Exploring the Toolbox Interactively
with GSBug and Nifty List

by Dave Lyons, 15-Jul-91

An excellent way to get comfortable with the GS Toolbox is exploring it interactively. For example, when you have a question about how a particular toolbox call behaves or when it is getting called, you can usually do a quick experiment to find out.

Techniques

- Use GSBug and Nifty List together. GSBug is great for taking control of the system when it executes particular toolbox calls or GS/OS calls, and Nifty List is great for examining the state of the system and making any toolbox calls you want "on the fly".

- Explore the online help: type "?" at the GSBug command line and at all the sub-screens. Type "?" in Nifty List, and get help for individual commands ("?_" for making toolbox calls, "?\" for a list of all the "backslash" commands, "?\files", etc)

- Use Nifty List as a toolbox quick-reference for tool sets, functions within tool sets, and the parameters needed for each function.

- Experiment with making toolbox calls from the Nifty List command line; watch the results on the Super Hires screen.

- Intercept tool calls being made by applications or other parts of the system. Examine their results. Modify the parameters before making the call, or create an "error" condition after the call.

Exercises

These exercises demonstrate a moderate variety of GSBug and Nifty List features. Feel free to experiment...these are just ideas to get you started.

(Some of the exercisers assume your GS is running the Finder.)

Entering and exiting Nifty List
Apple-Ctrl-Esc gets you to the CDA menu; move to Nifty List and hit Return. When you want to leave Nifty List, Esc takes you back to the CDA menu.

Entering and exiting GSBug
Apple-Option-Ctrl-Esc gets you to GSBug. "r Return" (for Resume) puts you back where you were. While in GSBug, you can usually use Apple-Ctrl-Esc to get to CDAs, including Nifty List. (In up-to-date GSBugs, the shortcut "n Return" also takes you to Nifty List.)

Step through code
From GSBug, type "s Return" and hit Space a few times. Each time you hit Space, GSBug executes a single instruction, or a single toolbox call or GS/OS call. You can watch the register contents change at the top of the screen. — Hit Esc to return to the command line, then "r Return" to leave GSBug.

Set a "Tool Break" or "OS Break"
From GSBug, type "SetTBrk _NewWindow" and "TBRkIn" (to enable your toolbreak). Resume and open a new window from the Finder. You'll drop into GSBug when the Finder calls NewWindow (it hasn't executed yet). Type "s Return". Type "s" to see the SuperHires screen. Hit Space to execute the NewWindow (the empty window appears). Type "t" to return to the Text screen. "Esc r Return" will let the Finder continue full-speed.

OSBreaks work similarly. Try "SetOSBrk _OpenGS" and "OSBrkIn", for example. (When you don't want your breaks any more, you'll want TBrkOut and OSBrkOut.)

You may want to try tool-breaking on TaskMaster. The Finder calls TaskMaster in its main loop. Lots of stuff can happen inside a TaskMaster call (windows are drawn, moved, resized; menus are tracked). — You may find it helpful after breaking on TaskMaster to press and hold the mouse somewhere (like on the menu bar), hit Space to execute TaskMaster, and do something with the mouse. TaskMaster returns control to GSBug only when, say, you finally choose a menu item (or release the button without choosing one). (After you're done, hit T to see the text screen.)

What Tool Sets are being used? What functions do they contain?
Enter Nifty List and type "v Return". All loaded tool sets are listed (tool number, version, name, and other information).

To see all the functions in a particular tool set, type the tool set number followed by "T". For example, "2t" for all the Memory Manager functions. — Most tool sets have more functions than fit on one screen, so use Space to freeze the screen and to advance by one line. Use Return to resume scrolling, and Esc or Apple-period to return to the command line.

You can also get information on a single tool call, if you know its function number or part of its name. For example, "902t" shows just information on NewHandle. So does `"NewH` (type the quote this time!).

Try `"Paint` to see all the toolbox functions containing "Paint" (most are in QuickDraw, but one is in Text Edit). — What is the difference between the "Paint" calls and the similar "Fill" calls? (You may need to refer to Toolbox Reference, Volume 2; but with Nifty List you can quickly confirm the apparent pattern.)

Make Some Toolbox Calls
Still in Nifty List, type _TotalMem. Nifty List makes the the call for you and displays the result. (For a more verbose display of the result, put a ` in front of it: `_TotalMem.) More easy ones: _SysBeep _GrafOn, _GrafOff, _QDStatus, _QDVersion, _Random.

How about a call that needs some input parameter? Integer Math? Type `"integer` to find that it's tool set number B. Type "bt" to see what calls are there. — Multiply probably scrolled off...type "90bt" to see its parameters. It needs two inputs, so type _multiply(3,3) to get 9.

Int2Dec converts an integer into a decimal string. It needs a place to put the result, and the memory from $00/0300 to $00/03BF is a good place to play with (good for people, not for programs). — Try `"int2dec to see the parameters, then (say)` _Int2Dec(100,300,10,0). Then type 300.30f;h to see what it put in memory at location 300.

(If you want other scratch space, you can always call NewHandle manually, or you can use the \getdp or \getmem commands.)

_GetNewID(1000) allocates a new memory ID in the $10xx range and gives you the result. What if you try something the toolbox considers "illegal", like asking for a memory ID in the "toolbox" range? Try it: _GetNewID(4000)

Sometimes it's handy to use the result of one toolbox call as a parameter to another one. No problem!

What handles are allocated in memory, and who owns them?
Type 0i to get a long(!) list of handles, addresses, sizes, owners, and owner names. The zero means "everybody"—you can cut down the list by typing a more limiting "user ID". Try ?i for help with the i command.

If you just want to see what memory IDs are who, try \ids.

Try a000i to see all handles owned by "init" files in your system; 5000i to see Desk Accessory handles ("?i" for a list of the memory ID ranges).

Disassemble some assembly code
Try FF1800L to see a disassembly with a few toolbox calls in it.

How about E1/0L to see a lot of system Jump vectors and where they jump to? Follow one and see what it does, or who it's jumping into (the "w" command is helpful). There's more than just one screenful of these vectors (type L again).

Who is using the Resource Manager?
Type \res to find out the IDs of all the things that have started the Resource Manager (and the addresses of their Resource Converters, if any). You'll probably find 401E (the Resource Manager itself), 1001 or 1002 (the Finder), and possibly more, especially if you have an NDA like the Control Panel open.

Keep exploring....
See if you know how to use all the features listed on the Quick Reference pages.

See if you can use all the "backslash" commands listed by ?\

See if you can use all the toolbox calls!

Extensions to Apple IIGS System 6.0 Finder

Finder Says codes

(information sent by the Finder, perhaps because user has done something)

`finderSaysHello ($0100)`

The Finder sends `finderSaysHello` late in its startup process, every time the Finder is launched.

`finderSaysGoodbye ($0101)`

The Finder sends `finderSaysGoodbye` early in its shutdown process to inform extensions that the Finder is going away (for whatever reason).

`finderSaysSelectionChanged ($0102)`

The Finder sends this whenever the set of selected icons may have changed. On receiving this notification, an extension can make the `tellFinderGetSelectedIcons` call to see what icons are now selected.

`finderSaysBeforeOpen ($0104)`

Gives your code the chance to handle opening a document.

`finderSaysOpenFailed ($0105)`

When the user opens a document icon, the Finder sends `finderSaysBeforeOpen`. If the request does not get handled, the Finder tries to find an appropriate application to launch for the document. If that doesn't work, the Finder sends `finderSaysOpenFailed` to give extensions a chance to handle the request knowing that no application was found.

`finderSaysExtrasChosen ($0108)`

Notifies extensions that the user selected an item from the Extras menu. The Extras menu remains hilited until the request processing is finished. If an extension puts up a modal dialog in response to `finderSaysExtrasChosen`, the Extras menu title remains hilited the whole time, as it should. In this case, the extension's menu item should end with an ellipsis (for example, "Encrypt Files...").

`systemSaysUnknownDiskInserted ($0002)`

Notifies extensions that a disk of an unknown file system was inserted (the Volume call returned error $52, unknown volume type). An extension may wish to put up an alert giving the user more information about the disk.

Tell Finder codes:

(application to Finder: please do this)
(send to "Apple~Finder~" using SendRequest)

`tellFinderAreYouThere ($8001)`
Ask the Finder if it is active.

`tellFinderOpenWindow ($8002)`
Tell the Finder to open the specified window.

`tellFinderCloseWindow ($8003)`
Tell the Finder to close the specified Finder window.

`tellFinderGetSelectedIcons ($8004)`
Tell the Finder to return the selected icons.

`tellFinderSetSelectedIcons ($8005)`
Tell the Finder to select the specified icons.

`tellFinderLaunchThis ($8006)`
Tell the Finder to launch the specified application.

`tellFinderShutDown ($8007)`
Tell the Finder to shut itself down in an organized manner, and whether to "Turn Off System Power," or "Restart System," or "Quit."

`tellFinderAboutChange ($800C)`
Tell the Finder to update the icons in a certain window because the contents of that folder have been changed.

`tellFinderColorSelection ($800E)`
Tell the Finder to apply a color selection to the selected icons on the desktop or in a window.

`tellFinderAddToExtras ($800F)`
Tell the Finder to add the specified menu item to the Extras menu (adding the Extras menu to the menu bar first if it is not already there).

`tellFinderIdleHowLong ($8011)`
Tell the Finder to return how many ticks it has been idle (so, for instance, a screen blanker could determine when to blank the screen -- or another extension could determine if the system is not in use, and free to launch an unattended application).

`tellFinderGetWindowIcons ($8012)`
Tell the Finder to return all the icons associated with a window.

`tellFinderGetWindowInfo ($8013)`
Tell the Finder to return the window type and a pointer to its pathname if the window has a path.

`tellFinderRemoveFromExtras ($8014)`
Tell the Finder to remove an item from the Extras menu.

==========
Nifty List v3.3

Revision History
==========

DAL Systems
P.O. Box 875
Cupertino, CA 95015

[InterNet: dlyons@apple.com]
[America Online: Dave Lyons]
[GEnie mail: DAVE.LYONS]
[CompuServe: 72177,3233]

15-Jul-91

==========

Version 3.3:

- Backslash command names are no longer case sensitive.
- Changed in-memory handling of the data file copy so that each section can be 64K. (The NList.AppleData file is larger than 64K now.)
- Option-return is like Ctrl-T Return Ctrl-T. This is useful for viewing the super-hires screen during command execution.
- T command displays "+" next to functions that are patched by the TS2/TS3 patches, "*" next to otherwise-patched functions, and "?" next to Weirdly-patched functions.
- Added nlClassifyAddr service to map from an input address into a code and character indicating ROM, system patch, other patch, invalid address. (Used by T, for example.)
- ;s stack dump of the GS/OS stack now continues with the caller's stack after dumping all the way up to $BFFF.

Goodies 1.5d2:

- Added \res command to dump Resource Manager globals. (Shows all current Resource Manager clients and their installed resource converter routines.)
- \shrsave and \shrload always use color table information from bank $E1, even when the main image is in bank 1.

Big Brother 0.8d1:

- 1\spy enables toolbox-call watching. 2\spy enables it with sound (a speaker click on every call). 0\spy turns it off.
- Spy is only partly done, but it's sometimes useful anyway.
- When spy is on, the system goes very slowly, and messages such as the following appear:

```
** Caution: function called while toolset inactive or missing **
** Caution: no such function in tool set **
** Caution: tool called with 8-bit registers **
** WARNING: tool call with Decimal mode set! **
** Caution: xxxBootInit and xxxReset are reserved for the system **
** Caution: functions $07xx and $08xx are reserved for future use **
** Caution: caller's memory is movable or purgeable **
** Caution: caller's memory ID is not valid **
** Caution: caller address is not allocated **
** Caution: caller's stack is not allocated **
** Caution: caller's stack is movable or purgeable **
** WARNING: At TLShutDown, tool $xx is still active **
```

The following messages appear only for DrawString calls, but these things and many more will be checked on most toolbox calls when it's finally done:

```
** Detected pointer to unallocated memory **
** Detected pointer to moveable or purgeable memory **
** Detected pointer with extra bits set **
** Detected illegal NIL pointer **
** Detected pointer to invalid area **
```

Version 3.2:

- Added fast text I/O routines. Output to the screen is around 3.5 times faster now.

- A command line that is exactly the same as the previous one no longer gets added to the command history. (For example, if you do a lot of separate L commands in a row you don't have to scroll through them all to find your previous commands in the history.)

- Anywhere that you can specify a range of addresses with abc.def you can now also specify a starting address and length, by separating them with a comma. For example, 3/100,40;h dumps 40 bytes in hex starting at $03/0100 (exactly like 3/100.13f;h).

- When the "_" command returns a result of 2 or 4 bytes, it stores makes that result the current address, so commands like the following are possible (you no longer have to retype the result):

```
_LoadResource(800c,07ff0001) h
_FrontWindow;h
_GetCodeResConverter L
```

- The R command now explicitly says which resource application's search path it is displaying. 0r displays the current search path.

- ~s tries to display the name of any scrap types present. It knows about text, picture, sound, Text Edit style, icon data, mask, and color table.

- Added HyperCard IIgs callback-vector support in the quote command, the List command. Two new constructs in expressions: "#" followed by the name of a HyperCard IIgs callback evaluates to the function code, and "_" lets you execute a callback. Examples:

```
`#HC:SendCardMessage
(result = 1)

_HC:SendCardMessage(300)
(executes the callback if HyperCard IIgs is active)
```

- =\ now pauses after each module's help message, in case you have enough command modules to make =\ scroll the screen.

- Sped up the "i" command by reducing the number of times it has to call the Loader to find the pathname that goes with an ID.

- Implemented the nlDisasml service for command modules (see the Writing.Modules file).

- Mainly for the convenience of Nifty List users within Apple, Nifty List now attempts to load NList.AppleData instead of NList.Data. If NList.AppleData is not present, it loads NList.Data just like before.

NiftyList Notes

Help

?	main commands
?\	commands in available command modules
?<command>	description of <command>

ASCII for string

`"<string>	displays length, then ASCII as it would be put into memory

Command line

<Ctrl-X>	clear
<clear>	clear
<up-arrow>	history

Controls

<ctrlHandle>;c	dump control list starting with specified control
0/0;c	dump all of front window's controls
<window>;c	dump specified window's controls

Decimal/Hex conversion

`<hex>
`#<decimal>

Disassemble (see Memory dump)

<location>L	(List memory)
<location>.<endLocation>L	
<location>,<range>L	
M	toggles accum/memory between 8/16
X	toggles index registers between 8/16
<Ctrl-N>	sets 16-bit operations (Native mode m and x)
<Ctrl-E>	sets 8-bit operations (Emulation mode m and x)

Errors

<error#>\err	find an OS or tool error

Files

\files	show open files

Find in memory

<id/handle/range>\find
0\find [22 bb 0 e1] finds JSLs (opcode $22) to $E100xx ($BB is a wildcard (!))

Handle, dereference

<handle>^	dereference
<handle>^L	list from dereferenced handle

Handle of application memory

\ids	active memory manager IDs and pathnames
<memID>i	get handles of memory assigned to memID
<flags>.<memID>i	just blocks with flags equal to <flags>
c018.1000i	locked/fixed/can't cross bank/no spec mem: typical for current code
1000I	all applications
100xI	your application

Handle of memory block that includes a specified address

<address>W

History

<up-arrow>	retraces history of commands
<address>	uses History as a scratchpad (remember address)
<cmd-line><Ctrl-S>	cancels command line, but adds it to History for use later
!<comment>	exclamation point adds comment to History

Indirect address

<address>@	returns 2-byte value at address (0/36@L)
<address>^	returns 3-byte value at address (E1/1^L)

Memory available

S	system status

Memory dump (see Disassemble)

<location>;H	in hex
<location>;A	in ascii
<location>;C	CDA header (name & entry points) (# is Action entry point)
<location>;N	NDA header (name & entry points) (# is Action entry point)
<location>;R	interpret next 4 words as Rect coords (left,top) (right,bottom)

Memory dump using templates

\loadtemp	loads *:System:System.Setup:GSBug.Templates
\loadtemp "*filename*"	loads named template file
\tempinfo	pathname, memory info on loaded template file
\temp "Templates"	master template list
\temp "*category*"	subcategories in the named category
<address>/temp "*templatename*"	view memory using template
\unloadtemp	release memory used by template

Memory Manager IDs

\ids	active memory manager IDs and pathnames

Memory, store to

<location>:<value/expression> <next value/expression> <next>

Menu Bar

<menubar handle>;m	displays menu bar info
0/0;m	current menu bar info

Message Center contents

~m

Monitor, visit

*<return>

<ctrl>Y *or* Q <return> return to NiftyList

Names

<location/memID/handle>\names

0\names	all names in allocated memory except from language card memory
\ids	active memory manager IDs and pathnames
<memID>\names	names in this id

(to assemble labels into object code:

(include M16.debug	
(debugSymbols equ 1	
(label name	expands to brl (around label) / signature word / labelLen / label
(label procName	as substitute for "Proc"

Negation

`-<hex>	max 4 bytes
`-#<decimal>	

OS Prefixes

S	system status

Pipeline

#	pipelines result of T, I, H, W, ;c, ;n cmds to following cmd (101T#L)

Port

~p	list of current Window Mgr and Menu Mgr ports
<address>;p	dumps port info
0/0;p	dumps current (not necessarily front) port info

Print

<Apple-H>	dump screen to printer in slot 1
<Apple-space>	Linefeeds your printer
<Apple-return>	Formfeeds your printer

Quit

<ESC>

Q

Resources

<memID>R	all specified open resource files, with file IDs and GS/OS ref #s
0R	current application's resource files
<resourceNum>\rtype	name of resource type
0\rtype	name of all resource types
<resourceID>\ri	handles of specified resource ID

Screen

<Ctrl-T>	toggle Super Hi Res screen
\SHRsave "*pathname*"	save Super Hi Res screen

Stack dump

<stack location>;S

String location

<id/handle/range>\Find "string

Tool call

_<toolcallname(param,...)>	execute toolcall

_NewHandle(_multiply(10,#5),_MMStartUp,C000,0)

"<toolcallname>	tool call's parameters
"<partialtoolcallname>	all tool calls containing <partialtoolcallname>
<toolcallnum>T	name, current loc, params, toolset it's in (0902T)
<toolsetnum>T	for all toolcalls in toolset

Toolset versions, start status

V	toolset#, +=started, version, calls in RAM/ROM, WAP, name

Windows

~w	list front to back
<address>;w	dumps window info
0/0;w	dumps front window info

Work Area

	$300-3CF generally available in 16-bit
<# of pages>\getdp	gets dp space (e.g., 3\getdp)
<# of bytes>\getmem	gets memory space (e.g., 1234\getmem)
5500i	show all workspace areas requested (all memory ids start w $5500)

GSBug 1.5 Read.Me 15-Jul-91

Apple Confidential

GSBug Version 1.5

July 15, 1991
Current Version 1.5b17

Version 1.5b17 (DAL)

Now supports inline imbedded procedure names (as generated by the name and procname macros in M16.Debug, for example). These show up in disassembly and as the operands of JSRs and JSLs.

OSBreaks trigger regardless of call class now. For example, you can SetOSBrk for either Open or OpenGS, and either one will cause a break.

When GSBug notices a toolbox call being made in other than full 16-bit mode, it displays a a warning dialog (using TLTextMountVolume). Hitting ESC at that dialog cancels future warnings until you reload GSBug.

When GSBug notices a toolbox call being made with Decimal mode on, it stops cold at a BRK $F8, rather than letting things get completely baked before crashing.

Fixed one old reference to $010100 to store $C0 instead of $80. Interrupts should reliably use $100..1C0 for stack space now.

Added 5 blanks to end of 'TRACE ' message so it completely overwrites the 'SINGLE STEP' message.

Version 1.5b15 (DAL)

Option-space now works reliably to bypass a memory-protect range, including a tool call. (This is an old feature, but keyboard translation normally prevented it from working!)

In trace mode, Space and ESC now kill "awainting RTx" mode.

Fixed "n" command so it won't crash if nobody has called DebugSetHook.

Having tool breaks on calls which get made indirectly by GSBug no longer cause a crash. You can break on NewHandle now, for example.

In Breakpoint subscreen, Space maps trigger count from 0 to 1 and from nonzero to zero. Tab moves between the address and count fields (easier than hitting arrows).

In the memory-protect subscreen, Tab moves between columns.

Added $01/FC00.FFFF (OS system service calls) to the memory protect list.

Changed the default trace-window setting to center-screen.

Changed the EmulStack value from $80 to $C0 (trying to get rid of some unpredictable crashes, probably caused by AppleTalk running out of stack space). This means you can safely trace a program while the stack is in the $01C1..01FF range, and that interrupts use $0100..01C0.

GSBug now takes a whole bank, minimizing its effect on where things are located in memory relative to each other, and ensuring that tool breaks work reliably (tool calls never break if they come from the same bank the debugger is in).

Version 1.5b14 (DAL)

Versions 1.5b12 and b13 were never officially released. 1.5b14 is fine, except that I make no guarantees about the Template commands. I think they work, but you'll get a funky error message from loadtemp.

When you let a JSL execute in real time (including a tool call), the debugger temporarily changes the owner ID of its own handle to match the owner of the handle containing the code you"re debugging. This way MMStartUp returns the appropriate memory ID, instead of always returning the debugger's ID.

DebugSetHook(nil) now removes the hook.

The 1K bank 0 segment GSBug allocates now has the same ID as the debugger (was previously always $80xx).

Fixed OS breaks to work after return from ProDOS 8 (added a Notify Proc to re-trap the OS vectors).

Changed the "_" command so that if you don't type a number, it's like typing zero. For example, if you have a template called "Template" which displays an informational message, you can type "_Template" instead of "_Template 0".

DP:xxx command dumps 16 bytes from DP to the command line.

Tool call $0CFF DebugGetInfo(word):long. Word=0 returns the current value of the program counter (useful from a procedure called by the N command).

Note that real-time counted breakpoints don't work for JMP() ($6C), JMP(,X) ($7C), JML() ($DC), and JSR(,X) ($FC).

Located the misplaced CLI that was causing the X command used on a JSR to accidentally return with the Bank register set to the debugger's bank, and the Stack set to the Interrupt-time stack.

GSBug Notes

Help

?

Weird screen stuff (every other column is a column of spaces)

OFF
ON

Breakpoint

bp
<value to break on>
<tab> or <spacebar> to set 1 as the iteration to break on
<iteration to break at; e.g., 1> or "i" for conditional breakpoints
<esc>
in sets real-time breakpoints
r return to program execution

Breakpoint conditionally

bp
<value to break on>
<right arrow>
i for conditional breakpoints
<esc>
SETIF <B or W> *expression* exprssn operators: =, # (not equal), <, > (greater than or equal)
SETIF W A<#$0101 break if word in accum is less than the constant (#) $0101
SETIF B $021234=$034321 break if byte at $02/1234 is the same as byte at $03/4321
SETIF W X>Y break if word in X greater than word in Y (regs: AXYSDPB)
in sets real-time breakpoints
r return to program execution

Breakpoint if value in register is such-and-such

setif W A>#$0001 W for word value; A for accum; > for ≥; so if accum>0, then
break
bp
<value to break on>
<right arrow>
i for 'if'; puts 'IF' in iteration area
<esc>
in sets real-time breakpoints
r

Memory, display

<address> 16 bytes on the command line
<address>: flip to memory screen (21 lines, 16 bytes each) (ESC to return)
DP: direct page
DP:xxx 16 bytes from direct page:xxx to command line
<address>:: indirection: use 2 bytes at address
<address>::: indirection: use 3 bytes at address
<spacebar> next block of memory

Memory, set

<address>:value put hex value in memory starting at address

Monitor, visit

Mon NOTE: it blows K / PC / Stack / DP / B / ... Use NiftyList!
Ctrl-Y <return> return to GSBug

Nifty List shortcut

N transfers control to NiftyList

OS Breaks

SetOSBrk _OpenGS append GS for Class 1 calls!!!
OSBrkIn
ShowBrks

Registers, alter contents (case sensitive)

<register>=<value>	use A, X, Y, K, PC, K/PC, B, D, S, P (& M,Q,L)
x	toggles x bit of P
m	toggles m bit of P
e	toggles emulation mode
DPAGE	sets D to direct page allocated for user by GSBug
STACK	sets S to stack allocated for user by GSBug

Set Step-and-Trace highlighted line

Set
<up-arrow> <up-arrow> ... as many times as necessary
<esc>
CSave *:System:System.Setup:GSBug.Setup

Step

S

<space>	for each step
X	skip across a jsr or jsl
<downarrow>	skip next instruction

Tool Breaks

SetTBrk _ShutDownTools
TBrkIn
ShowBrks