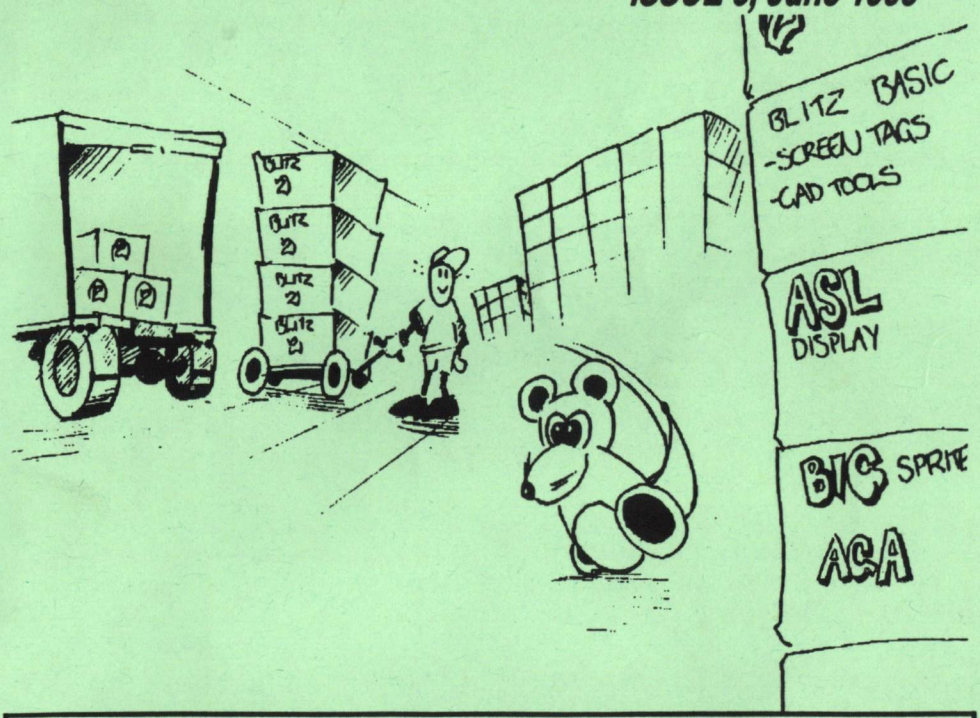
INFORMATION AND SUPPORT FOR BLITZ BASIC 2 USERS WORLD WIDE

BLITZ USER

ISSUE 5, June 1993



WOW, 40+ new commands for beta testing and some serious demos including documented source code! Who cares if it's 6 weeks late?

BACK IN THE OFFICE...

More gossip from the St Kevins Arcade office of Acid Software & Vision Software...

Up in St Kevins Arcade things have been as crazy as ever. Vision Software are looking to blow everyone out with Seek and Destroy a game which I can say makes another helicopter game released this year look like a heli-tour of Wimbledon common.

SkidMarks is shaping up well with Andrew spending even more time at the office than up at Auckland University where he's supposedly a full time student. Our raytracing efforts are looking really nice, the packaging is pretty much sussed and it's now time to finish the 12 tracks promised for final release. Wo, wouldn't it be nice for the first Blitz2 commercial game to hit the top ten?

The CD32 has arrived and has been fully tested with the entire CD collection (music that is). The controller is awesome with no less than 7 buttons plus joypad and the buster chip is going to get one hell of a workout over the next few weeks.

Favourite CD of the month goes to Shihad's CHURN, a local kiwi group which we all hope are going to kick some butt overseas.

Speaking of kicking butt, Blitz User's should look out for Amiga Format issues 51 and 52. With this sort of exposure we're hoping to pronounce a certain A**S PRO and accompanying compiler out of contention for Amiga language sales this XMas. However we need PD material to keep the reputation alive so come on everyone, anything will do as long as it's different!

No. 5

CONTENTS

BLITZ USER

Blitz User is a publication of Acid Software.

Editor

Simon Armstrong

Art Director

Rod Smith

Forward all contributions, advertising and correspondence to:

ACID SOFTWARE
10 StKevins
Arcade
Karangahape Rd
Auckland
New Zealand

fax:64-9-358-1658

EDITORIAL

Yo BlitzUsers! Well BUM5 is late, very late! Sorry, even now most of the new commands haven't got full error checking and the docs are a little rough. We'll just call this a beta testing issue so don't go getting upset cos things don't work properly. Just calmly fill out a bug report and send/fax/post it my way.

OK so what do you get to test? The new display library is coming together nicely, it's a bit different than how slices work but an improvement I think overall. If you have not already got it together with slices I'd wait for better documentation before having a crack with the new display commands.

For applications development there is new support for both GadTools and ASL requesters. It seems pretty stable at present but once again these new commands are still in the beta-test catagory.

It's been enjoyable work adding all the extra stuff and hopefully you'll be able to use it to create some decent software. Although we don't demand that Blitz2 be mentioned on any releases developed with it we would appreciate the publicity. It is in everyone's interest that Blitz2 sells well and Acid Software keeps paying it's bills so go on give us a credit in your next PD/commercial release.

Speaking of paying the bills, our best wishes go out to Commodore leading up to Christmas. If they pull through the next few months by shipping megatons of AGA machines the new year could offer us all some real opportunities for creative projects.

The best news of course is CD32, we'll be releasing ACID 1 before XMas which will include an awesome AGA version of SkidMarks, Defender (the controllers come with 6 buttons!!!) Insectoids2 and more. This confirms of course the existance of a complete set of CD32 developer commands for Blitz2. These with a special cable, HD CD emulation drivers and our special bootCD will mean Blitz2 might well indeed be responsible for the biggest revolution in CD development tools ever. Hell, we might even charge a royalty of US$3 per disk too (just kidding).

Other future enhancements for Blitz2 include Oopsi support (slowly working my way through my RKM :), a linker, another hack at a 3D environment, superbitmap windows, a new Intuitools program, hmmm better stop...

A big hi goes out to the AcidSoftware master distributors, we're slowly developing a cohesive world wide marketing plan for BB2, at present we have on board:

Germany: Tom & Falk
phone 02217710922 fax=0940

UK: Benoit Varasse
phone/fax 071 482 4066

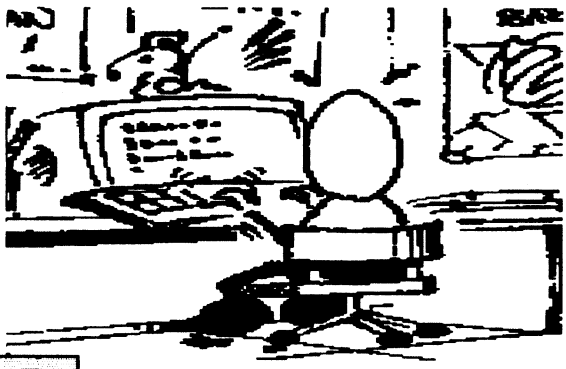
OZ: Roy Hurley
phone/fax 042 281 489

USA: Dave Maziarka
phone 608 257 9057

Once we start shipping decent quantities per month in each region we will be helping to get BUGs started (Blitz User Groups). The BUGs will take over most of the support including BBS support and distribution of BUMs (Blitz User Magazines). Keeping with the spirit of things I would also hope that BUGs can operate as PD Libraries and also gather contributions for BUMs. Each BUG will get 2 pages in future BUMs and we'll get some competitions between BUGs going to get everyone fired up.

Anyway, I'll get BUM6 out before XMas so everyone knows what's happening for 1994, promise!

SIMON



Programming a COMMODITY

by Thomas Boerkel

What is a Commodity?

Before AmigaOS 2.0, you had to write your own custom input handler and "link" it to the input.device if you wanted to react on several input-events. Sometimes this method caused problems when many programs created such handlers. There also was a lack of transparency and control for the user. Starting with AmigaOS 2.0, Commodore created a standard for programs which act on input-events.

The new commodities.library controls all those programs (called "Commodities") and helps the programmer with several useful functions. The control-program "Exchange", which can be found in your Workbench 2.x/3.x:Tools/Commodities drawer, shows all running Commodities and information about them. The user can enable/disable a Commodity, force it to open/close its window and terminate it.

Almost all programs which act on input-events can be implemented as a Commodity. Input-events are keystrokes, mousemoves, mouseclicks and some other ones that are less important. Some examples for programs which can and should be implemented as a Commodity are: Keytranslators, program-launchers (for example Shell-popup), mouseblankers, screenblankers, screen/window-tools...

What a Commodity has to do:

Commodities should have the following tooltypes:

CX_PRIORITY=x
CX_POPUP=YES|NO
CX_POPKEY=keystroke

The last two only apply to Commodities that can open a window.

Commodities have to react on messages from Exchange. If the user tries to start the Commodity again (while it is already running), the new started should shutdown itself immediatly. The already running one will receive a "UNIQUE"-message and should then do something, normally it will popup its window. Commodities should be as small as possible because they stay always in memory. They should only act on input-events of their interest to keep CPU-usage small. One exceptional type of Commodity is a screenblanker. This one is interested in all kinds of input-events.

How to program a Commodity

To program a Commodity, you have to install some objects. The main object is called "Broker". This object is linked to the Commodities-handler which is linked in the input-stream. The Broker must have a message-port, where it gets messages from Exchange and the commodities.library. Other objects are:

FilterObjects:filter inputeventsof your choice
SenderObjects:send messages to ports
TranslateObjects:translate/modify events.
CustomObjects:all other kinds of objects.

Messages from Exchange have to be sorted out and the required action has to be taken.

1. Test Commodity with popup-window

```
                -> Translator
               /
Broker -> Filter ->
               \
                -> Sender
```

The Broker sends all input-events to the Filter. The Filter sorts out one special keystroke and gives it to the Translator, which translates it to nothing (input-event is eliminated). The Filter also activates the Sender, which sends a message to a specific message-port. This can be and is often the same as the Brokers port. The filtering, translating and sending is done by the commodities.library without the need for the Commodity to do something. Actually the Commodity sleeps until the Sender sends it the message. Then it performs its action, in this example pops up its window.

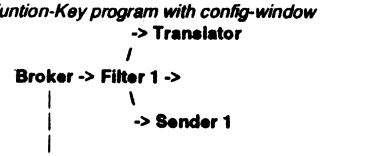
2 Mouseblanker with config-window.

```
                -> Translator
               /
Broker -> Filter 1 ->
          |          \
          |           -> Sender 1
          |
          -> Filter 2 -> Sender 2
```

Filter 1/Sender 1/Translator do the same as in Example 1. They inform the Commodity when it has to open its window and kill the input-event (keystroke)

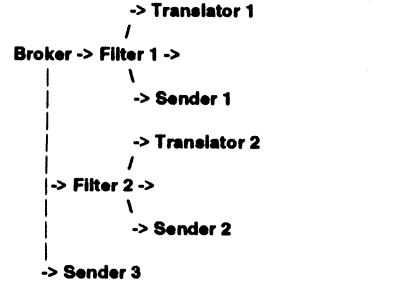
from the inputstream. Filter 2 activates Sender 2 on mousemoves. Sender 2 sends a message to the Commodity, which then triggers its timer for mouseblanking.

3. Funtion-Key program with config-window



Filter 1/Sender 1/Translator do the same as in Example 1. They inform the Commodity when it has to open its window and kill the input-event (keystroke) from the inputstream. For each F-key is a Filter and a Translator installed. The Filter activates the Translator if the specific key is pressed. The Translator translates the keystroke to a series of input-events (keystrokes). So pressing "F1" could bring out the string "Dir" or something like that. Please note that the Translator needs a number of chained input-events in reversal(!) order. There is a comfortable function in Amiga.lib "InvertString()" which could do this, if it could be used in Blitz2. But unfortunately this isnt possible in the current version of BlitzBasic so you have to build your own chain of input-events.

4. Screenblanker + config-window + blankkey



Filter 1/Sender 1/Translator 1 do the same as in Example 1. They inform the Commodity when it has to open its window and kill the input-event (keystroke) from the inputstream. Filter 2/Sender 2/Translator 2 are necessary for the blankkey. If the blanker gets a message from Sender 2 it has to blank at once. Sender 3 is directly connected to the Broker. This means that it sends a message to the port for every input-event, because a screenblanker is interested in almost every kind of input-event. The screenblanker has to select which action to do on which input-event. Normally only screenblankers have to look after every input-event. All other Commodities usually are only interested in special input-events. So the blanker can receive messages at the Brokers message-port from all 3 senders and from Exchange.

Sources

On the BUM5 coverdisk you should find the documented sources to BlitzBlank (full-featured screenblanker) and TestComm (simply a test with popup-window and key-translation for the F1-key).

Parts of special interest in BlitzBlanks source:

WB-startup-handling: BlitzBlank does not use "WBStartup", but does its own WB-startup-handling to be able to get its tooltypes. Look at the start of BlitzBlank to see how the WBMessage is got, and at the end of the main program to see what a program with its own wb-start-handling has to do at the end (hey, Simon, what about getting tooltypes with Blitz-commands?).

Getting the tooltypes: At the start of BlitzBlank, it gets its tooltypes with FindToolTypes_().

Screenmoderequester: The included procedure "Screenreq" gets the possible screenmodes from your machine and displays a screenmoderequester with GadTools-gadgets. It uses the global long variables "modeid", "width", "height", "depth".

GadTools-GUI: BlitzBlanks window is full of different kinds of GadTools-gadgets. Look at the subroutines "winon" and "gadgets" to see how that works. Preparing an image for a GadTools-gadget: The drawer-gadget's image is prepared in the subroutine "doimagedata" and freed in "freeimagedata".

Writing tooltypes to Icon: BlitzBlank is able to set its own tooltypes with the current settings if the user clicks on the SAVE-gadget. Look at the subroutine "writett" which does this.

Using the ASL-filerequester: BlitzBlank displays the ASL-filerequester when the user presses <HELP> in the path-gadget or clicks on the drawer-gadget. The requester is displayed with the subroutine "aslfilereq".

Using EasyRequesters: In certain cases, BlitzBlank displays an EasyRequest. This is done in the subroutine "requester".

Reading and writing config-data: The subroutines "readconfig" and "writeconfig" handle the "BB.modules.config" file. These are just the same routines as in the modules, but in BlitzBlank theyre done with OS-routines.

Setting up the Commodity: The subroutine "docommall" sets/resets the Commodity-objects.

Killing the Commodity objects: The subroutine "killcomm" deletes all Commodity-objects.

Finding the available modules: The subroutine "findmodules" searches with MatchNext() from DOSASL in the module-directory.

Handling an exec.library-list: An exec-list is needed for the ListView-gadget of modules. The subroutines "initlist", "addallnodes" and "freeallnodes" handle this list.

ROGER'S BEGINNERS COLUMN

Hi again and welcome to the second of the beginners tutorials, this issue we will be covering NUMBER SYSTEMS

Firstly, I'm gonna take some of the rap for the lateness of BUM #5 I've been a trifle lazy getting my stuff in to Simon, so sorry folks, still better late than never.....

This topic NUMBER SYSTEMS is actually quite advanced and you'll find many a BASIC programmer who has no concept of them whatsoever, however as with last month if you want to REALLY understand programming, you'll want to know these so ONWARD.....

(1) WHAT A COMPUTER IS, HOW COMPUTERS COUNT, WHAT MEMORY IS AND WHAT NUMBER SYSTEMS WE CONVERSE WITH A COMPUTER IN.

Phew.... Few techie words in that title eh? I have never seen a programming article that covers these topics first which is a pity as unless you understand how computers store information and how to retrieve and manipulate it then you wont ever FULLY understand programming.

What a computer is:

A computer is a collection of millions of interconnected switches, these combinations of switches can do all sorts of flash things but the fundemental of ALL computers, from calculators to main-frames is that they only understand 2 things. ON and OFF

So basically this mega-expensive machine you shelled out on is no more advanced than the light switch across the room, were all the tricky things you can do with a computer comes in with the fact that a computer can switch things FAST and in different combinations, However from a programmers point of view we need a convenient way to describe weather something is ON or OFF and the simplest way is to use the number 1 for ON and 0 for off

1 = ON
0 = OFF

This is called the Binary number system

The number system we use in everyday life is the DECIMAL number system (base 10), this system is used as we have 10 fingers and its those we used to count with, however to understand the basics of programming we need to learn BINARY (base 2) DECIMAL (base 10) we already know and HEXADECIMAL (base 16)

Whats a binary number?

Well to start you thinking heres a table with Binary numbers on the left and their decimal equivilents on the right from 0 to 10:

DEC	BIN
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010

Pretty Bizzarre huh? Well at least it looks that way until I tell you that all you have to do to be able to do Decimal/binary conversion is to add numbers and double numbers! ie:

32 | 16 | 8 | 4 | 2 | 1	Decimal numbers
1 | 0 | 1 | 1 | 0 | 1	Binary Number

Ok so lets examine that table:

First thing to note is that the '%' character in front of the binary number is just how programmers specify that the digit is in binary

ie: 10 is the number 10 in decimal
%10 is the number 2 in decimal

Ok now lets go back to the table, remember I told you all you had to do was be able to add and double well heres why, the top level of the table starts at the right at 1 and doubles (or goes to the power of 2) at each step to the left ie:

1 = 1
1*2 = 2
2*2 = 4
4*2 = 8
8*2 = 16
16*2 = 32 etc etc etc

Right, so that covers the doubling, now the next level down is the binary number. As you can see the little holes below the decimal numbers either have a 1 (ON) or 0 (OFF) in them. If the holes are ON then that means the decimal figure above is part of the resulting decimal number if the hole is OFF then it isn't so in the above example

%101101 = in decimal
32+8+4+1 = 45 decimal

Ok so draw yourself a copy of table 2 and using the knowledge you now have calulate the binary numbers for:
a) 32 b) 13 c) 24 d) 5 e) 64

If you have any problems just read through the example again until it 'clicks'

Ok, if you have done your sums (just like your back at school) then these are the answers
a)%100000 b)%1101 c)%11000
d)%101 e) %111111

Ok, now if we extended that table by doubleing our decimals you can see that by each position we add to the left doubles the maximum possible value we can represent

ADVANCED: This question is designed to get you guys really thinking for yourselves, I want you to try and work out the number 250 in binary, using common sense and the information above you hopefully will be able to do it.

ANSWER: You had to create more columns, you had to create eight columns so the answer looked like this:

250=128+64+32+16+8+2

128	64	32	16	8	4	2	1
1	1	1	1	1	0	1	0

so 250=%11111010

Right, so you can see so you can work out any number in binary, after a while you will be able to work out quite a few binary numbers in your head so keep practising, until you can do it mentally, check it using tables.

Ok now work out 32987928437098274 in Binary -- Just kidding

HEXADECIMAL

Now the only other number system you need to know is Hexadecimal. As discussed before binary is Base 2 Decimal is Base 10 and Hexadecimal is Base 16 (ie Decimal + Hexa) Most of you will know that Hex means 6 so 10+6= 16.

It's at this point that we come to a rather interesting problem, thus far we have only had to represent number systems with a base lower than 10 now that we have a base above 10 we need to invent some new number symbols and here they are

A = 10
B = 11
C = 12
D = 13
E = 14
F = 15

So there it is Hex values range from 0 to F and just as we preceeded a binary number with a '%' sign, we preceed a hex number with a '$' sign. Here are some Hex numbers: $1FAB, $FF, $FACE, $234DE

So how do we translate from decimal to hex and vica versa well the way I do it involves converting the number to binary inbetween.

PROB: To translate the number 182 in decimal to its hexidecimal value

ANSWER: Firstly convert 182 to Binary (%10110110), then we convert from Binary to Hex. The important part to understand here is that hex (base 16) can represent the numbers 0 to 15 before you have to do a carry over to the next column.

The next bit to realise is that we can represent the numbers 0 to 15 in binary in 4 bits ie:

8	4	2	1
1	1	1	1

%1111 = 15 = $f

So by dividing our binary number into a series of 4 bit segments we can work out our hex value ie: 182 in binary is %10110110, separate into 4 bit segments: 1011 & 0110 and then convert these 4 bit segments back to Hex ($0 - $F) which gives $B6.

This may seem rather long winded (IT IS!!) but you'll soon be able to do numbers like that in your head, and as you'll find out at the end of this lession there is a REALLY EASY way to do this stuff :) Ok now lets convert from hexidecimal to decimal Its basically the same process in reverse.

PROB: Convert the number $FA in hexidecimal to decimal.

ANSWER: First convert the Hex value to binary So $FA splits into 2 lots of 4 bits %1111 & %1010.

Now convert this 8 bit number to decimal ie:

128	64	32	16	8	4	2	1
1	1	1	1	1	0	1	0

So $Fa=128+64+32+16+8+2=250.

If you do bigger numbers and they don't split into even lots of 4 bit segments its no problem, just split them into 4 bit segments (allways starting at the right!!!) and work out your values then just work out your last value with whatever bits are left over ie:

%1101011101 => %11 0101 1101 =>$35D

So that's number systems, were now going to go into why we use these number systems, and I'm going to blither on about how this relates to last months article, but before I do remember I promised you a really easy way of working all this out?

Simply use the Calculator option in the Blitz2 compiler menu this will allow to enter a number in any base (jsut prefix it with % or $ if its binary or hex) click on the number system you want the answer to and Voila there is your answer, now before you lynch me for boring you to death with stuff you didn't need to know, YOU DO NEED TO KNOW THIS, next month when we stop all this boring theory and get into some actual BLITZING you will know why until then..... <puts on best Arnold Schwartznegger voice> 'TRUST ME'.....

Last month I did a paragraph about signed and unsigned numbers and ended up just telling you to believe me. I'm now going to show you why variables only hold up to certain numbers and why. THIS is one of the reasons why understanding number systems is necessary to understand programming. First a quick recap, if you remember there are 3 basic types of sizes a compter understands (blitz adds a few of its own but they are still based on these three types, they are.)

.b=BYTE=8bits
.w=WORD=16bits
.l=LONG=32 bits

Now last month I said a unsigned BYTE can hold from 0 to 255 THIS is why

0=%00000000->Smallest value 8 bits can hold
255=%11111111->Largest value

Now if we use SIGNED BYTES the computer needs a way of telling if the values is positive or negative, it does this by setting the most significant bit (thats techie talk for the one on the extreme left) to on if the valueis NEGATIVE

THIS MEANS, that the maximum value you can store is HALVED because that top bit is no longer available for your number and as you know every bit you add to a number doubles its capacity so taking one away halves it so as an example here are the answers to a binary value for a UNSIGNED and SIGNED byte
%10000001 as a unsigned byte equals 129
%10000001 as a signed byte equals -1

So now you see why in a byte you can represent either UNSIGNED 0 - 255 or SIGNED -128 to +128 The same is true for word and long word values I wont demonstrate them, but it is the case. Well that's it for another month, next month we do a tiny bit of theory on truth tables, and then get into basic programme flow control and into some REAL PROGRAMME WRITING. Until then, keep on hammering away yourselves, and if you write something then SEND IT OVER THE WATER to us here in NZ.

ROGER

$00	0	%00000000	$40	64	@	%01000000	$80	128		%10000000	$C0	192	À	%11000000
$01	1	%00000001	$41	65	A	%01000001	$81	129		%10000001	$C1	193	Á	%11000001
$02	2	%00000010	$42	66	B	%01000010	$82	130		%10000010	$C2	194	Â	%11000010
$03	3	%00000011	$43	67	C	%01000011	$83	131		%10000011	$C3	195	Ã	%11000011
$04	4	%00000100	$44	68	D	%01000100	$84	132		%10000100	$C4	196	Ä	%11000100
$05	5	%00000101	$45	69	E	%01000101	$85	133		%10000101	$C5	197	Å	%11000101
$06	6	%00000110	$46	70	F	%01000110	$86	134		%10000110	$C6	198	Æ	%11000110
$07	7	%00000111	$47	71	G	%01000111	$87	135		%10000111	$C7	199	Ç	%11000111
$08	8	%00001000	$48	72	H	%01001000	$88	136		%10001000	$C8	200	È	%11001000
$09	9	%00001001	$49	73	I	%01001001	$89	137		%10001001	$C9	201	É	%11001001
$0A	10	%00001010	$4A	74	J	%01001010	$8A	138		%10001010	$CA	202	Ê	%11001010
$0B	11	%00001011	$4B	75	K	%01001011	$8B	139		%10001011	$CB	203	Ë	%11001011
$0C	12	%00001100	$4C	76	L	%01001100	$8C	140		%10001100	$CC	204	Ì	%11001100
$0D	13	%00001101	$4D	77	M	%01001101	$8D	141		%10001101	$CD	205	Í	%11001101
$0E	14	%00001110	$4E	78	N	%01001110	$8E	142		%10001110	$CE	206	Î	%11001110
$0F	15	%00001111	$4F	79	O	%01001111	$8F	143		%10001111	$CF	207	Ï	%11001111
$10	16	%00010000	$50	80	P	%01010000	$90	144		%10010000	$D0	208	Ð	%11010000
$11	17	%00010001	$51	81	Q	%01010001	$91	145		%10010001	$D1	209	Ñ	%11010001
$12	18	%00010010	$52	82	R	%01010010	$92	146		%10010010	$D2	210	Ò	%11010010
$13	19	%00010011	$53	83	S	%01010011	$93	147		%10010011	$D3	211	Ó	%11010011
$14	20	%00010100	$54	84	T	%01010100	$94	148		%10010100	$D4	212	Ô	%11010100
$15	21	%00010101	$55	85	U	%01010101	$95	149		%10010101	$D5	213	Õ	%11010101
$16	22	%00010110	$56	86	V	%01010110	$96	150		%10010110	$D6	214	Ö	%11010110
$17	23	%00010111	$57	87	W	%01010111	$97	151		%10010111	$D7	215	×	%11010111
$18	24	%00011000	$58	88	X	%01011000	$98	152		%10011000	$D8	216	Ø	%11011000
$19	25	%00011001	$59	89	Y	%01011001	$99	153		%10011001	$D9	217	Ù	%11011001
$1A	26	%00011010	$5A	90	Z	%01011010	$9A	154		%10011010	$DA	218	Ú	%11011010
$1B	27	%00011011	$5B	91	[	%01011011	$9B	155		%10011011	$DB	219	Û	%11011011
$1C	28	%00011100	$5C	92	\	%01011100	$9C	156		%10011100	$DC	220	Ü	%11011100
$1D	29	%00011101	$5D	93	]	%01011101	$9D	157		%10011101	$DD	221		%11011101
$1E	30	%00011110	$5E	94	^	%01011110	$9E	158		%10011110	$DE	222	Þ	%11011110
$1F	31	%00011111	$5F	95		%01011111	$9F	159		%10011111	$DF	223	ß	%11011111
$20	32	%00100000	$60	96	`	%01100000	$A0	160		%10100000	$E0	224	à	%11100000
$21	33	! %00100001	$61	97	a	%01100001	$A1	161	¡	%10100001	$E1	225	á	%11100001
$22	34	" %00100010	$62	98	b	%01100010	$A2	162	¢	%10100010	$E2	226	â	%11100010
$23	35	# %00100011	$63	99	c	%01100011	$A3	163	£	%10100011	$E3	227	ã	%11100011
$24	36	$ %00100100	$64	100	d	%01100100	$A4	164	¤	%10100100	$E4	228	ä	%11100100
$25	37	% %00100101	$65	101	e	%01100101	$A5	165	¥	%10100101	$E5	229	å	%11100101
$26	38	& %00100110	$66	102	f	%01100110	$A6	166	¦	%10100110	$E6	230	æ	%11100110
$27	39	' %00100111	$67	103	g	%01100111	$A7	167	§	%10100111	$E7	231	ç	%11100111
$28	40	(%00101000	$68	104	h	%01101000	$A8	168		%10101000	$E8	232	è	%11101000
$29	41	) %00101001	$69	105	i	%01101001	$A9	169	©	%10101001	$E9	233	é	%11101001
$2A	42	* %00101010	$6A	106	j	%01101010	$AA	170	ª	%10101010	$EA	234	ê	%11101010
$2B	43	+ %00101011	$6B	107	k	%01101011	$AB	171	«	%10101011	$EB	235	ë	%11101011
$2C	44	, %00101100	$6C	108	l	%01101100	$AC	172		%10101100	$EC	236	ì	%11101100
$2D	45	- %00101101	$6D	109	m	%01101101	$AD	173	–	%10101101	$ED	237	í	%11101101
$2E	46	. %00101110	$6E	110	n	%01101110	$AE	174	®	%10101110	$EE	238	î	%11101110
$2F	47	/ %00101111	$6F	111	o	%01101111	$AF	175		%10101111	$EF	239	ï	%11101111
$30	48	0 %00110000	$70	112	p	%01110000	$B0	176	°	%10110000	$F0	240	ð	%11110000
$31	49	1 %00110001	$71	113	q	%01110001	$B1	177	±	%10110001	$F1	241	ñ	%11110001
$32	50	2 %00110010	$72	114	r	%01110010	$B2	178	²	%10110010	$F2	242	ò	%11110010
$33	51	3 %00110011	$73	115	s	%01110011	$B3	179	³	%10110011	$F3	243	ó	%11110011
$34	52	4 %00110100	$74	116	t	%01110100	$B4	180		%10110100	$F4	244	ô	%11110100
$35	53	5 %00110101	$75	117	u	%01110101	$B5	181	µ	%10110101	$F5	245	õ	%11110101
$36	54	6 %00110110	$76	118	v	%01110110	$B6	182	¶	%10110110	$F6	246	ö	%11110110
$37	55	7 %00110111	$77	119	w	%01110111	$B7	183	•	%10110111	$F7	247	÷	%11110111
$38	56	8 %00111000	$78	120	x	%01111000	$B8	184		%10111000	$F8	248	ø	%11111000
$39	57	9 %00111001	$79	121	y	%01111001	$B9	185	¹	%10111001	$F9	249	ù	%11111001
$3A	58	: %00111010	$7A	122	z	%01111010	$BA	186	º	%10111010	$FA	250	ú	%11111010
$3B	59	; %00111011	$7B	123	{	%01111011	$BB	187	»	%10111011	$FB	251	û	%11111011
$3C	60	< %00111100	$7C	124	|	%01111100	$BC	188	¼	%10111100	$FC	252	ü	%11111100
$3D	61	= %00111101	$7D	125	}	%01111101	$BD	189	½	%10111101	$FD	253		%11111101
$3E	62	> %00111110	$7E	126	~	%01111110	$BE	190	¾	%10111110	$FE	254	þ	%11111110
$3F	63	? %00111111	$7F	127		%01111111	$BF	191	¿	%10111111	$FF	255	ÿ	%11111111

$100	256	%0001 0000 0000
$200	512	%0010 0000 0000
$300	768	%0011 0000 0000
$400	1024	%0100 0000 0000
$500	1280	%0101 0000 0000
$600	1536	%0110 0000 0000
$700	1792	%0111 0000 0000
$800	2048	%1000 0000 0000
$900	2304	%1001 0000 0000
$A00	2560	%1010 0000 0000
$B00	2816	%1011 0000 0000
$C00	3072	%1100 0000 0000
$D00	3328	%1101 0000 0000
$E00	3584	%1110 0000 0000
$F00	3840	%1111 0000 0000

$1000	4096	%0001 0000 0000 0000
$2000	8192	%0010 0000 0000 0000
$3000	12288	%0011 0000 0000 0000
$4000	16384	%0100 0000 0000 0000
$5000	20480	%0101 0000 0000 0000
$6000	24576	%0110 0000 0000 0000
$7000	28672	%0111 0000 0000 0000
$8000	32768	%1000 0000 0000 0000
$9000	36864	%1001 0000 0000 0000
$A000	40960	%1010 0000 0000 0000
$B000	45056	%1011 0000 0000 0000
$C000	49152	%1100 0000 0000 0000
$D000	53248	%1101 0000 0000 0000
$E000	57344	%1110 0000 0000 0000
$F000	61440	%1111 0000 0000 0000

THIS PAGE WAS GENERATED and PAGESET IN 12 MINUTES! *Simon*

NEW COMMANDS

Due to severe time mis-management example code and explanations that make any sense of the following commands have had to be delayed until next issue (yeh sure Simon).

Some examples have been included in the examples drawer of BUM5's cover disk. Also source code to the new GadTools and Display libraries have been included in the libsdev drawer for those capable of making any sense of them.

Window Library Additions

Statement: Window

Syntax: **Window** *Window#,x,y,width,height,flags,title$,dpen,bpen[,gadgetlist#[,bitmap#]]*

The Window library has been extended to handle super bitmap windows. SuperBitMap windows allow the window to have it's own bitmap which can actually be larger than the window. The two main benefits of this feature are the window's ability to refresh itself and the ability to scroll around a large area "inside" the bitmap.

To attach a BitMap to a Window set the SuperBitMap flag in the flags field and include the BitMap# to be attached.

Statement: PositionSuperBitMap

Syntax: **PositionSuperBitMap** *x,y*

PositionSuperBitMap is used to display a certain area of the bitmap in a super bitmap window.

Example:

```
;
; super bitmap example
;

;create large bimtap for our superbitmap window

width=320:height=200 BitMap 0,width,height,2:
Circlef 160,100,160,100,1:Box 0,0,width-1,height-1,3

FindScreen 0

;two sliders for the borders (see new gadget flags next page)

PropGadget 0,3,-8,   $18000+4+8+64,1,-20,8
PropGadget 0,-14,10,$11000+2+16+128,2,12,-20

;reporting of mousemoves means we can track the propgadget as it is moved

AddIDCMP $10
SizeLimits 32,32,width+22,height+20
Window 0,0,0,100,100,$1489,"HELLO",1,2,0,0
```

```
Gosub drawsuper

Repeat
     ev.l=WaitEvent
     If ev=2 Then Gosub dosize
     If ev=$20 Then Gosub domove
Until ev=$200
End

dosize:
     SetHProp 0,1,posx/width,InnerWidth/width
     SetVProp 0,2,posy/height,InnerHeight/height
     Redraw 0,1:Redraw 0,2:Goto drawsuper

domove:
     Repeat:Gosub drawsuper:Until WaitEvent<>$10:Return

drawsuper:
     ww=width-InnerWidth:hh=height-InnerHeight
     posx=QLimit(HPropPot(0,1)*(ww+1),0,ww)
     posy=QLimit(VPropPot(0,2)*(hh+1),0,hh)
     PositionSuperBitMap posx,posy
     Return
```

Statement: **GetSuperBitMap & PutSuperBitMap**

Syntax: **GetSuperBitMap & PutSuperBitMap**

After rendering changes to a superbitmap window thebitmap attached can also be updated with the GetSuperBitMap. After rendering changes to a bitmap the superbitmap window can be refreshed with the PutSuperBitMap command. Both commands work with the currently used window.

Statement: **WTitle**

Syntax: **WTitle** *windowtitle$,screentitle$*

WTitle is used to alter both the current window's title bar and it's screens title bar. Useful for displaying important stats such as program status etc.

Statement: **CloseWindow**

Syntax: **CloseWindow** *Window#*

CloseWindow has been added for convenience. Same as **Free Window** but a little more intuitive (added for those that have complained about such matters).

Statement: **WPrintScroll**

Syntax: **WPrintScroll**

WPrintScroll will scroll the current window upwards if the text cursor is below the bottom of the window and adjust the cursor accordingly. Presently **WPrintScroll** only works with windows opened with the gimme00 flag set (#gimmezerozero=$400).

Statement: WBlit

Syntax: **WBlit** *Shape#,x,y*

WBlit can be used to blit any shape to the current window. Completely system friendly this command will completely clip the shape to fit inside the visible part of the window. Use GimmeZeroZero windows for clean clipping when the window has title/sizing gadgets.

Statement: BitMaptoWindow

Syntax: **BitMaptoWindow** *Bitmap#,Window#[,srcx,srcy,destx,desty,wid,height]*

BitMaptoWindow will copy a bitmap to a window in an operating system friendly manner (what do you expect). The main use of such a command is for programs which use the raw bitmap commands such as the 2D and Blit libraries for rendering bitmaps quickly but require a windowing environment for the user inyerface.

Functions: EventCode & EventQualifier

Syntax: **EventCode** & **EventQualifier**

EventCode returns the actual code of the last Event received by your program, **EventQualifier** returns the contents of the Qualifier field. Of use with the new GadTools library and some other low level event handling requirements.

Gadget Library Additions

Five new flags have been added when defining gadgets in Blitz2. The first four are for attaching the gadget to one of the windows borders, the GZZGADGET flag is for attaching the gadget to the "outer" rastport/ layer of a gimme zero zero window.

#RIGHTBORDER	$1000
#LEFTBORDER	$2000
#TOPBORDER	$4000
#BOTTOMBORDER	$8000
#GZZGADGET	$10000

PropGadgets have been upgraded to take advantage of the 2.0 "newlook" when/if available.

Statement: Toggle

Syntax: **Toggle** *GadgetList#,Id [,On|Off]*

The Togggle command in the gadget library has been extended so it will actually toggle a gadgets status if the no On|Off parameter is missing.

Screen Library Additions

Statement: CloseScreen

Syntax: **CloseScreen** *Screen#*

CloseScreen has been added for convenience. Same as **Free Screen** but a little more intuitive (especially for those that have complained about such matters (yes we care)).

Statement: HideScreen

Syntax: **HideScreen** *Screen#*

Move Screen to back of all Screens open in the system.

Statement: BeepScreen

Syntax: **BeepScreen** *Screen#*

Flash specified screen.

Statement: MoveScreen

Syntax: **MoveScreen** *Screen#,deltax,deltay*

Move specified screen by specified amount. Good for system friendly special effects.

Statement: ScreenTags

Syntax: **ScreenTags** *Screen#,Title$ [&TagList] or [[,Tag,Data]...]*

Full access to all the Amiga's new display resoutions is now available in Amiga mode by use of the Screen Tags command. The following tags are of most interest to Blitz2 programmers: (see autodocs/

#Left=$80000021:#Top=$80000022:#Width=$80000023:#Height=$80000024
#Depth=$80000025:#DetailPen=$80000026:# BlockPen=$80000027
#Title=$80000028:#Colors=$80000029:#ErrorCode=$8000002A
#Font=$8000002B:#SysFont=$8000002C:#Type=$8000002D:#BitMap=$8000002E
#PubName=$8000002F:#PubSig=$80000030:#PubTask=$80000031
#DisplayID=$80000032:#DClip=$80000033:#Overscan=$80000034
#Obsolete1=$80000035

#ShowTitle=$80000036:#Behind=$80000037:# Quiet=$80000038
#AutoScroll=$80000039:#Pens=$8000003A:#FullPalette=$8000003B
#ColorMapEntries=$8000003C:#Parent=$8000003D:#Draggable=$8000003E
#Exclusive=$8000003F

```
#SharePens=$80000040:#BackFill=$80000041:# Interleaved=$80000042
#Colors32=$80000043:#VideoControl=$80000044
#FrontChild=$80000045:#BackChild=$80000046
#LikeWorkbench=$80000047:#Reserved=$80000048

;
; open super wide screen with overscan set for smooth horizontal scroll
; for 2.0 and above with amigalibs.res in resident
;

*_BitMap=$8000002E:*_Overscan=$80000034:*_Width=$80000023:*_Height=$80000024

BitMap 0,1280,512,2:Circlef 320,256,256,1

ScreenTags 0,"TEST",*_BitMap,Addr BitMap(0),*_Overscan,1,*_Width,640,*_Height,512

*vp.ViewPort=ViewPort(0)

While Joyb(0)=0
    VWait
    *vp\DxOffset=-SMouseX,-SMouseY
    ScrollVPort_ *vp
Wend
```

Palette Library Additions

The Palette library has been modified in BUM5 for two reasons. Firstly, it was impossible to perform custom fades using two palettes as the Use Palette command affected the current Slice or Screen. Also with the advent of the Display library the extra properties of the Use Palette command (copy colors to current Slice or Screen) became unwanted.

The ShowPalette command has been added to replace the above functionality removed from the Use Palette command. Also, for compatability reasons NewPaletteMode On is used for enabling the above modifications (default is off).

Statement: **ShowPalette**

Syntax: **ShowPalette** *Palette#*

ShowPalette replaces **Use Palette** for copying a palette's colours to the current Screen or Slice.

Statement: **NewPaletteMode**

Syntax: **NewPaletteMode** *On/Off*

The **NewPaletteMode** flag has been added for compatibility with older Blitz2 programs. By setting **NewPaletteMode** to *On* the **Use Palette** command merely makes the specified palette the current object and does not try to copy the colour information to the current Screen or Slice.

MISC ADDITIONS

Statement: **SortList**

Syntax: **SortList** *Arrayname()*

The **SortList** command is used to rearrange the order of elements in a Blitz2 linked list. The order in which the items are sorted depends on the first field of the linked list type which must be a single integer word. Sorting criteria will be extended in future releases.

Statement: **LoadFont**

Syntax: **LoadFont** *IntuiFont#,Fontname.font$,Y size [,style]*

The LoadFont command has been extended with an optional style parameter. The following constants may be combined:

```
#underlined=1
#bold=2
#italic=4
#extended=8 ;wider than normal
#colour=64  ;hmm use colour version I suppose
```

Statement: **SpriteMode**

Syntax: **SpriteMode** *mode*

For use with the capabilities of the new Display library SpriteMode is used to define the width of sprites to be used in the program. The mode values 0, 1 and 2 correspong to the widths 16, 32 and 64.

Function: **Exists**

Syntax: **Exists** *(FileName$)*

Exists actually returns the length of the file, if 0 the file either does not exist or is empty or is perhaps not a file at all! Hmmm, anyway the following poke turns off the "Please Insert Volume Blah:" requester so you can use Exists to wait for disk changes:

```
Poke.l Peek.l(Peek.l(4)+276)+184,-1
```

Statements: **Runerrson & Runerrsoff**

Syntax: **Runerrson & Runerrsoff**

These two new compiler directives are for enabling and disabling error checking in different parts of the program, they override the settings in Compiler Options.

The New Display Library (#displaylib=143)

The new display library is an alternative to the slice library. Instead of extending the slice library for AGA support a completely new display library has been developed.

Besides support for extended sprites, super hires scrolling and 8 bitplane displays a more modular method of creating displays has been implemented with the use of CopLists. CopLists need only be initialised once at the start of the program. Displays can then be created using any combination of CopLists and most importantly the CreateDisplay command does not allocate any memory avoiding any memory fragmenting problems. The new display library is for non-AGA displays also.

Statement: InitCopList

Syntax: **InitCopList** *CopList#,ypos,height,type,sprites,colors,customs[,widthadjust]*

InitCopList is used to create a CopList for use with the CreateDisplay command. The ypos, height parameters define the section of screen. Sprites, colors and customs will allocate instructions for that many sprites (always=8!) colors (yes, as many as 256!) and custom copper instructions (to be used by the new DisplayFX library currently in devlopment).

The widthadjust parameter is currently not implemented, for display widths other than standard see the DisplayAdjust command. The following constants make up the type parameter, add the number of bitplanes to the total to make up the type parameter.

#smoothscroll=$10	#dualplayfield=$20	#extrahalfbrite=$40	#ham=$80
#lores=$000	#hires=$100	#super=$200	
#loressprites=$400	#hiressprites=$800	#supersprites=$c00	
#fmode0=$0000	#fmode1=$1000	#fmode2=$2000	#fmode3=$3000

For displays on non-AGA machines only #fmode0 and #loressprites are allowed. More documentation, examples and fixes will be published soon for creating displays.

Statement: CreateDisplay

Syntax: **CreateDisplay** *CopList#[,CopList#..]*

CreateDisplay is used to setup a new screen display with the new display library. Any number of CopLists can be passed to CreateDisplay although at present they must be in order of vertical position and not overlap. CreateDisplay then links the CopLists together using internal pointers, bitmaps, colours and sprites attached to coplists are not affected.

Statement: DisplayBitMap

Syntax: **DisplayBitMap** *CopList#,bmap[,x,y] [,bmap[,x,y]]*

The DisplayBitMap command is similar in usage to the slice libraries' show commands. Instead of different commands for front and back playfields and smooth scroll options there is only the one DisplayBitMap command with various parameter options. With AGA machines, the x positioning of lores and hires coplists uses the fractional part of the x parameter for super smooth scrolling.The CopList must be initialised with the smooth scrolling flag set if the x,y parameters are used, same goes for dualplayfield.

Statement: **DisplaySprite**

Syntax: **DisplaySprite** *CopList#,Sprite#,X,Y,Sprite Channel*

DisplaySprite is similar to the slice libraries ShowSprite command with the added advantage of super hires positioning and extra wide sprite handling. See also SpriteMode.

Statement: **DisplayPalette**

Syntax: **DisplayPalette** *CopList#,Palette# [,coloroffset]*

DisplayPalette copies colour information from a Palette to the CopList specified.

Statement: **DisplayControls**

Syntax: **DisplayControls** *CopList#,BPLCON2,BPLCON3,BPLCON4*

DisplayControls allows access to the more remote options available in the Amiga's display system. The following are the most important bits from these registers (still unpublished by Commodore!*()@GYU&^)

	BPLCON2		BPLCON3	BPLCON4
15	*		BANK2 * *active colour bank*	BPLAM7 *xor with bitplane*
14	ZDBPSEL2	*which bitplane for ZD*	BANK1 *	BPLAM6 *DMA for altering*
13	ZDBPSEL1		BANK0 *	BPLAM5 *effective colour*
12	ZDBPSEL0		PF2OF2 *col-offset for playfield 2*	BPLAM4 *look up*
11	ZDBPEN	*makes above bp hit ZD*	PF2OF1	BPLAM3
10	ZDCTEN	*ZD is bit#15 of colour*	PF2OF0	BPLAM2
09	KILLEHB	*	LOCT **palette hi/lo nibble mode*	BPLAM1
08	RDRAM=0	*		BPLAM0
07	SOGEN	*I sync on green*	SPRES1 **sprites resolution*	ESPRM7 *high order color*
06	PF2PRI	*H playfield 1/2 priority*	SPRES0 *	ESPRM6 *offset for even*
05	PF2P2	*H playfield/sprite priority*	BRDRBLANK *border is black*	ESPRM5 *sprites*
04	PF2P1		BRDNTRAN *border hits ZD*	ESPRM4
03	PF1P0			OSPRM7 *high order color*
02	PF1P2		ZDCLCKEN *ZD=14Mhz clock*	OSPRM6 *offset for odd*
01	PF1P1		BRDSPRT *sprites in borders!*	OSPRM5 *sprites*
00	PF1P0		EXTBLKEN *wo blank output!*	OSPRM4

! = Don't touch
H -See standard hardware reference manual
* - controlled by display library
ZD - any reference to ZD is only a guess (just sold my genlock)

Statement: **DisplayAdjust**

Syntax: **DisplayAdjust** *CopList#,fetchwid,ddfstrt,ddfstop,diwstrt,diwstop*

Temporary control of display registers until I get the widthadjust parameter working with InitCopList. Currently only standard width displays are available but you can modify the width manually (just stick a screwdriver in the back of your 1084) or with some knowledge of Commodores AGA circuitry.

Anyway, before I start going on about why they couldn't just give us byte per pixel instead of 8 darn bitplanes (CD32 to the rescue!) see the cover disk for more information...

The New ASL Library (#myasllib=80)

Our policy until now has been that we would only place emphasis on 1.3 compatible commands unless of course they had to do with AGA. Then again I don't even have a LoadWB in my startup-sequence! So instead of complaining I spent an uncomfortable week adding the following 2.0 above specific commands to Blitz2.

And as for those with 1.3 and want new ROMS? BURN BABY BURN...

Function: ASLFileRequest$

Syntax: **ASLFileRequest$** *(Title$,Pathname$,Filename$ [,Pattern$] [,x,y,w,h])*

The ASL File Requester is nice. Except for the highlight bar being invisible on directories you get to use keyboard for everything, stick in a pattern$ to hide certain files and of course you get what ever size you want. I made it call the Blitz2 file requester if the program is running under 1.3 (isn't that nice!). There is a fix that patches the ReqTools file requester but that doesn't have the date field.

I couldn't get the Save-Only tag or the "Create Directory" option working maybe next upgrade.

```
MaxLen pa$=192
MaxLen fi$=192

FindScreen 0

f$=ASLFileRequest$("test",pa$,fi$,"#?.bb",0,0,640,256)

If f$
    NPrint f$
Else
    NPrint "failed"
EndIf

MouseWait
```

Function: ASLFontRequest

Syntax: **ASLFontRequest** *(enable_flags)*

The ASL Font Requester is also pretty useful. The flags parameter enables the user to modify the following options:

#pen=1:#bckgrnd=2:#style=4:#drawmode=8:#fixsize=16

It doesn't seem to handle colour fonts, no keyboard shortcuts so perhaps patching ReqTools is an option for this one. The following code illustrates how a .fontinfo structure is created by a call to ASLFontRequest (just like programming in a high level language man!).

Example:

```
NEWTYPE .fontinfo
      name.s
      ysize.w
      style.b:flags.b
      pen1.b:pen2:drawmode:pad
End NEWTYPE

FindScreen 0

*f.fontinfo=ASLFontRequest(15)

If *f
      NPrint *f\name
      NPrint *f\ysize
      NPrint *f\pen1
      NPrint *f\pen2
      NPrint *f\drawmode
Else
      NPrint "cancelled"
EndIf

MouseWait
```

Function: **ASLScreenRequest**

Syntax: **ASLScreenRequest** (*enable_flags)*

Those who are just getting to grips with 2.0 and above will find this command makes your programs look really good, however I haven't got time to explain the difficulties of developing programs that work in all screen resolutions (what are ya?).

```
#width=1:#height=2:#depth=4:#overscan=8:#scroll=16

NEWTYPE .screeninfo
      id.l
      width.l
      height.l
      depth.w
      overscan.w
      autoscroll.w
      bmapwidth.l
      bmapheight.l
End NEWTYPE

FindScreen 0

*sc.screeninfo=ASLScreenRequest(31)

If *sc
      NPrint *sc\width," ",*sc\height," ",*sc\depth
Else
      NPrint "cancelled"
EndIf
MouseWait
```

The New GadTools Library (#mygadtoolslib=141)

GadTools is a 2.0 and greater extension to the operating system that gives the Amiga programmer a few extra enhancements to create juicy user interfaces with. Instead of listing each as a separate command this issue I'll just add a brief description and a relevant taglist to each of the 12 gadgets.

You are allowed both standard gadgets and GadTools ones in the same window, of course id clashes must be avoided and unlike standard gadgets, gadtools gadgets are attached to the Window after it is open with the AttachGTList command.

GTButton *GTList#,id,x,y,w,h,Text$,flags*

Same as Blitz2's TextGadget but with the added flexibility of placing the label Text$ above, below to the left or right of the button (see flags).

GTCheckBox *GTList#,id,x,y,w,h,Text$,flags*

A box with a check mark that toggles on and off, best used for options that are either enabled or disabled.

GTCycle *GTList#,id,x,y,w,h,Text$,flags,Options$*

Used for offering the user a range of options, the options string should be a list of options separated by the | character eg. "HIRES | LORES | SUPER HIRES"

GTInteger *GTList#,id,x,y,w,h,Text$,flags,default*

A string gadget that allows only numbers to be entered by the user.

GTListView *GTList#,id,x,y,w,h,Text$,flags,list()*

The ListView gadget enaables the user to scroll through a list of options. These options must be contained in a string field of a Blitz2 linked list. Currently this string field must be the second field, the first being a word type.

GTMX *GTList#,id,x,y,w,h,Text$,flags,Options$*

GTMX is an exclusive selection gadget , the Options$ is the same as GTCycle in format, GadTools then displays all the options in a vertical list each with a hi-light beside them.

GTNumber *GTList#,id,x,y,w,h,Text$,flags,value*

This is a readonly gadget (user cannot interact with it) used to display numbers.

GTPalette *GTList#,id,x,y,w,h,Text$,flags,depth*

Creates a number of coloured boxes relating to a colour palette,

GTScroller *GTList#,id,x,y,w,h,Text$,flags,Visible,Total*

A prop type gadget for the user to control an amount or level, is accompanied by a set of arrow gadgets.

GTSlider *GTList#,id,x,y,w,h,Text$,flags,Min,Max*

Same as Scroller but for controlling the position of display inside a larger view.

GTString *GTList#,id,x,y,w,h,Text$,flags,MaxChars*

A standard string type gadget

GTText *GTList#,id,x,y,w,h,Text$,flags,Display$*

A read only gadget (see GTNumber) for displaying text messages.

The parameters x,y,w,h refer to the gadgets position and size, the Text$ is the label as referred to above. The flags field is made up of the following fields:

```
#_LEFT=1  ;positioning of the optional gadget label Text$
#_RIGHT=2
#_ABOVE=4
#_BELOW=8
#_IN=$10
#_High=$20      ;highlight
#_Disable=$40   ;turned off
#_Immediate=$80 ;activate on gadgetdown
#_BoolValue=$100 ;checkbox on
#_Scaled=$200   ;scale arrows for slider
#_Vertical=$400 ;make slider/scroller vertical
```

Statement: AttachGTList

Syntax: **AttachGTList** *GTList#,Window#*

The AttchGTList command is used to attach a set of GadTools gadgets to a Window after it has been opened.

Statement: GTTags

Syntax: **GTTags** *Tag,Value [,Tag,Value...]*

The GTTags command can be used prior to initialisation of any of the 12 gadtools gadgets to preset any relevant Tag fields. The following are some useful Tags that can be used with GTTags:

```
#tag=$80080000
#GTCB_Checked=#tag+4     ; State of checkbox
#GTLV_Top=#tag+5         ; Top visible item in listview
#GTLV_ReadOnly=#tag+7    ; Set TRUE if listview is to be ReadOnly
#GTMX_Active=#tag+10     ; Active one in mx gadget
#GTTX_Text=#tag+11       ; Text to display
#GTNM_Number=#tag+13     ; Number to display
#GTCY_Active=#tag+15     ; The active one in the cycle gad
#GTPA_Color=#tag+17      ; Palette color
#GTPA_ColorOffset=#tag+18 ; First color to use in palette
#GTSC_Top=#tag+21        ; Top visible in scroller
#GTSC_Total=#tag+22      ; Total in scroller area
#GTSC_Visible=#tag+23    ; Number visible in scroller
#GTSL_Level=#tag+40      ; Slider level
```

```
#GTSL_MaxLevelLen=#tag+41 ; Max length of printed level
#GTSL_LevelFormat=#tag+42 ;* Format string for level
#GTSL_LevelPlace=#tag+43  ;* Where level should be placed
#GTLV_Selected=#tag+54    ; Set ordinal number of selected
#GTMX_Spacing=#tag+61     ;* Added to font height to
```

All of the above except for those marked * can be set after initialisation of the Gadget using the **GTSetAttrs** command. The following is an example of creating a slider gadget with a numeric display:

```
f$="%2ld":GTTags #GTSLLevelFormat,&f$,#GTSLMaxLevelLen,4
GTSlider 2,10,320,120,200,20,"GTSLIDER",2,0,10
```

Function: **GTGadPtr**

Syntax: **GTGadPtr** *(GTList#,id)*

GTGadPtr returns the actual location of the specified GadTools gadget in memory.

Statement: **GTBevelBox**

Syntax: **GTBevelBox** *GTList#,x,y,w,h,flags*

GTBevelBox is the GadTools library equivalent of the Borders command and can be used to render frames and boxes in the currently used Window.

Statement: **GTChangeList**

Syntax: **GTChangeList** *GTList#,id [,List()]*

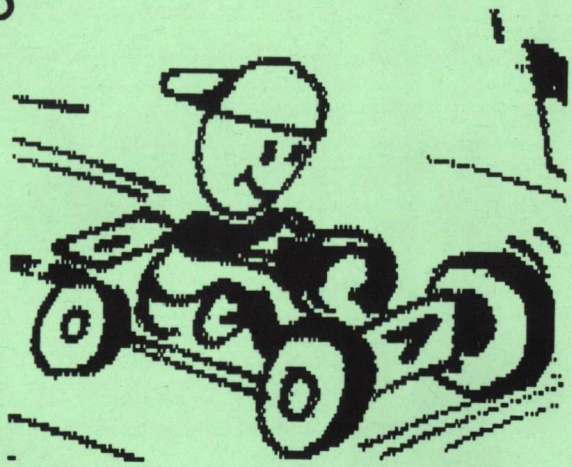
GTChangeList must be used whenever a List attached to a GTListView needs to be modified. Call GTChangeList without the List() parameter to free the List, modify it then reattache it with another call to GTChangeList this time using the List() parameter.

Statement: **GTSetAttrs**

Syntax: **GTSetAttrs** *GTList#,id [,Tag,Value...]*

GTSetAttrs can be used to modify the status of certain GadTools gadgets with the relevant Tags. See **GTTags** for more information on the use of Tags with the GadTools library.

BETA TESTER VERSION
INCLUDED ON BUM5
COVER DISK



TITLE:	**SKIDMARKS**
RELEASE DATE:	**22nd Nov. 1993**
PUBLISHER:	**Acid Software**
NUM TRACKS:	**12**
VEHICLES:	**4**
CUSTOMISED CARS:	**YES**
MODEM CONNECT:	**YES**
MAX PLAYERS:	**4**
MINIMUM MEM	**1Mb Amiga**