LES Debugger AmigaGuide Documentation

COLLABORATORS			
	TITLE : LES Debugger AmigaGuide Documentation		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 29, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

LES Debugger AmigaGuide Documentation

1.1 Main Menu

```
                    LES Debugger v2.1, Internal v1.41
                    =================================

                       Written by Stephen McNamara
                                  of
                          Leading Edge Software

                       Original debugger by
                       Mark Sibly of Acid Software

                              Release Date:
                                19/10/1995

                                 *Note*
    See the section New Stuff for additions and changes in this release
                             of the debugger.

                  This program is OS2.0+ only and requires
                          the reqtools.library v38+
```

1.2 The Index Page

```
                              LES Debugger v2.1
                              =================

                     Please select the topic you wish to read :

          Legal stuff
          About the debugger
          Debugger windows
          Debugger menus
          Installation
          New stuff for v2.1/1.41
          New stuff for v1.3
          New stuff for v1.21-1.0
```

IMPORTANT NOTE:

Installing the Debuglib.obj file into BlitzLibs:System/ is no longer optional. IT MUST BE INSTALLED there. See the section on Installation for more information.

1.3 Disclaimer, Copyright Notice & Public Domain Notice

Disclaimer, Copyright Notice & Public Domain Notice

God I hate formality..... here we go anyway.....

This program is public domain. People may spread it to anyone they like, as long as all the relevant files are included when it is spread. Relevant files include the actual program and all documentation files supplied with the program. PD libraries etc may not charge more than a reasonable price for copying and disks. This program may not be distributed in a commercial package without the permission of the author (Stephen McNamara).

This program, and all associated files are the work of Stephen McNamara. The main debugger program is based on code by Mark Sibly of Acid Software and is used with permission. This program file may not be disassembled or modified in anyway and the docs must not be altered.

The author will not be held liable for any damage, however caused, whether indirect or direct as a result from the (mis)use of this software.

This program uses the reqtools library which is (C) Nico Francois.

1.4 About the debugger

About the debugger

This program replaces the default debugger supplied with B.U.M. 7. It is only of use for people who have version 1.9 of Blitz2 (which was supplied with B.U.M. 7) or later. It cannot be run stand-alone - it can only run from inside Blitz2.

The standard debugger supplied with Blitz2 is very basic in that it only allows you to step programs and evaluate variables. It opened a window on the Workbench screen, in which it had program listing and some program control gadgets. This debugger goes for a full MonAm (the disassembler/debugger supplied with Devpac) approach in that it opens its own screen. On this screen are a series of windows that allow you to

watch what you're program is doing whilst running it, see the section
WINDOWS for information on them.

There are a few things that should be pointed out about using this program. Firstly, it is big and memory hungry, thus you will need to have at least 1meg to use it alongside Blitz2. Also, at the moment anyway, the program uses the ReqTools library via Neil O'Rourke's blitz library. You will thus have to have reqtools.library installed on your machine.

1.5 Windows In The Debugger

Windows In The Debugger
==========

The debugger can have a very large number of windows open at once, each with different functions and uses. Note though that having a large number of windows open at once can cause your programs performance to degrade during trace operations. Since the debugger does no window updating whilst you are RUNning your program, this is not effected.

The windows that can be opened.

Program Control Window
Register window
Source Window
Memory Windows
Object Trace Windows
Variable Trace Window
680x0 Disassembly window
Copper Disassembly Window
Search Memory Window
Array Window
Stucture
Custom Chip Registers

1.6 Program Control Window

Program Control Window
==========

This window contains the gadgets that were found in the original debugger plus a few extra. The extra ones are:

BLTZ : Click to view blitz mode display (if program is in blitzmode). Press a mouse button to return to debugger.
REGS : Toggle register window (open/closed)
MEM1 : Toggle memory window 1 (open/closed)
MEM2 : Toggle memory window 2 (open/closed)
VARS : Toggle variable trace window (open/closed)
ASM : Toggle disassembly window (open/closed)
COP : Toggle the copper disassembly window (open/closed)

```
GS   : Toggle gosub stepping on/off (highlighted=on)
       See New Stuff 1.41 for information.
PS   : Toggle procedure control on/off (highlighted=on)
       See New Stuff for information.
BC   : Toggle Blitz mode control on/off
       See New Stuff for information.

  Old gadgets are:
STOP : Causes program execution to stop and the debugger
       top take over
STEP : Execute the current instruction (whilst program
       stopped)
SKIP : Ignore current instruction and move to the next one
TRACE: Trace the program (like running, except the
       debuggers windows are updated.  Your program will
       run very slowly in trace mode, but you'll be able
       to see exactly what is going on.
RUN  : Resume normal speed execution of the program
<    : Move backwards through history buffer
>    : Move forwards through history buffer
EXEC : Execute a basic instruction
EVAL : Evaluate a value (e.g. variable, addition etc)

Note: All gadgets have keyboard shortcuts associated
      with them.  See the menu option 'Keyboard Shortcuts'
      inside the debugger for the list of keys.

Closing the program control window causes the debugger to
quit and your program to end.

*V1.2*
This window now displays the current mode of your program
inside its title bar.  Possible modes are: AMIGA, QAMIGA
and BLITZ.
```

1.7 Register window

```
                          Register window
                          ===============

This window allows you to view the contents of the 8 data
and address registers.  It will mainly be of use to people
who have a little knowledge of assembler.  As well as the
register contents, you are also shown the 6 words that the
address registers point to.  The status register flags are
also shown in this window, as is the program counter (which
will be mainly used to make the disassembly window open at
the current instruction).
```

1.8 Source Window

Source Window
==========

This window shows you the source code of the program
currently being traced. Whilst it is active, you can use
the cursor keys to move up and down through your code.

V1.2 This window now shows free memory in its title

1.9 Memory Windows

Memory Windows
==========

This windows allow you to look through memory. They are both
identical in that they show memory in both hex and ascii form,
there are two so that you can keep tabs on different addresses
at the same time.
The windows can be moved through memory be using the cursor
keys (and shift for page scroll). You can also press the 'm'
key to enter an address to jump to (or any blitz command that will
a numeric result).

V1.2 You can now edit memory directly inside the Memory
windows. See New Stuff for more information.

V1.3 You can now edit memory in both hex and ascii.
You can also resize memory windows horizontally now.
See New Stuff 1.3 for more information.

1.10 Object Trace Windows

Object Trace Window(s)
==========

These windows allow you to keep tabs on particular objects
whilst your program is being traced or single-stepped.
They hold the structure of an object and will show you
the objects items in the relevant format (e.g. strings will
appear as actual strings, not addresses).
Open another object window by selecting a new one from the
object menu strip.
Change the object number being traced by dragging the slider
which is located on the right side of this window.

V1.2 You can have upto 10 object windows. You must now
close them yourself - they are nor automatically closed
when you select a new object. See New Stuff for more
information

V1.3 You can now open object windows containing

structures that you have defined manually.
A bug has been found that caused object windows to corrupt
if the 'maximum' number of objects for the type was over
127. This has now been fixed.
See New Stuff 1.3 for more information.

1.11 Variable Trace Window

Variable Trace Window
=====================

This window allows you to trace different variables whilst
your program is running. It has been completely fixed to
work fine with all variable types (strings now work all the
time!).
Add variables to the trace list by selecting the Add Trace...
menu item. You'll be asked to enter the name of the variable
to trace, then the type, and then the output mode (if relevant
to your type). Output mode is either DECimal, HEX or BINary.
Variables can be added whilst this window is closed, but you
must open it yourself before tracing will start.
Variables can also be added directly by the program being
traced by using the AddVarTrace library command (see
documentation for the RIDebug Library).

NOTE Variables must be added *after* they have been
defined.

New option: there is now a menu option that sorts all
variables added into alphabetical order.

V1.2 As of this version, you can define the type of
a variable with its name, doing this *validly* causes
the type requester to be skipped. The following are
now valid in the name requester (after the name of a
variable):

    $
    .s
    .b
    .w
    .l
    .q
    .f

1.12 680x0 Disassembly window

680x0 Disassembly window
========================

This window displays disassembly of memory into 680x0
instructions. See New Stuff for more information
(specifically: V1.0 new stuff).

V1.3 Disassembly window is now laid out slightly
differently to allow displaying of breakpoint information
All breakpoint readouts are in hex.
The disassembly window is now locked to PC-2, meaning that
it always displays disassembly of the next basic
instruction to be executed.
See New Stuff 1.3 for more information

1.13 Copper Disassembly Window

Copper Disassembly Window
==========

V1.2
This window displaya dissassembly of memory into copper
instructions. The following 3 instructions will be
shown:

```
MOVE <$VALUE> <REGISTER>
WAIT <$xx,$yy> MASK <$xx,$yy>
SKIP <$xx,$yy> MASK <$xx,$yy>
```

The WAIT and SKIP instructions will sometimes have [BFD]
written after them. If present, this means that the
instruction has "blitter finish disable' on.
See New Stuff for more information.

1.14 Search Memory Window

Search Memory Window
==========

V1.21
This window allows you to control the debuggers memory
searching facilities. From here, you can enter the start
and end addresses for the search, the value to search for
and the type of the value. You can search for the all 6
basic variable types: byte, word, longword, string, quick
and float.

1.15 Array Window

Array Window
==========

V1.3
This window allows you to easily view the contents of a
one or two dimensional array of bytes,words or longwords.
The display is in hexadecimal, with array indexes shown

```
  across the top and down the left side of the window.
  Scroll bars are positioned in the window borders for when
  the array is too big to fully display.
```

1.16 Debugger Menus

```
                                Debugger Menus
                                --------------

  Here is a the menu structure for the debugger, along with an explanation
of the items.

  [ITEM]    [KEY] [DESCRIPTION]
<Project>
  Status Register   Display flags for the status register
  About debugger    Display program information
  About LES     Display info about Leading Edge
  ---------------------
  Save configuration  W Saves out the current debugger setup,
        including screen info, window positions
        and window status (open/closed & size)
  Screen mode...    Allows control over  screen mode
  Window height...    Allows selection of maximum window height
  ---------------------
  Quit        End the debugger and the program being
        debugged
  Quit & Die      End the debugger - leaving debug process
        active (ONLY USE AFTER A DEBUG PROCESS
        CRASH!)

<Windows>
  REGS      D Open the Register window
  MEM1      M Open Memory window 1
  MEM2      N Open Memory window 2
  VAR     V Open Variable trace window
  ASM     6 Open Disassembly window
  COP     C Open Copper disassembly window
  GS                            Toggle gosub stepping ON/OFF
  PS      P Toggle procedure stepping ON/OFF
  BC      B Toggle blitz control ON/OFF
  ---------------------
  Search Memory   S Open the memory search window
  Array window... R Open the array window (with prompt for
        array to open as)
  Structure...    T Open an object window as a structure.
        You will be prompted for the name of
        structure to find.
  Custom chips...   Open the custom chip window

<Object>
  <Object name>     Open the selected object inside a window

<Misc>
  Add Trace...    A Add a variable trace
  Del Trace...      Delete a variable trace
```

```
  Del All...      Delete all traces
  ---------------------
  Auto EVAL...      Enter a string to EVALuate after every
        line of code is stepped.
  ---------------------
  Add breakpoint... [ Add a breakpoint
  Delete...   ] Delete a breakpoint
  Delete All...     Delete all breakpoints
  List...     List all current breakpoints
```

1.17 Installation/How To Use

```
                          Installation/How To Use
                          -----------------------
```

Installing is done is two parts, these being installing the debugger executable and installing the library object files needed to let your basic programs communicate with the debugger.

Installing the debugger program couldn't be easier. Just copy the defaultdbug file into your Blitz2:Dbug/ drawer and load up Blitz2. Before you do this, though, copy the original version of the defaultdbug file to a storage area (or rename it to something like olddbug). Do this in case you don't like this debugger, or find it unusable for some reason (tell me about this though!). Now whenever the debugger is invoked (by running a program that crashes, or by pressing Ctrl-Alt-C) the new debugger will pop up and offer you its services.

There are two library objects that need to be installed, they both need to be copied to folders in your Blitzlibs: volume. The files are:

RIDebugLib.obj - copy to Blitzlibs:Userlibs/. This library gives you the commands that allow you to order the debugger about.

Debuglib.obj - copy to Blitzlibs:System/. This library in an Acid Software one that has been extended. It is required by the debugger. If you do not install this library, the debugger will not function at all.

switchlib.obj - copy to Blitzlibs:System/. This library in an Acid Software one that has been extended. It is required by the debugger. If you do not install this library, the debugger will not function at all.

When both these objects have been copied to the relevant folders, you should run the program MakeDefLibs which can be found on your original Blitz 2 disks. This will remake your Blitz2:deflibs file based on the contents of the Blitzlibs: volume. You should be aware that on floppy this will be very slow. When MakeDefLibs has finished just load blitz and you'll have access to the new debugger and the additional commands supplied by the RIDebug library. Please see the additional docs for the RIDebugLib

```
library for information about commands.
```

1.18 Defining Structures

```
                          Defining Structures
                          -------------------

  When using structures the debugger requires an extra file to hold
information about offsets into an ascii file which itself defines all the
structures.  This 'offset' file is produced by using the program
MakeOffsets.bb2 (which can be found withing the Debugger archive).  This
program allows you to select files to be included in a single offset file.
The debugger can only have one offset file, so you must include all your
structure files inside this.

  An offset file looks like this:

!Assembly:Development/structure.offset    ------- Source file
00000000 AChain         ------- File offsets
000000DD AmigaGuideHost          (hex) and
000001DE AmigaGuideMsg           structure names
0000032D AnalogSignalInterval
000003AC AnchorPath
000004F3 AnimComp

The offset file can contain multiple lines starting with !, meaning that
you can have multiple structure files indexed from inside one offset file.
There must be ***NO*** comments inside this file though.

  Structure files are defined in the same format as the Amiga Includes file
structure.offsets.  This file has a strict layout as follows:

MYStructure:        --- name of structure
  $0024    36  sizeof(MYStructure)  --- must be in!
  $0000     0     4  pointer    --- first item in structure
  $0004     4     4  another    --- next item


The items are in order:

    $0000   - HEX offset from start of structure for
          this item
        0   - DECIMAL offset from start of structure
          for this item
        4   - DECIMAL size of this item


Your structure file must be written out in this format.  When finished you
should use the MakeOffsets.bb2 program to build an offset file for it.

  Supplied with the debugger is an offset file for V40 of the Amiga
Includes.  Unfortunately since the actual file structure.offsets is
copyrighted it cannot be supplied along with the debugger.
```

1.19 New Stuff for V1.41

```
                          New Stuff for v1.41
                          -------------------
```

This version has had loads of minor bugs that were found (by myself and other LES members) in V1.3 of the debugger. This version, though, may still itself not be completely sound on all systems - it is relatively untested (due to the fact that people who took copies of V1.3 never reported back about it).

Note that this version now identifies itself as V2.1 of the debugger, this is to keep in line with Acids current numbering system.

The following are changes made between versions 1.3 (beta version) and 1.41 of this program.

```
                    Gosub Stepping
                    Bump Prefs
```

1.20 Gosub Stepping

```
                          Gosub Stepping
                          --------------
```

The debugger now allows you to step over calls to subroutines via the GOSUB instruction as if they were a single instruction. Control of this mode of operation is via the 'GS' gadget/menuitem. If on, STEPping a gosub command will cause the debugger to run the program being debugged until the subroutine RETURNS back to the calling routine.

Note that if you step a subroutine whilst in blitz mode, the blitz mode display is redrawn - even if you have Blitz Control on (via the BC gadget/menuitem). Also note that with this function on, you can only step one line of code at a time.

1.21 Bump Prefs

```
                          Bump Prefs
                          ----------
```

The executable inside the debugger directory named BumpPrefs can be used by users of previous versions of the debugger to make their saved configuration information usable by this version of the debugger. The versions it can handle are:

```
      V1.3ß
      V1.4
      V1.41
```

These versions can and will be bumped to version 2.1 of the debugger.

1.22 New Stuff for V1.3

```
                          New Stuff for v1.3
                          ------------------
```

There have been quite a lot of behind the scenes changes to the debugger since the last general release. For a start, none of the debuggers windows are gimme-zero-zero windows. For those of you who don't know what this means, all you have to know is that less memory is now taken up and redraw times are better on some windows.

An extra feature has been added to all windows that accept the use of the 'm' key to enter a base address for them. As of this version, pressing <Shift>+'m' will take the value held in the display part of the Source Window and use it as the base address of the window. Thus if you had just EVALuated the value of variable 'address' you could make a memory window point to the value given just by activating it and pressing <Shift>+'m'. Leading on from this is a slight extension to the object windows that allows you to set their base object number by pressing 'm' or <Shift>+'m'.

1.23 Custom Chip Registers

```
                       Custom Chip Registers
                       ---------------------
```

This window allows you to easily fine the address of a custom chip and vice versa, allows you to find the custom chip name that an address refers to.

The window contains two string requesters:

Address$ Enter an address here. NOTE that the number should be given in HEX. The number can either be in the form of a full address (e.g. dff002) or just as an offset from $dff000 (e.g. 2).

Name The name of the custom chip. Type a name in here to get the address of a custom chip register.

1.24 Screen Mode Selection

```
                       Screen Mode Selection
                       ---------------------
```

The debugger now supports control of its screen mode by the user. Previously the user did not have any control over the type of screen the debugger opened, the screen was always 'Like Workbench' on OS3 machines or 640xdispheight on OS2 machines. Now the debugger sports a screen mode requester that allows you to select the size and resolution of the screen to open. You have three options with regards to screen mode:

'Workbench' - you can now select to open directly onto the Workbench screen. This saves memory and is more convenient if the program being debugged opens on Workbench itself.
'Like Workbench'- the debugger opens up a screen that matchs the display mode, width and height of your Workbench screen. The depth is always set to 2 bitplanes (4 colours).
'Custom' - when you select this a screen mode requester will open allowing you to select a display mode and screen size for the debugger.

After changing the screen mode you ***MUST*** save your configuration, otherwise your changes will be lost. Changes do not come into effect until the debugger is run again. Note that the minimum allowable screen size is 640x200.

As an extra for those users with lovely multisync monitors, you can now specify the maximum height of the debuggers windows. This is done after you have selected a screen mode from the mode window. This option allows you to change the internal buffer that the debugger keeps to allow fast printing and window updating - the bigger the height, the more chip mem this buffer will take up. The minimum height of the buffer is 200 pixels, the maximum is 1024 pixels.

1.25 Structures

Structures

Now to a big update... This one is mainly for use of people who do a lot of work with the operating system and need access to the OS defined structures detailed in the Amiga Include files. Inside the includes is one big file called 'structure.offsets' which contains an entire list of all the structures. This file has a very specific layout, meaning that the debugger can easily interpret its contents. So what we now have is an extension to the object windows that lets you open an OS structure inside an object window. The structure is displayed in exactly the same way as the standard object windows but there is a slight difference. If you click on any value inside the window a request will open asking you if you wish to move the base address of the window to the address given in the value. This is ideal for when you are trying to navigate round a device or library list, just click on 'next' or 'prev' to move backwards and forwards through the list.

This extension to the object windows can be controlled by the user of the debugger, who can specify his own structures if he so desires. The structure details need to be formated and placed inside a file where the

debugger can find them.

See Defining structures for more information.

1.26 Locking

Locking

It is now possible to lock the memory windows to an expression. This expression is made up of numeric values and register names. You CANNOT lock windows onto a variables value. The debugger uses a complex expression evaluater, which can except the following tokens:

Note that float values cannot be used. You can use parenthesis (brackets) to group arguments together.

$ - The following number is hexadecimal
% - The following number is binary
+ - Add two operands
- - Subtract two operands
* - Multiple two operands
/ - Divide two operands
= - Test for equality between operands
< - Test for less than
> - Test for greater than
^ - Power operation
& - Logically AND two operands
| - Logically OR two operands

This function will mainly be useful for assembly programmers since it allows you to trace code and have a window continuously pointing to a value held in a register.
The disassembly window now uses this 'lock' function to lock onto the address PC-2 - meaning that it always displays the disassembly of the next Blitz instruction to be executed.

1.27 STEP enhancement

STEP enhancement

A little quick one, now if you press the <CTRL> key whilst stepping (by using the gadget or the keyboard shortcut) you will be asked to enter how many instructions to step. Thus you can step through 50 instructions in one good - without having to repeatedly press 'STEP'. Note that when you use this feature Auto-EVAL will not function.

1.28 Memory Windows

Memory Windows

Memory windows can now be sized horizontally as well as vertically. When the width changes the debugger evaluates how much it can display inside the window and adjusts its contents accordingly. The debugger can only display multiples of four bytes in the window.

Memory Editing

It is now possible to edit memory in hex and ascii. The initial mode when the debugger first runs will be hex, to change to ascii just press <TAB> whilst in edit mode (go into edit mode by activating a memory window and pressing <SPACE>). When in ascii mode an characters you type will be written directly to memory. <SPACE> and <TAB> are interpreted differently to other keys though, since <SPACE> exits edit mode and <TAB> changes from ascii to hex mode and vice versa. To write the characters <SPACE> and <TAB> to memory you should hold down the <CTRL> key whilst pressing them. This overrides their functions and writes them directly to memory.

Memory Window Locking

Memory windows can now be locked to an expression. See Locking for more information.

1.29 Breakpoints

Breakpoints

For those of you not familiar with MonAm, breakpoints are a way of stopping your program when a certain instruction is executed. They are equivalent to a STOP instruction placed in your source code.

Breakpoints are set up via the Disassembly window. To set up a breakpoint on an instruction you just have to click on the basic line of code inside the disassembly window.

The debugger allows you to set up 3 different types of breakpoint on your code:

Countdown: these breakpoints have a counter which says how many times they can be passed before the program is stopped. Thus you can have a breakpoint stop you program after a specific line of code has executed a specific number of times. After the counter has reached 0 the breakpoint will be cleared.

Permanent: these always stop your program when they are reached. They are not though cleared when your program is stoped by them.

```
  Counter:  these don't actually stop your program, instead
      they just count how many times your program went
      passed them.

Breakpoints are displayed in the disassembly window in the following way:

  Countdown:  [xx]  <Source code>
  Permanent:  [*] <Source code>
  Counter:  [=xxxx] <Source code>

These displays replace the standard '>>>' which precedes a basic
instruction in the disassembly window.  Breakpoints can only be set up on
basic instructions - they cannot be setup on the assembly code of basic
instructions.

The debugger allows you to set up upto 5 different breakpoints inside your
program.  Each breakpoint *must* be set at an unique address.  You should
note that breakpoints slow your program down slightly - have as few as
possible going at once if you want maximum speed.
```

1.30 New Stuff

```
                          New Stuff for v1.21
                          -------------------

  This version is a slight update to v1.2 (which was a beta version) that
fixes some problems found.  There were somw problems regarding blitz mode
control that have been sorted out, as well as some library problems.

  One major update has been made to the debugger in that you can now search
memory for values and strings.  This is done via the search window, which
is opened by selecting the Search option from the menus.

  There has also been an extension to the save configuration function that
lets you save the position, size and status of every window in the debugger
except object trace windows.

  Finally, there are now several example programs that show off a few
features of the debugger to you.


                          New Stuff for v1.2
                          ------------------

  Quite a few additions and improvements have been made to this version of
the debugger.  New features have been added as well as improvements and
bugs fixes (which have come about from reports by people who've tried the
debugger out).

  Summary of new features:
        Procedure control
        Blitz mode control
        Coplist Object tracing
        Copperlist Disassembly
```

```
        Multiple Object Windows
        Configuration Saving
        Memory Edit Mode
        Data/Address register edit
        Program mode display

  Improvements made:
        Loads of bug fixes
        Speed improved for redrawing etc
```

Procedure Control

  This function allows you to skip over procedures as if they were single instructions when either STEPing or TRACEing your program. When running, this function has no effect. When you use this function, the procedure runs exactly the same except that it no longer causes the debugger to update its windows whilst it is executing.
  In effect, what this function does it to, at the start of a procedure, which the debugger into RUN mode, and then at the end chuck it back into its previous mode.

Blitz mode control

  This function is for all those people who're getting annoyed at the debugger constantly rebuilding the blitz mode display every time they step an instruction or evaluate an expression. With this function on, singular calls to the debugger like evaluating and execing no longer cause Blitz to redraw the display.
  To understand this you have to have a little knowledge of what blitz does when it goes into blitz mode. When blitz mode is activated, Blitz disables multitasking, sets up a custom copperlist and owns the blitter. What this function does is to stop blitz setting a custom copperlist. The program being debugged *IS* in blitz mode, it just doesn't have the display. This may not be useful at times (for when you need, for example, to check to see where a blit is occuring) - you must decide yourself when and where to enable/disable it (default is disabled). That said, this function is very useful, since it stops all flashing, including flashes for opening object windows etc.

Coplist Object Tracing

  The Coplist object type (used in the Display library) has been added to the debuggers list. The whole object could not be included, as it is fairly massive, so the object holds *most* of the object definition (all the important bits are in). This object was added mainly for use with copperlist disassembly.

Copperlist Disassembly

  You can now disassemble copperlists directly from inside the debugger.

Just select the option and you'll be given a nice window, where you can use cursors to scroll up and down, plus use the 'm' key to enter a start address. All this comes in useful when combined with the Coplist object, and some custom copperlist coding. Now its possible to check what Blitz is doing with your custom commands in the display library.

Multiple Object Windows

You can now have upto 10 independent object windows opened at once inside the debugger. This should be more than enough for everyone. A quick warning, though, with this many windows open, unless you have an accelerated Amiga, you will slow the debugger down a great deal. Like all the windows in the debugger, you should only have open the windows that you are actually using.

Because of the additional object windows, selecting a new object from the menu strip *will* not replace any current windows. You must close individual object windows yourself after you have finished with them. Also note that it is perfectly allowable to have, for example, two bitmap windows open. Thus allowing you to trace more than one object of each type at once.

Configuration Saving

A default config file for the debugger can be saved now from inside the debugger. Inside the config file, are size, position and status information for the following windows:

```
Source    - MUST BE OPEN!
Registers
Control   - MUST BE OPEN!
Memory1
Memory2
```

When the debugger first runs, it will look for the volume ENV:. If it finds it it'll then attempt to load the file ENV:BB_DBug.prefs. If it can find this, your prefered window layout will be loaded, otherwise the default layout will be used.

Please note that the debugger will not bring up any requesters if it cannot find the file. When the debugger saves its preferences file, it saves to both ENV: and ENVARC: so both of these should be made available if you want to save your config (this will probably mean floppy users having to put there boot disk into a drive).

At the moment, the configuration file is 64 bytes in size.

Memory Edit Mode

Whilst a memory window is active, pressing <SPACE> will put you into edit mode. Here you can move a cursor round the windows hex display, and edit using 0-9/A-F on the keyboard. Press <SPACE> again to exit, or deactivate the window by selecting another (or pressing TAB).

NOTE: You should be careful what you change in the edit window. Only change/edit memory that your program owns! Do not go messing around through memory as you could cause a crash.

Register Edit

Clicking on registers d0-d7/a0-a6 in the register window will now bring up a requester into which you can type a replacement value for the register. The value can be typed in as a number or as a 4 character string, surrounded by quotes. Thus the following are allowed:

```
$fff0
12466
%1001111111100000
"CMAP"
"ANHD"
```

Program Mode Display

The current mode of your program is now shown in the title of the Program Control Window. The mode will be one of AMIGA, QAMIGA or BLITZ.

SPEED.....

Speed improvements have been made to several areas of the debugger. The main improvements have been in text printing and window updating. By using the Configuration Saving, though, you can speed up debugger initialisation by closing extra windows.

Odd Addresses

Odd addresses can now be displayed properly in the register window and the two memory windows. Also the memory windows can be moved one byte at a time using the left and right cursor keys.

Variable Window

You can now sort all currently traced variables into alphabetical order by selecting the 'Sort Names' menu item.

Bug Fixes

Just a few bugs that have been fixed:

o Hex display in memory and register window was sometimes displayed

incorrectly due to a mistake in conversion. This effected odd addresses only.
- o BLITZ gadget has been sorted so that clicking out of the BLITZ screen will not immediately reactivate the gadget.
- o Loads of OS2.0/68000 compatibility problems sorted (e.g. peek.l at an odd address).
- o Wrong font used inside the Object window (used default WB font rather than topaz.8).

Problems with string printing inside the object window sorted

New Stuff for v1.0

So whats new in this release of the debugger? Well there are loads of changes and updates that have been made to the debugger, these being:

680x0 Disassembly

You can now disassemble your blitz basic programs directly into 680x0 (68000 to 68020) instructions. When runtimes are on in a blitz basic program, the TRAP instruction is used before every basic instruction in the program. When the disassembler finds the correct trap (#1 is used by blitz) it automatically replaces this instruction, plus the 8 data bytes after it, with the line of blitz basic code that the following 680x0 belongs to. This doesn't always work, though (for some reason - I'll blame blitz for this :)) so you *might* not always display the correct line of source (multiple commands on one line, plus blank or commented lines bugger up).

The cursor keys, plus [SHIFT] can be used to move up and down through memory when the disassembly window is active. Also, pressing 'm' will bring up a requester where you can enter an address to move the disassembly window to.

Blitz Gadget

Clicking this gadget allows you to view the blitzmode display of the program being traced. When you have finished viewing the display, press the mouse button to return to the debugger.

AutoEVAL

The EVAL button in the debugger allows you to evaluate any variables, find label addresses etc. AutoEVAL takes this a step further and allows you to enter a line that will be evaluated after *EVERY* instruction is single stepped (using the STEP gadget). To enter a line to AutoEVAL just select the menu option, when a non-null string is entered, the AutoEVAL will be enabled and will display its output after every step at the bottom of the source window.

An example of how this can be used is to evaluate an expression like mymap(x,y) after every instruction is stepped. This can be used to easily keep track of what your program is doing. The output of AutoEVAL for this example instruction will look like this:

```
  mymap(x,y)=50
```

You can only AutoEVAL one line at a time. Also, you should note that the AutoEVAL does degrade the speed of your program, since the EVAL command itself is slow.

  Remove the AutoEVAL by selecting the menu option and clearing the command string.

Standard Gadgets

  The menu items that mirror the standard gadgets in the program control window have been removed. They have been replaced with keyboard shortcuts for the gadgets that do not require the right amiga to be pressed to activate them.

New Objects

  The list of objects that can be traced inside the debugger has been extended to include all the main objects I could think of. The full list of objects is:

- Bitmap (extended to hold the *full* bitmap definition)
- Blitzfont
- Buffer
- File
- Intuifont
- Module
- Palette
- Queue
- Slice
- Sound
- Sprite
- Shape
- Stencil
- Tape
- Window

RIDebug Library

  This library allows you to give the debugger 4 basic instructions. It mainly allows you control over the variable tracing facilities inside the debugger. Please see the RIDebugLib docs for more information about how to use the library.

Bug Fixes

  Too many to mention :)

1.31 Problems...

```
                                Problems...
                                -----------

  A couple problems that I know of that people have had with the debugger
in the recent past.  As more problems are reported, I'll extend this list
so that it becomes like a FAQ for the debugger.

  A quick plea to possible bug reporters:

  When you report bugs to me about this program, can you please
include details of the machine the debugger is running on, including amount
of memory and an 'add-ons' you may have, and the operating system version
number you are using.  The more I know about what situation the debugger is
running in, the easier it'll be for me to track down possible bugs or help
you with setting it up.

 Prob 1: I cannot trace any variables.  The debugger always responds with
"Variable not found" when I do.

 Solution: There are two main reasons for this happening:

  a) You are typing the variable name wrong.  Remember that Blitz
names are case sensitive, also make sure that any strings defined with a
'$' instead of '.s' must be added with '$' after there name.

  b) You are being affected by a bug in the current version of Blitz.
The debugger uses the EVAL function (the gadget on the debugger screen)
that Blitz provides to locate a variable in memory.  What happens, though,
is that if a particular library is not included in your program when you
compile, you CANNOT access the EVAL function.  Thus the debugger gets 0
back from its locate request, and tells you the variable doesn't exist.
     To get round this bug, you must have a command like Print used
in your program.  The command DOESN'T have to run, it is only there to make
sure that Blitz includes the correct library when compiling your program.
What you can do is just add 'Print ""' to the end of your program (this is
what I do)


 Prob 2: I cannot trace objects.  I just get an 'No object found' message in
the object window.

 Solution: Again, this problem is caused by one of two things:

  a) The objects don't actually exist in your program.  For example,
if you never use the Blitz Slice library to create Blitz displays, you
won't be able to look at slice objects since there won't be any in your
program.

  b) The Blitz bug crops up again.  This function is effected in
exactly the same way as Variable tracing by a bug in the EVAL function.
Add a print command to your program to solve this problem.
     See Problem 1 for more info about this bug.

 Prob 3: CTRL-ALT-C doesn't make the debugger pop up.
```

Solution: Quite a few for this one:

a) Your program may have crashes. Check this by going to the debugger screen (is possible) and trying to evaluate an expression or variable. If you don't get an answer back - your program has gone bye-byes.

b) The debugger has crashed. This may or may not cause your program to crash. Most likely a crash will result eventually.

c) You have a ZAPPO CD-ROM drive installed (like me!). I have had problems with using programs that want to allocate CIA resources with the driver software loaded. I get a lot of 'Resource already in use' messages when I run particular software. When I have the drive installed, Blitz refuses to acknowledge any keyboard responses (except through windows). I have tracked down the fault to the fact that the CD driver software is in memory, but I do not know what is causing the problem.

Prob 4: Debuggers menus all appear in the same colour and are unreadable.

Solution:

The debugger defaultly opens it window 'like workbench', but it does not mimmick the Workbench screens depth. Meaning that if you have your Workbench screen in more than two bitplanes AND have default screen pens that use these extra colours you will be a confused debugger screen. The answer to this is to either have the debugger open on the Workbench screen by default or to adjust your screen pens back down into the bottom four colours of your palette.

1.32 Version history

Version history

For those interested in the development of this program, here's the version history for it as taken from the top of the source code. Pretty pointless putting it here but what the hell - somone might read it ;-).

If you have a bug to report can you please first check through this section to make sure that it hasn't already been sorted. Thank you.

Last change: 16:25, 17/4/95

17/4/95
- Added breakpoints. Can only be placed on basic instructions (no breakpointing the asm code of instructions). Can have upto 5 breakpoints ATM, 3 diff. types of breakpoint supported.

16/4/95
- Revision bump: .3
- Register window now shows PC-2 (now points to TRAP inst.)
- Added Screenmode requester, can select either

```
          'Workbench'
          'Like Workbench'
        or 'Custom'.
   - Added a maxheight option for windows - from 200 to 1024 tall
   - Memory can now be edited in ASCii.  [TAB] swaps between
     editing in HEX & ASCii.  Pressing <CTRL> and <SPACE> or <TAB>
     allows you to write these to memory (instead of them being
     interpreted as commands).
   - Structure window now remembers the name of the last structure
     opened.
15/4/95
   - Pressing <CTRL> whilst clicking STEP or pressing S allows
     you to enter number of commands to step
   - Memory windows can now be sized horizontally
   - Fixed bug in mem window edit - stopped cursor left working
     if address=0
14/4/95
   - Can now lock memory windows to any register
     Can also use an offset, e.g. lock window on PC+8
     or A6-4.
   - Fixed setting register values - was not setting address
     registers properly
11/3/95
   - DisAsm window is now locked to current source instruction
27/2/95
   - Fixed Disassembly window - now properly prints the command
     text
   - Can not now enter value for register without the debugger
     being STOP mode.
   - Going into edit mode in memory windows now delays to stop
     key repeat from exiting edit straight away.
23/2/95
   - Debugger now understands the structure.offsets file format
     and can scan through for a structure + open it in a window.
17/2/95
   - Now use M and SHIFT-M on object windows
9/2/95
   - bitmap object now has xcurs + ycurs items
6/2/95
   - Press SHIFT-M to grab the currently display EVAL result
     as the start address for mem/disassembly windows.
   - Removed printing of return values from procedure control,
     also removed forbid_ and permit_.  These were responsible
     for blitzmode cockups when procedure stepping.
1/2/95
   - Object definitions now have sizeof.type word at front end
   - Object windows store base address of type so that the debugger
     can locate the address of each object number itself.
31/1/95
   - Object windows are no longer gimmezerozero windows
29/1/95
   - Maxobject for object windows was being held as a byte instead
     of a word - PRATT!  IDIOT!  TWAT! :)
18/1/95
   - Release VERSION: 1.21
17/1/95
   - Fixed bug: when pressing escape to quit the memory search,
```

```
     the program exited - the ESC key hit was being sent as an
     event.  Fixed by flushing all keyboard events (1024) from
     the event list.
11/1/95
   - Search memory in and mostly working.
     Searches for: Bytes, Words, Longs, Quicks & Floats.  Strings
     are to be added.
   - Windows automatically activate when opened AFTER end_init
     goes non-zero (at the end of initialisation)
   - CTRL \ now closes the current window (except: #_winSource,
     #_winBackdrop & #_winControl)
   - Slight changes to RedrawVarWin{} - now uses FPrint instead
     of window printing
   - Config file now saves Var/Cop/680x0 windows so these can
     now be open by default.
   - Bug fixes in blitz mode handling
7/1/95
   - BETA TEST: Version 1.2
   - Config file now 2 bytes bigger.  Saves ProcStatus and
     blitzcontrol at end of file.
   - NEW BLITZ MODE HANDLING!  Can now stop Blitz mode display
     being recreated whilst stepping/tracing/EXECing/EVALing
6/1/95
   - Procedure stepping now controlled by debuglib library.  The
     library will call d_prochandler if variable procstatus is
     non-zero
   - Extra gadget PS: procedure tracing status.  If highlighted,
     procedures are treated as a single command.
5/1/95
   - New debug command causes copper disassembly window to
     open at the given address
   - When adding a variable trace from the debugger you can
     now specify the type directly, e.g. a.s,b$,d.q etc
   - Debug messages have changed: addresses are given as one
     longword rather than a hex string
4/1/95
   - VERSION: V1.2
   - Edit mode toggled by pressing <SPACE> *WHILST*
     a memory window is active
   - Program running mode now shown in control window title
   - Exiting MemEdit mode now redraws both windows (in case
     the windows' addresses overlap)
   - COPPER gadget added - Copper Dis... menu item removed
   - PREFERENCES FILE NOW SAVED TO ENV: and ENVARC:
     Loaded from ENV: (if available!)
   - Preferences file now has a program version number
     in first byte and revision number in second byte.
     Then two byte pad before window data (could hold number
     of window definitions?)
1/1/95
   - d_eval{} and d_exec{} now increment stepcnt themselves
31/12/94
   - Bug when moving mem window to label address. Was only
     adjusting MEM1 instead of MEM1 and MEM2
   - TAB key now cycles all windows properly
   - MemEdit now exits when window gets deactivated
   - Can now move mem windows into negative numbers from 0
```

```
30/12/94
   - Menustate command was trying to deactivate an unitialised
     menu item.  Command has been removed
   - Fucked up .xtra file..... causing gurus (not enough
     windows.......)
26/12/94
   - Trial custom font printing
24/12/94
   - VERSION: V1.1
   - Object window: data types now shown on left of central
     line and are not rewritten every refresh (only during
     full redraw)
   - Added copper instruction disassembling
   - New command in library: GetCopIns$ - convert copper
     instructions into strings
   - Library changes:
     - D_CheckNMove now takes an optional count parameter
     - Changed the way D_GetSR works
   - Fixed BLITZ gadget disable/enable (AGAIN!)
   - Capital letter for object names
   - Objects can now contain quick and float types
   - Added coplist object type - whole definition won't fit
     inside a 256 tall screen though :)
   - Added checks in NewTypeWindow{} to make sure the window
     will never be larger than can fit on the screen
21/12/94
   - CheckNMove syntax change: now GIVE address to take 4 bytes
     from rather than giving the actual longword of data
20/12/94
   - Added 'Sort names" menu item for variable window - sorts
     names into alphabetical order
   - Click in register window to modify d0-d7/a0-a6
   - Memory window have been widened to 208 pixels to allow
     odd addresses to be displayed properly
   - Left/Right cursor now move memory windows 1 byte at a
     time
19/12/94
   - Bug when workbench screen was 640 wide:
     Control window was opened at x=#_winControl rather than x=0
17/12/94
   LOADS of stuff :)
   - Bug fixed in 'hexwordodd - wrong mask values
   - All windows now use the winstatus() array to say whether
     they are open or not
   - Default window layout can be saved into
     Blitz2:Dbug/dbug.prefs.  Only windows 0-5 though
15/12/94
   - Trial thingy: mybuffer is loaded from SetHandler{} rather
     than the Statehandler_ as before
   - NOTE: Register window contents aren't always valid when
     program is first run.  The statehandler_ routine MUST be
     called before they become valid
14/12/94
   - RELEASE 1.0
   - Removed 'Disassemble...' menu item
   - Fixed 'Del Trace...' to make it update the variable window
   - Fixed font cockup in Object window - printing was being
```

```
     done with default font rather than topaz8.
     Thanx to Rupert Henson for the bug report
12/12/94
   - Fixed UpdateDisAsm{} to show blitz2 code properly
2/12/94
   - Removed standard nine gadgets from menu strip - keyboard
     shortcuts stay the same (but obviously minus the RAmiga)
   - Removed all occurences of CPUCls
   - Added checks in RedrawMem and HexWord{} for odd addresses
     (for 68000 compatibility) - thanx S.Le for the help with
     identifying that as a prob.
   - 'Bumped' OS version number to 39 for use of ScreenTags
     command.
24/11/94
   - Added load of new object types
   - Added option to display extra info for string tracing
       (can display length and maxlen of string,
        AddVarTrace: use output=2 for extra info)
   - STRING tracing is now sound :)
20/11/94
   - Added Del Trace... menu option
   - Added Auto EVAL - does an auto evaluation after every STEP
16/11/94
   - Added DisAssembly window.  Can be moved with cursors [+shift]
     or positioned using the EVAL function
15/11/94
   - Changed HexWord{} to improve efficiency ;-)
 6/11/94
   - Added menutitle: Variables with options to add/del traces
 5/11/94
   - Variable trace window put in - no definable traces yet
```

1.33 Communications

```
                              Communications
                              --------------

  The debugger has a communications channel that can be made use of by the
Blitz program it is currently debugging.  This channel is invoked in one of
two ways, depending on the situation:

  AlibJsr $d509 - if the code is inside of a Blitz library

  TokeJsr $d509   - if the code is inside a Blitz program

*** NOTE *** Multitasking must be enabled before you call this channel - do
not call it in Blitz mode.

This input channel expects register a0 to point to a data array, laid out
in the following format:

  +0    Command number
  +1    Data byte 1
  +2    Data byte 2
  +3    pad
```

```
  +4    Data longword 1
  +8...255  String area

  Valid commands at the moment, and the parameters they expect are:

addtrace: Add a variable trace

    Command number =255
    Data byte 1    =variable type (debugger format)
    Data byte 2    =output preferences
    Data longword 1=address of variable
    String area    =string to display in variable window

   NOTE: This command could be used to trace internal variables inside
libraries.  E.g. setting up a trace on an important 'dc.l'ed variable.


deltrace: Delete a variable trace

    Command number =254
    String area    =name of trace to kill


varwindow:  Open the variable trace window

    Command number =253


diswindow:  Open the disassembly window at an address

    Command number =252
    Data longword 1=address to open at


copperlist: Open the copper disassembly window at an address
    Command number =251
    Data longword 1=address to open at


proccontrol:  Toggle procedure control
    Command number =250
    Data byte 1    = -1 for ON, 0 for OFF (OPTIONAL
    Data byte 2    =Mode select:

        -1=Data byte 1 holds new mode
         0=Toggle current mode


memwindowmove
    Command number =249
    Data longword 1=address to move to


  It may in the future be possible for the user to add their own
functions to this channel by implemation of external debug modules for the
debugger
```

1.34 Leading Edge Software

```
                          Leading Edge Software
                          ---------------------

  This program is part of the Leading Edge Software Development Suite of
programs.  Other programs within this family are:

    ShapeZ II - shape grabber/editor/manipulator
    LES Map Editor  - block based map editor

Leading Edge Software are:

      Steven Matty
      Stephen McNamara
      Nigel Hughes
      Steven Innell
      Mike Richards
      Steven Green

Send any mail regarding this program to:

      Stephen McNamara
      17 Mayles Road
      Southsea
      Hants
      PO4 8NP
      England

Email to the blitz-list may still filter down to me if anyone wants to
contact me by email.  My email address is no longer active - so mail to it
will either bounce back or get no reply.
```

1.35 Thanx go to the following

```
                          Thanx go to the following
                          -------------------------

                      OS2.0/68000 testing and fixing:
                                Son H. Le

                      Suppling the disassembler code:
                      Simon Armstrong (Acid software)

                            Help and criticism:
                               Steven Matty
                               Steven Innell
                    And loads of chappies on the Blitz-list

                       For feedback and bug reports:
                                Son H. Le
                              Rupert Henson
                         And the others I've forgotten :)
```

Quick hello's to:
Martin 'it doesn't work with StarWoids' Kift
Jurgen 'AmigaGuide' Valks
Mark 'Virtual Worlds' Tiffany
Nigel 'Cascade' Hughes
And everyone on #amiga/Blitz-list