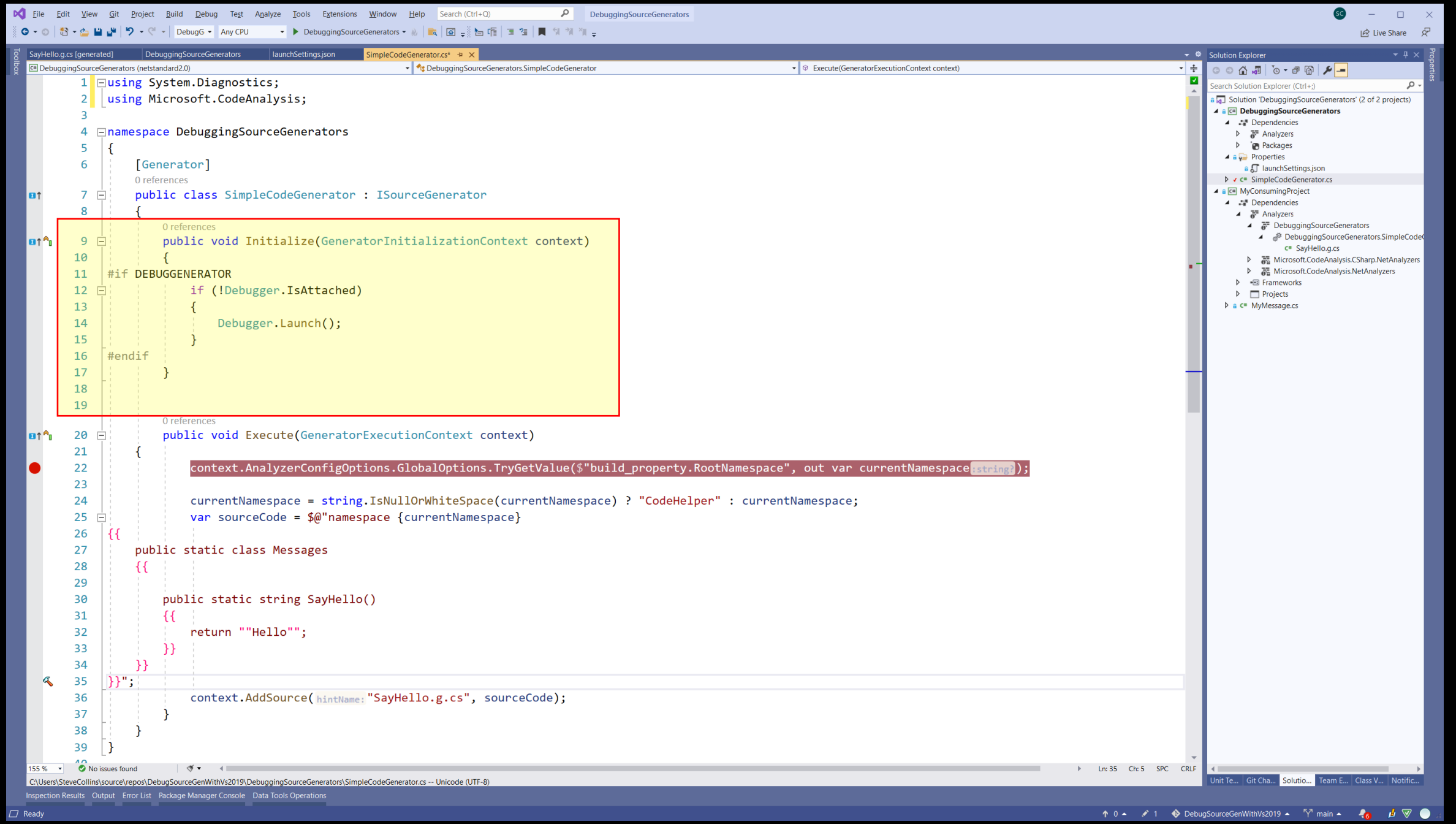
Roslyn Source Generators - Gotchas

Not a perfect world

- The generator project must have a .NET Standard 2.0 target
  - Can dual target with .NET 6, must always have .NET Standard 2.0 target with Visual Studio as well as Roslyn
- Can in theory use .NET Standard 2.1
  - Works with dotnet build or dotnet msbuild but does not work with .NET Framework MSBuild tooling
- Only use code or local files as metadata source
  - Do not use slow I/O as a source (such as databases and HTTP calls) due to latency and availability (offline/permissions) that will cause compilation to break
- Can only generate C# or VB code code not text files
  - Not text files - open proposal for changes in Roslyn to support embedded resources and non-code artefacts https://github.com/dotnet/roslyn/issues/49935
  - F# not supported as does not use Roslyn for compilation

C# 9 Source Generators - Debugging

- Debugging experience was very poor when first introduced
- Better since VS2019 V16.10 (and carried on in VS 2022)
  - https://stevetalkscode.co.uk/debug-source-generators-with-vs2019-1610
- Problem with Visual Studio caching the generator
  - needs restart of VS each time generator code is changed to debug it
- Alternative is to create a custom build profile
  - Use #IF directive to attach and launch the debugger when the generator gets invoked




```
1  using System.Diagnostics;
2  using Microsoft.CodeAnalysis;
3
4  namespace DebuggingSourceGenerators
5  {
6      [Generator]
       0 references
7      public class SimpleCodeGenerator : ISourceGenerator
8      {
           0 references
9          public void Initialize(GeneratorInitializationContext context)
10         {
11 #if DEBUGGENERATOR
12             if (!Debugger.IsAttached)
13             {
14                 Debugger.Launch();
15             }
16 #endif
17         }
18
19
           0 references
20         public void Execute(GeneratorExecutionContext context)
21     {
22         context.AnalyzerConfigOptions.GlobalOptions.TryGetValue($"build_property.RootNamespace", out var currentNamespace:string?);
23
24         currentNamespace = string.IsNullOrWhiteSpace(currentNamespace) ? "CodeHelper" : currentNamespace;
25         var sourceCode = $@"namespace {currentNamespace}
26 {{
27     public static class Messages
28     {{
29
30         public static string SayHello()
31         {{
32             return ""Hello"";
33         }}
34     }}
35 }}";
36         context.AddSource( hintName: "SayHello.g.cs", sourceCode);
37     }
38 }
39 }
```

Write Unit Tests!

- Make use of Rosyln being available as Compiler As A Service!
- Lot easier than trying to debug in real-time
- Several steps
  - Write the input source code as a string (including library references)
  - Parse it to get a Syntax Tree
  - Use the Syntax Tree to create a compilation
  - Trigger the compilation to invoke the source generator
  - Compare the output of the source generator to the expected result

Other resources

- List of Source Generator Projects
  - https://github.com/amis92/csharp-source-generators
- Source Generator Cookbook
  - https://github.com/dotnet/roslyn/blob/main/docs/features/source-generators.cookbook.md


.NET
6

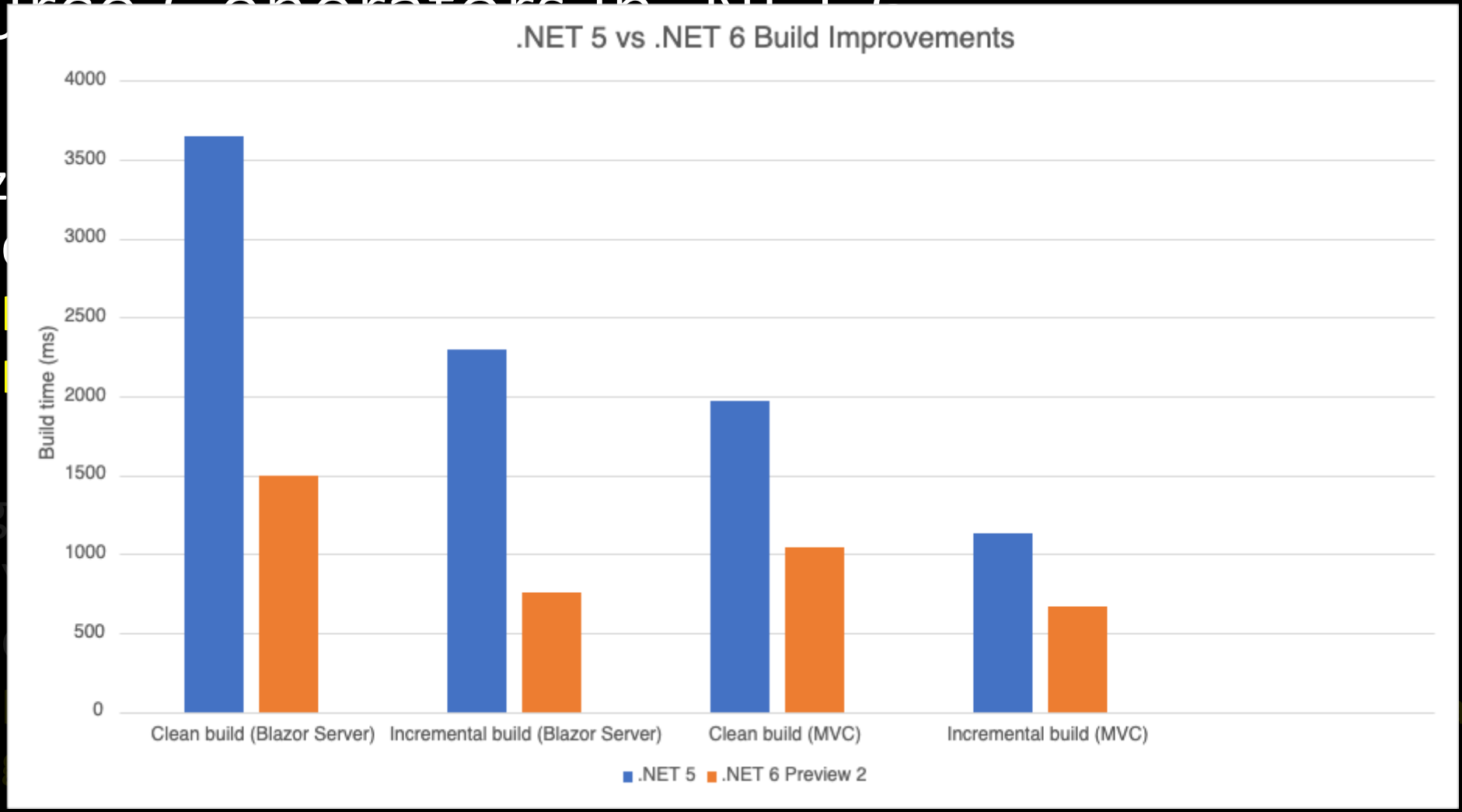
Source Generators in .NET 6

- Razor Compiler
  - On by default, but can turn off in csproj file if you want … by why?
  - https://devblogs.microsoft.com/aspnet/asp-net-core-updates-in-net-6-preview-2/#razor-compiler-updated-to-use-source-generators

- Logging
  - You write partial method and add `[LoggerMessage]` attribute
  - Generator writes boiler plate code
  - https://docs.microsoft.com/en-us/dotnet/core/extensions/logger-message-generator

Sou

- Raz
  - (
  - I
    I
- Log
  - V
  - (
  - I



ge-

Source Generators in .NET 6

- Razor Compiler
  - On by default, but can turn off in csproj file if you want … by why?
  - https://devblogs.microsoft.com/aspnet/asp-net-core-updates-in-net-6-preview-2/#razor-compiler-updated-to-use-source-generators
- Logging
  - You write partial method and add `[LoggerMessage]` attribute
  - Generator writes boiler plate code
  - https://docs.microsoft.com/en-us/dotnet/core/extensions/logger-message-generator

Source Gene

```
public partial class InstanceLoggingExample
{
    private readonly ILogger _logger;

    0 references
    public InstanceLoggingExample(ILogger logger)
    {
        _logger = logger;
    }

    [LoggerMessage(
        EventId = 0,
        Level = LogLevel.Critical,
        Message = "Could not open socket to `{hostName}`")]
    0 references
    public partial void CouldNotOpenSocket(string hostName);
}
```

- Razor Compiler
  - On by default, bu ant … by why?
  - https://devblogs e-updates-in-net-6-preview-2/#razor-compiler-updated-to-use-source-generators
- Loggi
  - You
  - Ge
  - htt ssage-ger

```
partial class InstanceLoggingExample
{
    [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.Extensions.Logging.Generators", "6.0.0.0")]
    private static readonly global::System.Action<global::Microsoft.Extensions.Logging.ILogger,
        global::System.String, global::System.Exception?> __CouldNotOpenSocketCallback =
        global::Microsoft.Extensions.Logging.LoggerMessage.Define<global::System.String>
        (global::Microsoft.Extensions.Logging.LogLevel.Critical,
            new global::Microsoft.Extensions.Logging.EventId(0, nameof(CouldNotOpenSocket)),
            "Could not open socket to `{hostName}`",
            new global::Microsoft.Extensions.Logging.LogDefineOptions() { SkipEnabledCheck = true });

    [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.Extensions.Logging.Generators", "6.0.0.0")]
    3 references
    public partial void CouldNotOpenSocket(global::System.String hostName)
    {
        if (_logger.IsEnabled(global::Microsoft.Extensions.Logging.LogLevel.Critical))
        {
            __CouldNotOpenSocketCallback(_logger, hostName, null);
        }
    }
}
```

Source Generators in .NET 6 – JSON Serializer

Try the new System.Text.Json source generator

Layomi

July 22nd, 2021

In .NET 6.0, we are shipping a new C# source generator to help improve the performance of applications that use `System.Text.Json`. In this post, I'll go over why we built it, how it works, and what benefits you can experience in your application.

With the introduction of the `System.Text.Json` source generator, we now have a few models for JSON serialization in .NET to choose from, using `JsonSerializer`. There is the existing model which is backed by runtime reflection, and two new compile-time source generation modes; where the generator generates optimized serialization logic, a static data access model, or both. In both source-generation scenarios, the generated artifacts are passed directly to `JsonSerializer` as a performance optimization. Here's an overview of the functionality that each serialization model provides:

	`JsonSerializer`	`JsonSerializer` + pre-generating optimized serialization logic	`JsonSerializer` + pre-generating data access model
Increases serialization throughput	No (baseline)	Yes	No
Reduces start-up time	No (baseline)	Yes	Yes
Reduces private memory usage	No (baseline)	Yes	Yes
Eliminates runtime reflection	No	Yes	Yes
Facilitates trim-safe app size reduction	No	Yes	Yes
Supports all serialization features	Yes	No	Yes

https://devblogs.microsoft.com/dotnet/try-the-new-system-text-json-source-generator/

Source Generators in .NET 6 – JSON Serializer

- System.Text.JSON has 3 modes
  - Existing Model - reflection based as used in previous versions
  - Pre-generating optimized serialization logic
  - Pre-generating data access model

- Create custom partial class that inherits from JsonSerializerContext
  - Source generator will generate methods and properties required based on type specified in `[JsonSerializable]` attribute and the GenerationMode specified


.NET
7

Source Generators in .NET 7 –RegEx

```
public static partial class UrlValidator
{
    [GeneratedRegex(@"[(http(s)?):\/\/(www\.)?a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&//=]*)",
        RegexOptions.CultureInvariant, matchTimeoutMilliseconds: 1000)]
    private static partial Regex MatchIfValidUrl();

    public static bool IsValidUrl(string url) => MatchIfValidUrl().IsMatch(url);
}
```

- Introduction of **GeneratedRegexAttribute**
  - Requires a partial class containing a static partial method
  - Only suitable for fixed regular expression patterns known at compile time
  - As the method is static, nothing to ‘new’ up
  - Can see the generated code in Visual Studio

RegexGenerat...[generated]

This file is auto-generated by the generator 'System.Text.RegularExpressions.Generator.RegexGenerator' and cannot be edited.

BenchmarkOldRegEx | UrlValidator | MatchIfValidUrl()

```
1   :auto-generated/>
2   .lable enable
3   igma warning disable CS0162 // Unreachable code
4   igma warning disable CS0164 // Unreferenced label
5   igma warning disable CS0219 // Variable assigned but never used
6
    rences
7   ial class UrlValidator
8
9   /// <remarks>
10  /// Pattern explanation:<br/>
11  /// <code>
12  /// o Match a character in the set [#%()+.-:=?-Z_a-z~] greedily at least 2 and at most 256 times.<br/>
13  /// o Match '.'.<br/>
14  /// o Match a character in the set [a-z] greedily at least 2 and at most 6 times.<br/>
15  /// o Match if at a word boundary.<br/>
16  /// o 1st capture group.<br/>
17  ///     o Match a character in the set [#%&amp;+--:=?-Z_a-z~] atomically any number of times.<br/>
18  /// </code>
19  /// </remarks>
20  [global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Text.RegularExpressions.Generator", "7.0.6.47203")]
    2 references
21  private static partial global::System.Text.RegularExpressions.Regex MatchIfValidUrl() => global::System.Text.RegularExpressions.Generated.MatchIfValidUrl_0.Instance;
22
23
24  space System.Text.RegularExpressions.Generated
25
26  using System;
27  using System.CodeDom.Compiler;
28  using System.Collections;
29  using System.ComponentModel;
30  using System.Globalization;
31  using System.Runtime.CompilerServices;
32  using System.Text.RegularExpressions;
33  using System.Threading;
34
35  /// <summary>Custom <see cref="Regex"/>-derived type for the MatchIfValidUrl method.</summary>
36  [GeneratedCodeAttribute("System.Text.RegularExpressions.Generator", "7.0.6.47203")]
    4 references
37  file sealed class MatchIfValidUrl_0 : Regex
38  {
39      /// <summary>Cached, thread-safe singleton instance.</summary>
40      internal static readonly MatchIfValidUrl_0 Instance = new();
41
42      /// <summary>Initializes the instance.</summary>
        1 reference
43      private MatchIfValidUrl_0()
44      {
45          base.pattern = "[(http(s)?):\\/\\/(www\\.)?a-zA-Z0-9@:%._\\+~#=]{2,256}\\.[a-z]{2,6}\\b([-a-zA-Z0-9@:%_\\+.~#?&//=]*)";
46          base.roptions = RegexOptions.CultureInvariant;
47          base.internalMatchTimeout = TimeSpan.FromMilliseconds(1000);
48          base.factory = new RunnerFactory();
49          base.capsize = 2;
50      }
51
52      /// <summary>Provides a factory for creating <see cref="RegexRunner"/> instances to be used by methods on <see cref="Regex"/>.</summary>
        1 reference
53      private sealed class RunnerFactory : RegexRunnerFactory
```

110 % | No issues found | Ln: 21 Ch: 5 SPC CRLF

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'RegExSourceGenTest' (3 of 3 projects)
  - BenchmarkOldRegEx
    - Dependencies
      - Analyzers
        - Microsoft.CodeAnalysis.Analyzers
        - Microsoft.CodeAnalysis.CSharp.Analyzers
        - Microsoft.CodeAnalysis.CSharp.NetAnalyzers
        - Microsoft.CodeAnalysis.NetAnalyzers
        - Microsoft.Interop.JavaScript.JSImportGenerator
        - Microsoft.Interop.LibraryImportGenerator
        - Microsoft.Interop.SourceGeneration
        - System.Text.Json.SourceGeneration
        - System.Text.RegularExpressions.Generator
          - SYSLIB1045: Convert to 'GeneratedRegexAttribute'.
          - System.Text.RegularExpressions.Generator.RegexGenerator
            - RegexGenerator.g.cs
      - Frameworks
      - Packages
    - Imports
    - amd64
    - arm64
    - bin
    - obj
    - x86

Solution Explorer | Git Changes

Properties

Source Generators in .NET 7 –RegEx

```
public static partial class UrlValidator
{
    [GeneratedRegex(@"[(http(s)?):\/\/(www\.)?a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&//=]*)",
        RegexOptions.CultureInvariant, matchTimeoutMilliseconds: 1000)]
    private static partial Regex MatchIfValidUrl();

    public static bool IsValidUrl(string url) => MatchIfValidUrl().IsMatch(url);
}
```

- Introduction of **GeneratedRegexAttribute**
  - Requires a partial class containing a static partial method
  - Only suitable for fixed regular expression patterns known at compile time
  - As the method is static, nothing to ‘new’ up
  - Can see the generated code in Visual Studio

Method	Mean	Error	StdDev	Ratio	Gen0	Allocated	Alloc Ratio
'Using standard RegEx.IsMatch with new RegEx instance'	4,289.89 ns	25.371 ns	21.186 ns	1.00	0.5417	4592 B	1.00
'Using standard RegEx.IsMatch with already created RegEx instance'	197.50 ns	0.632 ns	0.560 ns	0.05	-	-	0.00
'Using generated MatchIfValidUrl'	77.70 ns	0.394 ns	0.349 ns	0.02	-	-	0.00

Source Generators in .NET 7 – LibraryImport

- Introduction of LibraryImport attribute
  - Using the DLLImport attribute to P/Invoke native code creates an IL stub that is JIT-ed at runtime to transition from managed to unmanaged code
  - The LibraryImport creates the transition code compile time

```
internal static class NativeCode
{
    [DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    public static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);
}

internal static partial class NewNativeCode
{
    [LibraryImport("user32.dll", StringMarshalling = StringMarshalling.Utf16, SetLastError = true)]
    public static partial int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);
}
```

Source Generators in .NET 7 – LibraryImport

```
internal static unsafe partial class NewNativeCode
{
    [System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.Interop.LibraryImportGenerator", "7.0.6.47203")]
    [System.Runtime.CompilerServices.SkipLocalsInitAttribute]
    2 references
    public static partial int MessageBox(nint hWnd, string lpText, string lpCaption, uint uType)
    {
        int __lastError;
        int __retVal;
        // Pin - Pin data in preparation for calling the P/Invoke.
        fixed (void* __lpText_native = &global::System.Runtime.InteropServices.Marshalling.Utf16StringMarshaller.GetPinnableReference(lpText))
        fixed (void* __lpCaption_native = &global::System.Runtime.InteropServices.Marshalling.Utf16StringMarshaller.GetPinnableReference(lpCaption))
        {
            System.Runtime.InteropServices.Marshal.SetLastSystemError(0);
            __retVal = __PInvoke(hWnd, (ushort*)__lpText_native, (ushort*)__lpCaption_native, uType);
            __lastError = System.Runtime.InteropServices.Marshal.GetLastSystemError();
        }

        System.Runtime.InteropServices.Marshal.SetLastPInvokeError(__lastError);
        return __retVal;
        // Local P/Invoke
        [System.Runtime.InteropServices.DllImportAttribute("user32.dll", EntryPoint = "MessageBox", ExactSpelling = true)]
        static extern unsafe int __PInvoke(nint hWnd, ushort* lpText, ushort* lpCaption, uint uType);
    }
}
```

Source Generators - C# 11 Improvements

- Introduction of File Scoped Types
  - Limit access to a type to the file it is declared in using the `file` modifier
  - Useful for avoiding naming collisions in between source generated code and consuming project
  - As limited to file, the type cannot be a parameter or return value for a method that has a greater visiblity

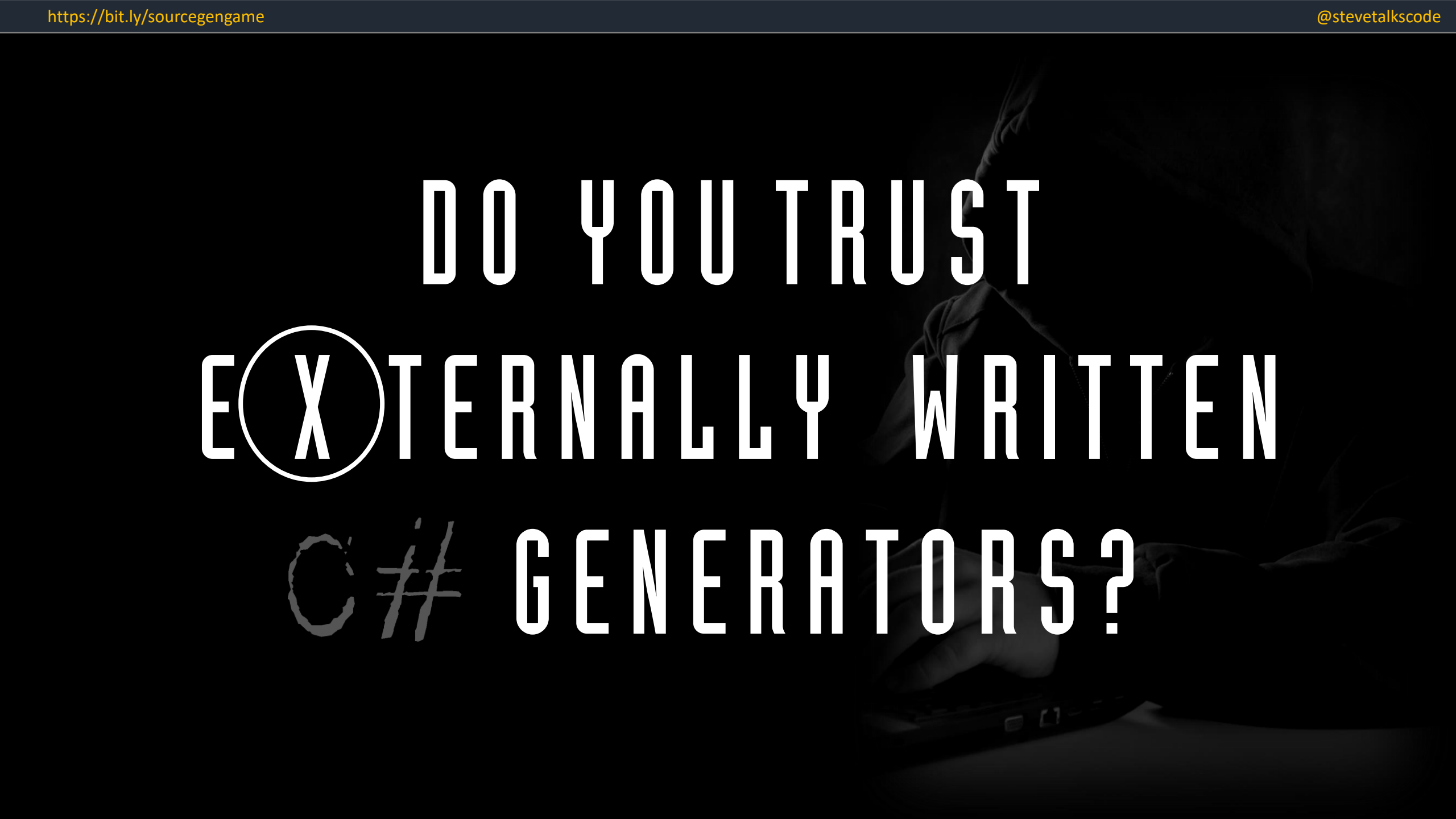
Source Generators - C# 11 Improvements

- Introduction of Raw String Literals
  - Building a string of code in a source generator is painful as need to use lots of escape characters!
  - Can now use multiple (at least 3) double quotes to indicate that the quotes, backslash, curly brackets should be used as literals
  - Iterpolation symbols ($ and {}) can also be handled by prefixing with multiple $ symbols

Source Generators are great!

But ...

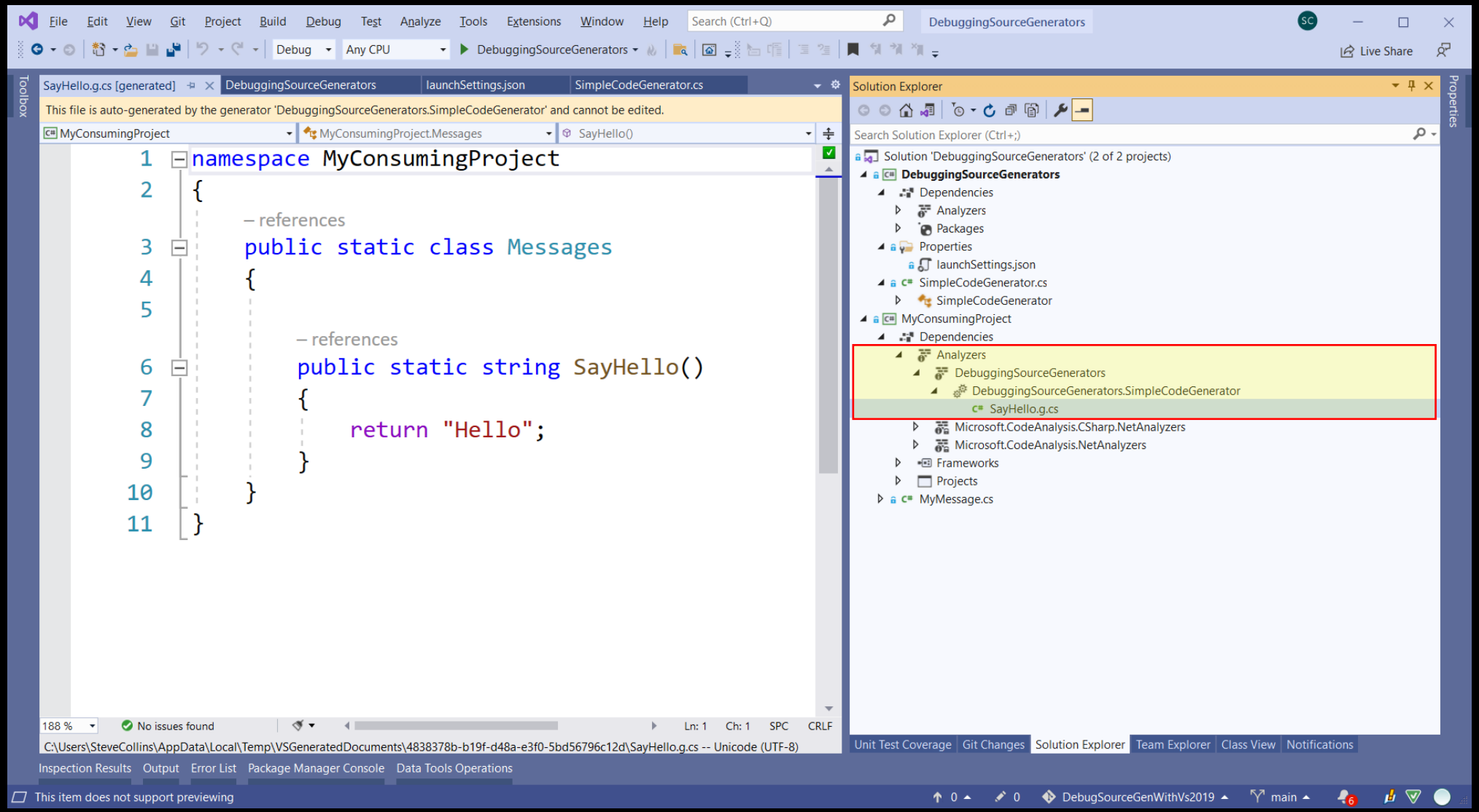

WITH GREAT POWER COMES
GREAT RESPONSIBILITY


DO YOU TRUST
EXTERNALLY WRITTEN
C# GENERATORS?

Generators Can Be An Attack Vector

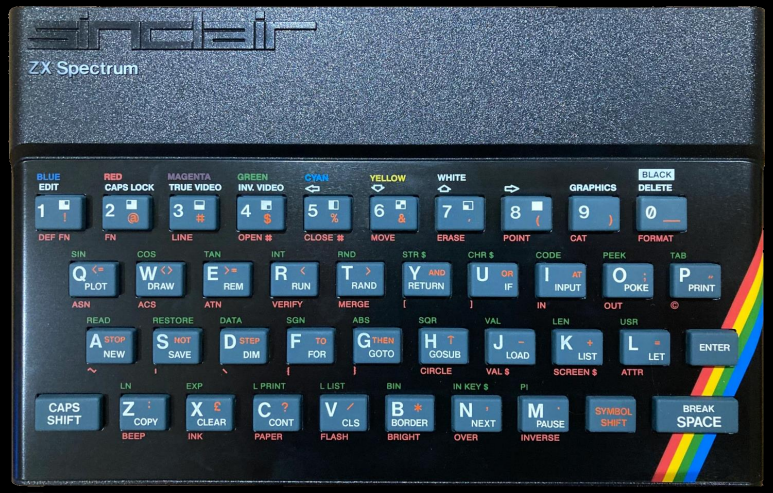
- Do you know what the generator is doing with the source code it is reading?
  - Is it sending information out over the internet?
  - Is it logging data about your code anywhere unexpected?
- Do you have the source code for the generator?
  - How much do you trust that NuGet package being build from correct source?
- Can you trust the code that is being generated to be secure?
  - Is it building malicious code to execute inside your code?
  - Could be doing this just on Release build and not Debug build …
  - … only builds malicious code on build server, not dev machine

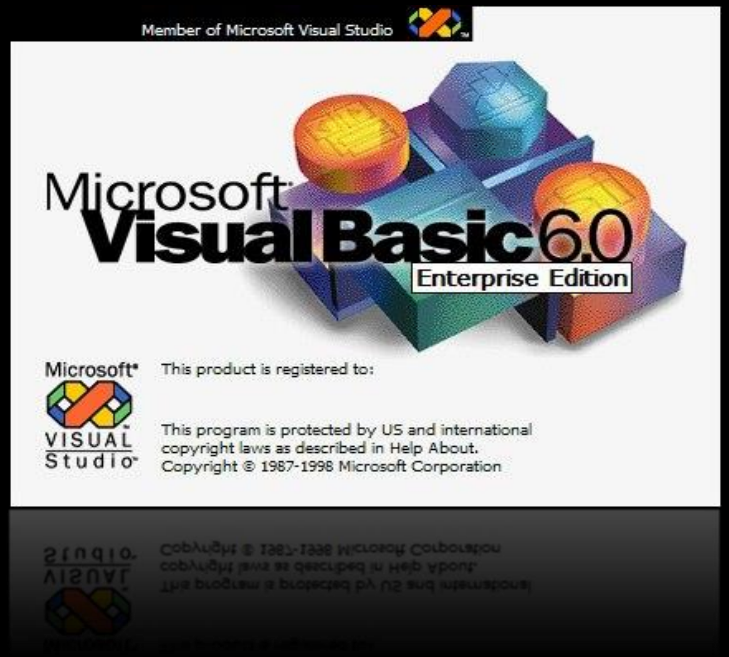
Generators Can Be An Attack Vector



Generators Can Be An Attack Vector

- If open source
  - Check the source code
  - Are you sure the NuGet package comes from the known repository?
- If commercial NuGet package
  - Do you trust the vendor?
  - Is it digitally signed and verifiable?
- If developed in-house
  - Has it been code reviewed?
  - Can a code change be merged into CI monitored branch without a pull request?

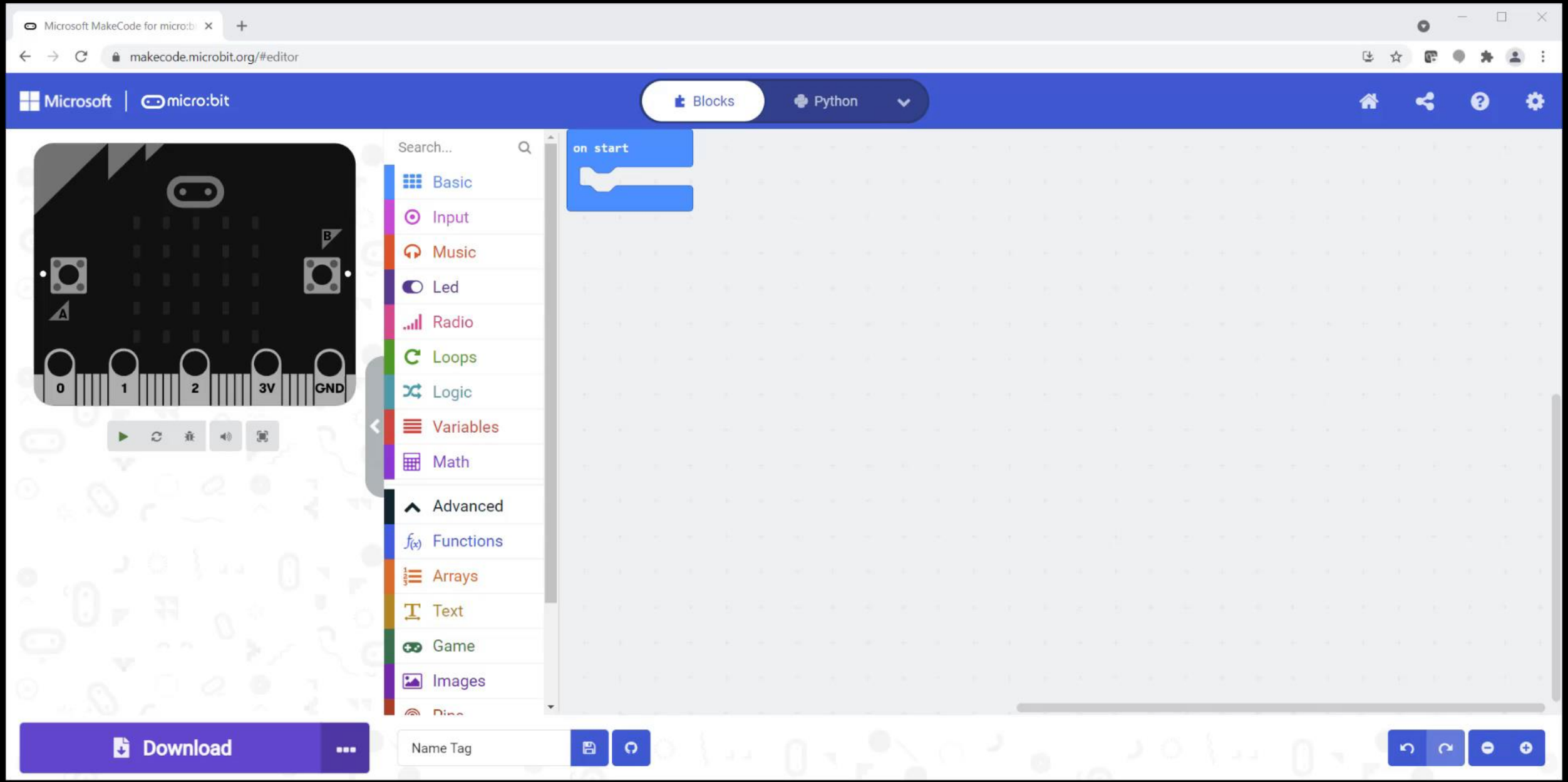


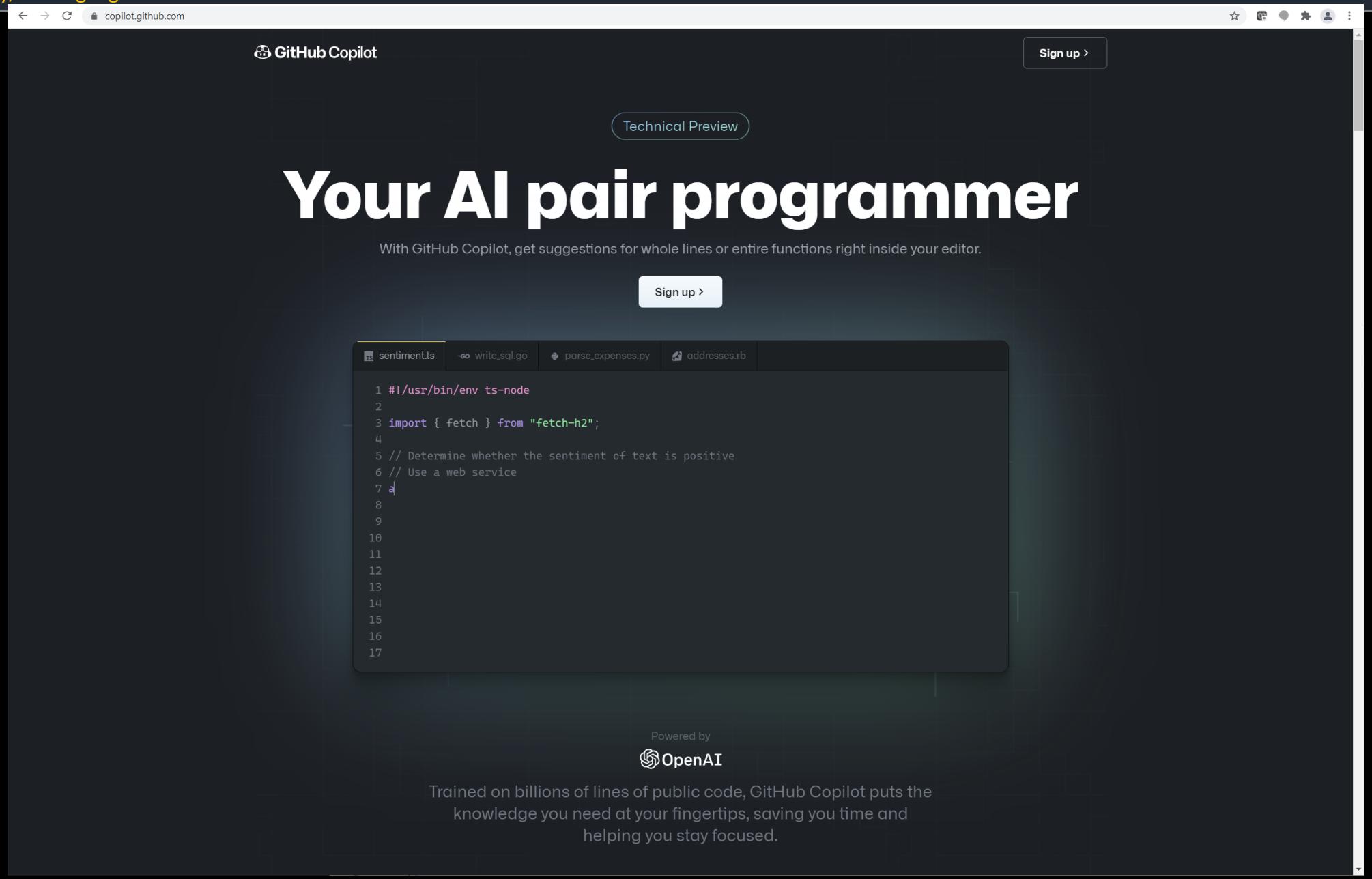


.NET 7

1988 2022

Photo by Joshua Woroniecki on Unsplash

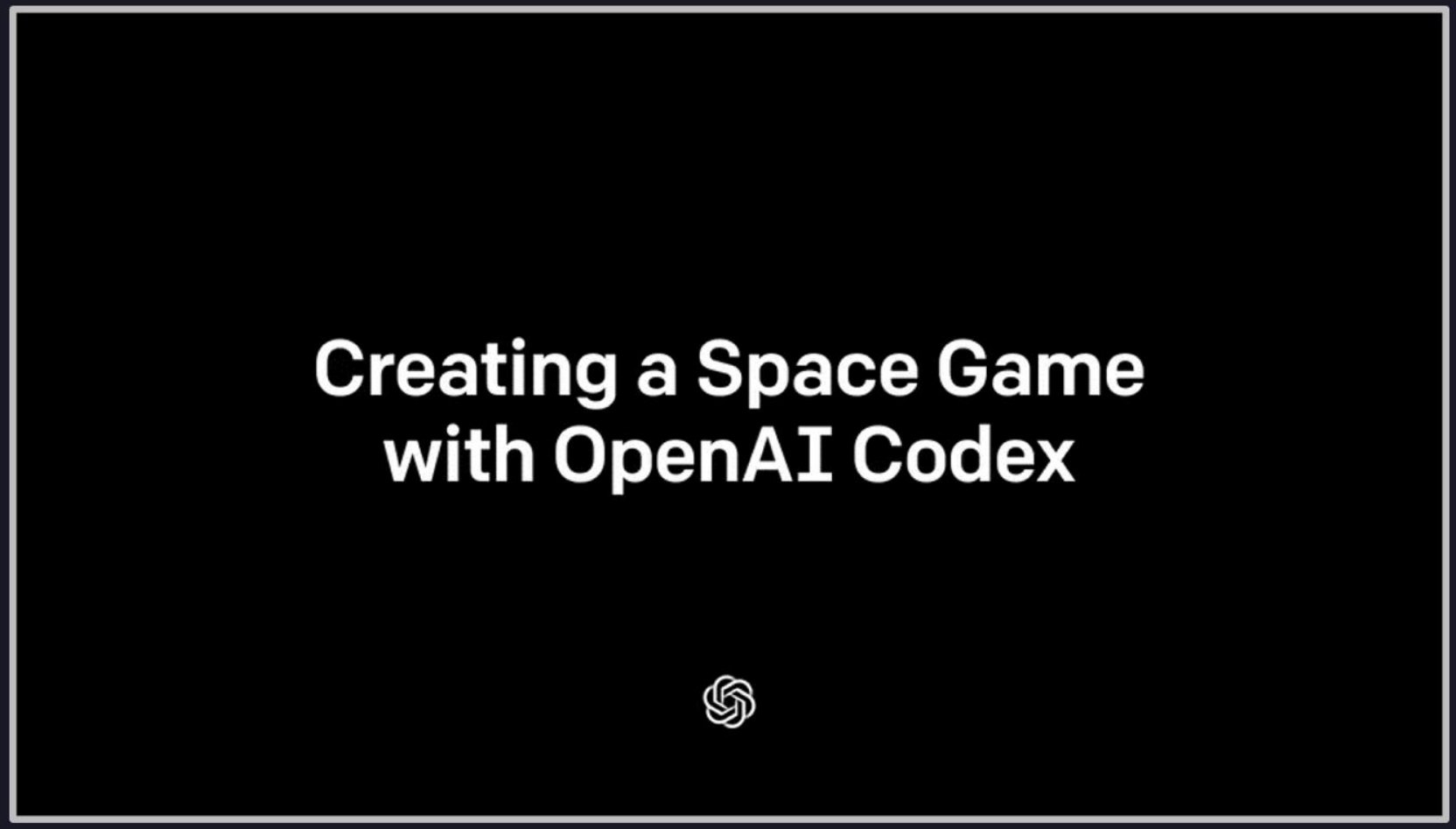

Microsoft MakeCode for micro:b
makecode.microbit.org/#editor
Microsoft
micro:bit
Blocks
Python
on start
Search...
Basic
Input
Music
Led
Radio
Loops
Logic
Variables
Math
Advanced
Functions
Arrays
Text
Game
Images
0
1
2
3V
GND
A
B
Download
Name Tag


copilot.github.com
GitHub Copilot
Sign up >
Technical Preview
Your AI pair programmer
With GitHub Copilot, get suggestions for whole lines or entire functions right inside your editor.
Sign up >
sentiment.ts
write_sql.go
parse_expenses.py
addresses.rb
#!/usr/bin/env ts-node
import { fetch } from "fetch-h2";
// Determine whether the sentiment of text is positive
// Use a web service
Powered by
OpenAI
Trained on billions of lines of public code, GitHub Copilot puts the knowledge you need at your fingertips, saving you time and helping you stay focused.

OpenAI Codex

August 10, 2021
3 minute read

We've created an improved version of OpenAI Codex, our AI system that translates natural language to code, and we are releasing it through our API in private beta starting today. Codex is the model that powers GitHub Copilot, which we built and launched in partnership with GitHub a month ago. Proficient in more than a dozen programming languages, Codex can now interpret simple commands in natural language and execute them on the user's behalf—making it possible to build a natural language interface to existing applications. We are now inviting businesses and developers to build on top of OpenAI Codex through our API.



https://openai.com/blog/openai-codex

NVIDIA

HOME AI DATA CENTER DRIVING GAMING PRO GRAPHICS AUTONOMOUS MACHINES HEALTHCARE INCEPTION AI PODCAST

40 Years on, PAC-MAN Recreated with AI by NVIDIA Researchers

GameGAN, a generative adversarial network trained on 50,000 PAC-MAN episodes, produces a fully functional version of the dot-munching classic without an underlying game engine.

May 22, 2020 by ISHA SALIAN

https://blogs.nvidia.com/blog/2020/05/22/gamegan-research-pacman-anniversary/

Summary: What Goes In Drives What Comes Out

(In) Metadata

- The raw material from which you generate source code
  - API structures -WSDL / OpenAPI (Swagger) / gRPC Proto files
  - Database structures – ISO standard definition for INFORMATION_SCHEMA
  - Data structures - XML / JSON / CSV – understanding the schema of your data
  - Code - Syntax and Semantic Trees
  - Anything you can analyse and get meaning from – even raw memory

(Out) Syntax

- Understand how target language statements are constructed
- No intellisense to help you!
- Unit Tests – Real-time debugging is hard!

Thank You!

Thank You

- Twitter – @stevetalkcode
- Blog – https://stevetalkscode.co.uk
- Milton Keynes .NET User Group – @dotMiltonKeynes
- Other Talks
  - .NET Configuration Is Easy - Right?
  - .NET Dependency Injection – The Booster Jab
- Podcasts
  - Dot Net Core Show #49 & #75 - https://dotnetcore.show/
  - Unhandled Exception Podcast #24 - https://unhandledexceptionpodcast.com