The Mensch Days

This part of the document details all changes made to the debugger from version 1.3b1 to 1.5b8.

Bug Fixes:

Real Time Tool Breaks - The mechanism for using tool breaks real time has been modified, the old method never worked quite right, what used to happen was that when a tool break was to occur, the stub in the dispatch vector would simply jump into the debugger, this would cause the stack and processor not to be set as if they came from an interupt, and the next exit of the debugger would cause code execution to start at an inapropriate case, also the actual tracing of a tool break would also cause the debugger to improperly save the current registers, so that the stack/direct page and processor status might come back damaged from a tool break. The stub has been modified to now enter the debugger with a break instruction instead of jumping to the front. This seems to solve all real time tool break problems.

Error tool breaks - These also did not work well, and did not seem to be used by anyone anyway, so they have been removed to make room for OSBreaks.

Version 1.5b3:

- display code was fixed so that the '*' would properly be displayed in front of any toolbox glue that is detected.
- Fixed template data display so that if the data being displayed crosses a bank boundry, the data is properly followed into the new bank.
- Modified the IN command so that it respects trigger counts of 0 and does not insert real time breaks.

Version 1.5b5:

- Added Glue tool break support. Now toolbreaks work for regular and glue tool breaks. ErrorBreaks do not work for glue as they do not make a lot of sense.
- Made debugger work even when a DA window is open. See new features below.

• Added keyClick so that you hear whenever the system draws a key from the event queue. Also, cleaned up some comments

Version 1.5b6:
• Added new DebugStr toolcall that can be used to get more from the debugger

Version 1.5b7:
• Added support for real time conditional breaks.
• Added OSBreaks.
• Fixed the Debugger tool calls (like debugstr) so that they use the proper tool numbers (they had the toolset number in the high byte instead of the low byte) Documentation for them should also be right.

Version 1.5b8:
• Added a debugger version and status call to the debugger tools
• Enhanced the OSBreak facility
• Removed the keyclicks because Dave complained.

New Features:

DebugStr:

This feature is designed to allow developers to better know where in their program the debugger was entered. The way this feature works is that if you want to enter the debugger programatically you can now also pass a string to the debugger which will be printed on the bottom line of the screen when the debugger is entered.

The way this mechanism works is a fake tool call that the debugger now supports. Since this looks like a normal tool call it can be easily called by any high level language as well as by assembly language. This toolcall called DebugStr takes a single parameter, a pointer to a pascal string. When the tool call is made the debugger is entered as if you had put a break into your code, but the string you passed is displayed on the screen and the program counter has been bumped passed the tool call jsl. This way you can simply resume execution with two simple keystrokes (with the init version hit 'R' <return>).

Calling this new feature might look something like this

```
        PushLong #DebugStr
        ldx #$09FF
        jsl >$E10000

        ...
DebugStr   str 'You are about to do blah.'
```

from pascal the same would be achieved by doing this...

```
        DebugStr('You are about to do blah.');
```

I would recommend that assembly language users use a macro to make the toolcall and call it _DebugStr. Max pascal users could use the following to define the debugstr routine:

```
Procedure  DebugStr(theString:str255);
      INLINE $09FFA2, $E1000022;
```

This call will also work when being called via the glue vector.

NOTE: Since this call is only available when the debugger is loaded you will ALWAYS want to be sure to remove ALL calls to the debugger before releasing your program (or even using it on machines that do not have a debugger installed).

SetMileStone:
SetMileStone operates exactly the same way as debugstr except that the debugger is not actually entered. This will allow tracking of random crashes by allowing you to call the SetMileStone routine with milestones that have been met. any time the debugger is entered via a brk instruction or thru the keyboard the last string passed via the SetMileStone call will be displayed. the toolcall number for sweetstring is $0AFF, the max pascal interface might look like this:

```
Procedure  SetMileStone (theString:str255);
     INLINE $0AFFA2, $E1000022;
```

DebugVersion, DebugStatus:
These calls are added so that an application can identify the version of the debugger that is loaded and thus know what calls can be made to the debug tool. currently these calls are identical and return the same result. The status call returns non-zero for true (as opposed to $FFFF which most people might want...) other than that these calls act the same as any other toolbox status or version call, each call requires a word space on the stack for the result which is on the top of the stack when the call completes. If a debugger is loaded that does not support this feature you will get a standard tool locator error. For version 1.5b8 the version number returned is $158F as you might expect this number will change as the debugger version changes and it will always go up never down.

Glue Snypher:
Glue snypher is a routine that can recognize high level language tool calls that are made with the standard glue entry, when a call to glue is detected while disassembling an instruction the call to glue will be replaced with ***_TooName** in a manner similar to how real tool calls now work. Glue snypher is also used by the memory protect feature, if a glue call is detected while the standard toolbox memory protect range is on, the glue call will be treated as if it were a JSL >$e10000. All glue entries MUST be a JSL to one of 3 standard types of glue that are shown below:

```
ToolGlueType1       LDX #$ToolNum
                    JSL >$E10004
                    ...
                    RTL

ToolGlueType2       LDX #$ToolNum
                    JML CommonCall
                    ...
CommonCall          JSL >$E10004
                    ...
                    RTL

ToolGlueType3       LDX #$ToolNum
                    JMP CommonCall
```

If any other types of glue calls are used, they will not be properly detected.

Glue Breaks:
In addition to glue snypher further support was added for high level languages by extending the tool break support to the glue vector ($E10004). Since error conditions are treated differently with glue (The second RTL ain't mine no more...) it seems unreasonable to also support error breaks via the glue vector.

OSBreaks:
OSBreaks work very much the same way that tool breaks work with three exceptions. First, instead of breaking on a tool call they will break on a call to the OS (gee....). Next, you can NOT specify an OS break by name, only by number. Lastly, they are not supported in trace mode, only in real time mode (maybe in a future version...). To use OSBreaks you simply type setOSBrk #xxxx where xxxx is the number of the OS routine that you want to break on. This number is matched exactly when an OS call is made, so if you do not know if the target call is class 0 or class 1 you must set the break for both calls. OSBreaks are supported via both vectors, inline and stackbased. The following is a list of commands that operate on OS breaks and what they do.

SetOSBrk - adds a number to the OS break list
ClrOSBrk - Removes a given number from the OS break list
ClrAllOSBrk - Removes all numbers from the OS break list
OSBrkIn - enables real time scanning for OS breaks

OSBrkOut - disables real time scanning of OS breaks
ShowBrks - lists all tool and OS breaks currently set.

DA Debugging support:
In the past it has always been difficult to debug certain new desk accs because they accept keystrokes, and keep the debugger from recieving them. I have added a patching mechanism to the system event call that is installed when the debugger is installed that fixes this problem. The way this feature now works is that if the caps lock key is down no keyboard events get passed to system event (and then to any open DA's) and the debugger will always get them. a side effect of this is that when the debugger screen is not active applications will get the same keystroke events if the caps lock key is down. As a gentle reminder of this feature, when installed, it will make your GS speaker click whenever a key is processed by getNextEvent.

New Template types:

We have added support for a few more data types in templates, these are mainly for use with GS/OS strings, but can also be used for any word length string. they are:

InputStr: This type will read the next word of data and treat it as a length word for string data that follows. The entire contents of the string will be displayed.

OutputStr: Similar to InputStr except that the first word is a buffer length and the second word is the string length. After the string is displayed the debugger will skip to the end of the buffer to find the next byte of data to process.

RealTime Conditional Breakpoints:

The debugger now supports conditional real time breakpoints. In the past when a breakpoint was entered in the breakpoint list and you used the IN feature to make them work real time, the trigger count that you entered was ignored. Now, If you the trigger count is supported the same way it is supported in trace mode, that is, if you insert a real time breakpoint with a count higher than 1 it will break on the Nth execution of the opcode at that location. NOTE: Currently