

SMX[®] Embedded Debugger Comparison Chart¹

Feature	VisualProbe v2.1.343	SoftScope v5.31	Paradigm C++ IDE v5.00 (real mode)	Microsoft Developer Studio v6 ²	Turbo Debugger Remote (real mode) ³
Documentation ⁹	Excellent. Complete, detailed manual in printed and PDF form, plus online help.	Excellent. Complete, detailed manual in printed form, plus online help.	Good. Full chapter in User's Guide, plus online help.	Fair. Online help only. Minimal information; a single short section of Visual C++ "manual".	Excellent. Complete, detailed manual in printed form.
User Interface	Excellent; true Windows interface. All windows may be positioned and resized. Commands are accessible from buttons on toolbar or via menus. Some advanced commands are only supported via command line, though. Does not have context-sensitive menus.	Adequate. A true Windows interface but can be a bit clumsy to use. Buttons on toolbar are context-sensitive to window selected, but behavior is sometimes buggy. Commands accessible from menus too. Does not have context-sensitive menus.	Very Good. A true Windows interface with context sensitive menus. The remote connection and downloading capability are well done. The CPU window and lack of separate windows for displaying some things are liabilities. Paradigm has improved over Borland in some areas. More work to do.	Very Good; true Windows interface that mostly works well, but it is a bit wasteful of space so a larger monitor is required. Also, docking windows can be frustrating. Context-sensitive menus are convenient. Judging from the documentation, this debugger has little functionality. See note 5 for more details.	Good. Excellent for a DOS text mode interface but this style is archaic. Big help to use a mouse. Most operations are accessible from menus that appear on right-click. Otherwise, some operations require knowledge of keyboard shortcuts. Has context-sensitive menus.
IDE Features	None	None	Project management, quick edit/debug cycles, integration with version control and other utilities.	Project management, quick edit/debug cycles, integration with version control and other utilities.	None
Memory of Layouts, and Settings Between Debug Sessions	Excellent. Layout for project is retained across runs. Can also define and save any number of screen layouts that can be loaded at any time.	No. Claims to, but doesn't work.	Very Good. Remembers everything about screen layout, breakpoints, watches, etc. for each project. However additional screen layouts cannot be created.	Good. Remembers everything about screen layout, breakpoints, watches, etc. However, layouts are not unique per project. Supposedly window layouts can be saved, but the View Define Window Layout command documented in Help does not seem to exist.	Very good. Can save virtually everything, even remembers breakpoints if load same exe file.

Feature	VisualProbe v2.1.343	SoftScope v5.31	Paradigm C++ IDE v5.00 (real mode)	Microsoft Developer Studio v6 ²	Turbo Debugger Remote (real mode) ³
History of keyboard inputs ¹²	Yes	No	Yes	N/A See note 12.	Yes
Macros	Attach to user buttons in tool bar, making them very convenient to use.	Available but not easy to use or create.	A lot of emphasis was placed on this capability. A full scripting language is available for customizing the IDE and for running sequences of commands. A complete manual is provided for this feature.	Supports VBScript macros, but these may be difficult or not very useful for automating debug tasks. See note 4.	Keystroke macros easy to use and create.
Multitasking, allowing other programs to run on host while debugging	Excellent	Excellent	Excellent	Excellent	Poor. Tends to want to dominate processor. Seems to work better with NT than 95, but may have problems with mouse.
Command line interface accessibility	Excellent. Can use separate window which is displayed at all times and has command history etc.	Adequate. Pop up dialog allows entry of command.	Very Good. Appears as a single "Run" line above status bar. Has command history, but entry line disappears after each command is entered.	None, apparently.	No
Embedded Target Support	Pre-built debug monitors for roughly 10 common eval boards	No pre-built monitors for particular boards, but has nice configuration utility to help generate monitor for user's target board.	Wizard makes it easy to create debug monitor. Can choose from several particular 3 rd party boards or select processor, UART, etc, and it generates a set of files to start from.	No. Target must be a PC running Windows.	Only supports DOS real mode, unless using Paradigm DEBUG version, which supplies monitors for various targets.
Establishing Serial Link: specifying comm. parameters	Excellent. Searches ports each time, allows user input to manually select if fails to link.	Not dynamically. Must edit ini file and restart.	Excellent. If there's a connection problem, a dialog appears to allow trying another port or speed.	Not tested since target must be PC running Windows.	Not dynamically. Pass as parameters on command line, restart. Easy to start remote debugging.

Feature	VisualProbe v2.1.343	SoftScope v5.31	Paradigm C++ IDE v5.00 (real mode)	Microsoft Developer Studio v6 ²	Turbo Debugger Remote (real mode) ³
Download code from host	Yes	Yes, but <i>smx/pmEasy</i> don't support.	Yes	Yes; not tested since target must be PC running Windows.	Yes.
C++ Debugging	Fair. Available for some compilers in limited form.	Fair. Available for some compilers in limited form.	Excellent since Paradigm has full control over compiler and it is in sync with the debugger.	Excellent since Microsoft has full control over compiler and it is in sync with the debugger.	Excellent. Limited compilers supported.
C Source-level Debugging	Very Good. Nice display and element coloring, but does not show which lines are "in". May lag a version in compiler support due to debug format changes.	Good. Nice display but no element coloring. Does not show which lines are "in"	Excellent. Nice display and element coloring. Shows which lines are "in" using dots in the margin.	Very Good. Nice display and element coloring, but does not show which lines are "in".	Good. Indicates which lines are in. No element coloring, etc. Guaranteed support of latest debug symbolics (since they control them!).
Mixed Source/Assembly code display and opcode display ¹⁰	Excellent. Displays source line followed by assembly lines. Shows opcodes if desired.	Excellent. Displays source line followed by assembly lines. Shows opcodes if desired.	Poor. Display is in separate "CPU" window that has other panes that make it clumsy to use. Does not show C statements nicely, and other panes reduce viewable area. Shows opcodes only if window is stretched large enough.	Excellent. Displays source line followed by assembly lines. Shows opcodes if desired.	Poor. Display is in separate "CPU" window that has other panes that make it clumsy to use. Does not show C statements nicely, and other panes reduce viewable area.
Single Stepping without source code for a module.	Very Good. Shows names of public routines and variables in disassembly. Only works for references to start address of routines and variables; would be nice if it showed symbol+nn when the reference is not on the boundary, which would be good when code indexes into an array or jumps to an offset in a routine.	Fair. Makes no use of symbol table; no symbolic names appear, just addresses.	Good. Shows names of public routines in disassembly, but not variables. This makes it harder to figure out what the code is doing; tedious to consult map for each. Necessary to switch to CPU view to step into such a routine or debugger will just step over it.	Excellent. Shows names of public routines and variables in disassembly. Also shows symbol+nn when the reference is not on the boundary, which is good when code indexes into an array or jumps to an offset in a routine.	Excellent. Uses all symbol table info very effectively.

Feature	VisualProbe v2.1.343	SoftScope v5.31	Paradigm C++ IDE v5.00 (real mode)	Microsoft Developer Studio v6 ²	Turbo Debugger Remote (real mode) ³
View At and Run To comands ¹³	Both are supported and easy to use. View At is a button at the bottom of the code window for convenience.	Both are supported and easy to use.	Run To doesn't work, but View At (Locate symbol) does. Can use View At and then set a breakpoint. (Not convenient, though.) See note 13.	Neither! See note 13.	Both are supported. View At (Goto) is only available by right-clicking mouse.
Breakpoints: Ease of setting and clearing	Very Good. Can go to line and press button or can manually enter info via dialog. Pressing the button again removes it. Breakpoints can be easily edited if desired.	Very Good. Can go to line and press button or can manually enter info via dialog. Pressing the button again removes it. Breakpoints can be easily edited if desired.	Excellent. Clicking in the margin beside the desired line sets the breakpoint. Clicking again disables it. Clicking a third time removes it. Breakpoints can be easily edited if desired.	Very Good. Can go to line and press button or can manually enter info via dialog. Pressing the button again removes it. Breakpoints can be easily edited if desired.	Very Good. Can be easily toggled on/off with F2 key. Can edit breakpoints but not as convenient as other debuggers, since breakpoint list is separate window.
Complex breakpoints: break on data change etc.	Excellent. Nice dialog box that lists all breakpoints and allows easily editing. Offers many options, including control of pass count (when reset) and action to take when break is reached. With smxAware, can do task-specific breakpoints. See note 7.	Good. Dialog box lists all breakpoints and allows easily editing. Must have manual to know what to put in fields; no canned, easy options to select from. Can run macro when breakpoint is reached.	Excellent. Dialog box lists all breakpoints and right-clicking one allows easily editing it. Offers several options, including control of pass count (when reset) and action to take when break is reached. Can define groups of breakpoints. See note 7.	Very Good. Good dialog box that lists all breakpoints and allows easily editing. Offers some options, such as break if expression true and pass count. See note 7.	Adequate. Limited options.
Call Stack: ability to trace call nesting and parameters	Excellent. Shows the parameters by name and their values as well as the module and line number that made the call, in a nice format.	Does not seem to work. Usually gives no information when debugging <i>smx</i> apps.	Excellent. Shows the parameters by name and their values as well as the module name that made the call, in a nice format.	Very Good. Shows parameter values but does not show their names, only the types. Not as nice as VisualProbe's display.	Excellent. Always knows what calls have been made no matter how convoluted.
Trace previous execution	Yes, but from command line only and doesn't work very well.	Limited capability. Can do trace on single-stepped commands.	Yes, with emulator or JTAG. (Source code trace!) Unavailable with software-only debugging (PDREMOTE monitor).	No	No

Feature	VisualProbe v2.1.343	SoftScope v5.31	Paradigm C++ IDE v5.00 (real mode)	Microsoft Developer Studio v6 ²	Turbo Debugger Remote (real mode) ³
Debug hardware isr's ⁸	Yes	Yes	Yes, with emulator or JTAG. No, with PDREMOTE.	Probably not since Win32 applications can't hook hardware isr's.	Yes
Step into software interrupt (INT call) ⁸	No	No	Yes, from the disassembly view in the CPU window, but not from assembly or C source.	Probably, but can't hook interrupts in Win32.	Yes
Basic Register Display	Excellent. Separate window shows all registers and flags in readable format and highlights changed values in color. Has both vertical and horizontal layouts.	Good. Separate window displays registers in readable format, but changed values are not highlighted. Horizontal layout only.	Very Good. Separate window displays registers in readable format and highlights changed values in color. Vertical layout only.	Very Good. Separate window shows all registers and flags in readable format and highlights changed values in color. Lines wrap for horizontal or vertical display. Display could be better; see note 6.	Excellent. Highlights changed values in special window.
FPU Register and Stack Display	Yes	No	No	Yes	Yes
Special / Protected Mode Register Display	Excellent. Able to watch changes easily.	Good.	N/A	No. Big disadvantage for debugging embedded PM startup code, etc.	N/A
IDT, LDT, GDT Display	Excellent. Full information about each descriptor is shown. Nicely formatted.	Good. Reasonably well formatted.	N/A	No. Big disadvantage for debugging embedded PM startup code, checking interrupt vectors, etc.	N/A
Memory Dump: Ease of use and readability ¹¹	Good. Can easily vary the width to 8, 16, 32 bit items. Can specify the number of elements to display per line. Also highlights changed values in the range currently displayed in the window. Does not allow display of word or dword at odd address; used to but not anymore. See note 11.	Fair. Can vary the width to 8, 16, 32 bit items. Only hex display. Can only select 8 bytes per line or 16. Need to press Shift button to display word and dword at odd address. Some setting changes cause it to move to a different address, requiring re-typing address.	Good. Can easily vary the width to 8, 16, 32 bit items and select from a few data types besides hex display. Window size controls whether 8 bytes or 16 displayed per line. Does not highlight changed values in the current view. Allows display at odd address.	Very Good. Can easily vary the width to 8, 16, 32 bit items. Can update the number of items per line by adjusting pane width. Also highlights changed values in the range currently displayed in the window. However, can only display in hex, not other data types. Allows display at odd address.	Very Good.

Feature	VisualProbe v2.1.343	SoftScope v5.31	Paradigm C++ IDE v5.00 (real mode)	Microsoft Developer Studio v6 ²	Turbo Debugger Remote (real mode) ³
Stack Display	Very Good. Follows the stack pointer so it is always at the top or bottom of the window (direction controlled by button on window). Also marks ebp. Has button to quickly return to ss:esp if user has scrolled to inspect stack.	Not Available. Must do memory dump to see stack. Very inconvenient to watch the stack since memory dump doesn't follow ss:esp.	Fair. Stack display is one of the panes in the CPU window so it is clumsy to use and requires having the whole CPU window on the screen which blocks the code window. A separate window would be much more convenient.	Not Available. Must do memory dump to see stack. Very inconvenient to watch the stack since memory dump doesn't follow ss:esp.	Adequate. Shows BP, SP. Can't resize window independent of CPU window. Only a few lines visible at a time
View local variables	Very Good. Special window automatically shows all locals. Works like watch window; structures can be expanded, etc.	No special window. Must display in the watch window as any other variable; not automatic.	Good. Special window. Like watch window; can't expand structures, etc.	Excellent. Special window. Has Auto tab that shows variables near the current instruction, which is updated on each step so most relevant variables are at the top of the list.	Fair. No special window.
Data Display in Watch Window and Locals Window	Very Good. Expands/collapses structures nicely. Display has a nice format. Sometimes it doesn't know the type of a data structure and can't display it well, but this is rare. Can not typecast variables.	Good. Expands/collapses structures and arrays. Display not very nicely formatted and when expanding structures, fully expands instead of letting user expand items of interest. Can not typecast variables.	Good. Watch window shows values but cannot expand structures in watch window. Can use inspector windows for this, but only 1 variable per window, so can end up with many windows, cluttering display and hard to find right one when so many stacked.	Very Good. Expands/collapses structures and arrays nicely. Can only typecast variables to basic types, not user-defined types. Unfortunate when displaying data pointed to by void * in certain format, for example.	Very good. Can enter C-like expressions, including typecasts.
I/O accessibility and display	Yes. Separate window for display.	Adequate. Via command line only.	Yes, with emulator or JTAG. No, with PDREMOTE.	No	No
smx-aware debugging	Yes. <i>smxAware</i> DLL.	Yes. <i>smxAware</i> DLL.	Not yet; Paradigm is creating DLL interface.	No	Yes. DEBUG/ <i>smx</i> DLL for Paradigm DEBUG
Emulator Support (ICE) (front end)	Yes	Yes	Yes	No	No. Yes for Paradigm DEBUG.

Notes

1. This chart applies specifically to the use of these remote debuggers used to debug SMX real mode and protected mode applications. This information has been garnered from real life experiences debugging SMX products.
2. Developer Studio's debugger, as supplied with Microsoft Visual C++ is not an embedded debugger; it has no ability to debug code running on embedded targets. However, with our WinBase product, it is possible to use it to develop some of the application code in a Win32 environment.
3. The Turbo Debugger column applies to Paradigm Debug as well (This is the older tool that pre-dates Paradigm C++). Paradigm Debug also has *smx*-specific features and a real mode target monitor.
4. In Developer Studio, it is not clear how to define a macro to automate a debug process such as: "Go to routine(), step n times, set breakpoints at locations, Go." It has the ability to record macros as you step through the code, etc, but we were unable to change the cursor location in order to set a breakpoint at a particular line. It can record step over / step into sequences, but we're not sure you can tell it to run to particular routines. Macros can also be composed in VBScript, but who knows what incantations and voodoo you do to get it to follow sequences such as the example above. There seems to be no documentation of a command-line interface to the debugger, so this type of use may not be possible. If it is, it is presumably an advanced feature and Microsoft would rather that you learn it from a colleague than document it.
5. In Developer Studio, a fair bit of space is wasted by borders between panes and thick scroll bars and tabs, so a 17 inch monitor is a bit too small. Docking windows can be frustrating, often requiring several tries to get it to snap to the desired window and screen location. Button placement is fairly good, but the execute and breakpoint buttons should be placed with other code stepping buttons. Similarly, the Breakpoints dialog box is accessed from the Edit menu rather than the Debug menu — since this is an IDE, there are other non-debugging features, so it would be nice if all debug-related operations were under the Debug menu.
6. Register display for Developer Studio would be better if basic registers, segments, and flags were considered groups so that wrapping would work better. As it is, the registers just wrap to fit, so the first segment register may display to the right of the last basic register and similar for the flags. It would be nicer if each register group started on a new line. This is especially a problem when docking to other windows since resizing one resizes all that are docked.
7. Breakpoint control is a very important aspect of a debugger since this is what allows the user to see how the program is running and to control its execution for solving more difficult problems, so it is worth some detailed discussion: Although Developer Studio's breakpoint editing capability is fairly good, VisualProbe offers more options. It allows setting whether the pass count is reset on each Go or whether it must be manually reset; Developer Studio only supports manual reset so if you wish to run, say 100 passes, repeatedly, you need to edit the breakpoint each run — if you have a loop counter that goes up to 50000 iterations, for example, that's a lot of times you'd have to edit the pass count if you want to run 100 loops at a time to zone in on which iteration is the last to run successfully. Also VisualProbe offers many options for what it does when the breakpoint is reached: break, break and do command, do command and continue, display message and break, display message and continue, beep and break, beep and continue. Both allow setting breakpoints for locations, both as simple breakpoints or to check a condition when the location is reached. Both offer expression breakpoints that are not tied to a particular code location.

The Paradigm C++ debugger offers good control of breakpoints, like VisualProbe. The choice of actions to do at a breakpoint are a little different: log expression, evaluate expression, log message, enable group, disable group. Checkboxes allow breaking or continuing when reached and starting and/or stopping the logging. Breakpoint groups are an interesting feature. Breakpoints can be specified to be part of a group and then can be all enabled or all disabled as the result of reaching a breakpoint. Strangely, it seems you cannot manually enable or disable a group. This feature is unique to this debugger. A

pass count can be specified, but it has the same limitation as Developer Studio, in that it's a one-shot passcount. Unfortunately, it takes a bit of effort to get to the advanced breakpoint settings dialog, compared to VisualProbe. Whether this is good or bad is a matter of personal preference. This approach takes an information hiding approach, where the user is not distracted by details, yet it takes several mouse movements to get there: press breakpoint edit button, right-click breakpoint, select Edit breakpoint, press Advanced button. By contrast VisualProbe displays everything in a single dialog box, which saves a lot of mouse motion and clicking each time you want to configure a breakpoint. Paradigm's debugger does not seem to offer expression-only breakpoints that are not tied to a particular code location.

8. All debuggers allow debugging software interrupts because they are nothing more than procedure calls. However, most debuggers do not make it convenient to reach the interrupt handler. It is very convenient if the debugger lets you step into the int instruction just as if you were stepping into a function call. The alternative is to View At the isr and put a breakpoint there or do a go to cursor, both of which are much less convenient.
9. It is a definite plus to have a separate, printed manual for the debugger that you can have for easy reference while learning it. Online help can be useful, but with IDE's such as Paradigm's or Microsoft's, searches cannot be done specifically for debugger information only. This is a problem when searching for a term such as "macros" to find out if the debugger supports them, since this term is used so heavily in the documentation relating to programming that it wastes a lot of time checking each reference that comes up to see if it is relevant to the debugger.
10. A good mixed Source/Assembly display is important. The best way to do it is to do it in the code window with C/C++ statements shown as normal, but with the assembly opcodes interleaved between the source lines. It should also allow easily toggling on and off the display of opcodes. VisualProbe does this the best, although there is trouble when the disassembly for one source statement spans longer than the height of the window (this is rare). The Paradigm debugger and Turbo Debugger are poor in this regard because they use a separate CPU window that covers the code window. To make the code pane of this window large enough to be useful, its other panes cover important windows you have open on the screen. Also, the display of the source lines in this window is poor and not like they are displayed in the main code window.
11. It is important to be able to display types larger than a byte at any boundary, since the program might have a dword stored at an address ending in 1, 2, or 3. The rating for VisualProbe for this category had been Excellent but was downgraded severely because this one feature no longer works.
12. History of inputs means that when prompted to input something, such as the name of a routine to go to or a variable to watch, the debugger offers a drop-down list of previous inputs that you can select from, in order to save re-typing them. VisualProbe and the Paradigm debugger do this nicely, with previous entries sorted so that the most recently entered is at the top of the list. Soft-Scope has no memory of previous commands and we find this frustrating. Developer Studio was marked N/A because it does not have commands that need this, such as "go to routine" and "view at." The quick watch seems to be the only time it asks for input and that does have history of input. Mostly, this category doesn't apply to Developer Studio because of its interface.
13. View At and Run To are very important commands. View At is useful for setting breakpoints or just to get to the source code for a routine quickly to inspect it. Without it, one needs to remember what module a routine is in and where it is in the module. It can waste minutes, sometimes, trying to manually find a routine. That is why it is so amazing that Developer Studio does not implement this command. It's frustrating and distracting to have to remember what module to open (or grep to find it) and then to scroll to find the routine when the module is, say, 2000 lines long and contains 50 routines. Run To is essentially the simplest way to do a breakpoint since there is no need to go to the location to set the breakpoint. The absence of Run To is inconvenient but it's not as important as View At since a workaround is to View At the routine and then set a breakpoint there. Without View At, though, it is difficult to quickly set breakpoints in various routines, especially when they are scattered across different modules.

Qualitative Analysis

VisualProbe: This is the nicest debugger we've used so far, so it serves as a reference point for others in the chart. It has many features and a clean, intuitive interface. There are separate windows for each thing that can be displayed, and these can be arranged and sized as desired. The layout is saved with each project individually, and additional named layouts can be saved and switched among easily. It has very nice display of the descriptor tables and a good stack display window. It has a nicely organized toolbar, as well as buttons at the bottoms of windows for frequent operations for the particular window. For example, the bottom of the code window has a Break +/- button for quickly toggling breakpoints and an IP button to quickly move the view to the current instruction, a drop-down list for changing between pure C and mixed C and assembly display, as well as other buttons. Unlike Developer Studio there is no wondering about features VisualProbe supports because menus are simple and not very deep, plus full documentation is provided in a nicely printed (and PDF) manual so you don't have to search endlessly through online help pages to see if some feature is supported or not. It allows easily changing contents of variables, registers, memory dumps, descriptor tables or anything else by entering directly into the windows. Adding watches is very easy since they can be directly entered into the watch window without needing to go through the menu and a dialog box. The target interface works well, allowing the user to easily try different settings if it doesn't connect. Not enough good can be said about this debugger.

However, it is not perfect. Since the developer supports various compilers and does not offer a compiler of their own, there have been problems due to the compiler vendor making changes to debug symbolics. Sometimes the debugger shows a variable "untyped" so it is unable to display it in its proper format. We use this debugger extensively in debugging our own code, and in practice, this has not been much of an issue. C++ support has been an issue in past versions, but it seems that the problems have been ironed out — for Borland and Microsoft 32-bit compilers, at least. Another disadvantage is that it is a free-standing tool and not part of an IDE, so it lacks some of the convenience that IDE's offer in being able to switch quickly between editing and debugging. However, it is not a big deal to have the debugger and IDE (or command line) open simultaneously in separate windows and just switch between them.

This is the best debugger to use for located protected mode applications.

Soft-Scope: This debugger's greatest strength is the fact that it can be used for debugging *pmEasy* (EXE) applications. This is due to the fact that a locator is offered that we use to extract the debug symbolics from the executable and because it offers a linkable version of the target monitor that was not difficult to integrate with *pmEasy*. This debugger offers basic capability and allows viewing protected mode structures. However, it lacks much of the capability of VisualProbe and is generally not as well done. (Note that VisualProbe cannot be made to support debugging *pmEasy* applications due to lack of a locator or utility to extract the debug symbolics and locate them to match where *pmEasy* loads the application.) Unfortunately, this debugger will not be further improved, but we are permitted to continue to offering it.

Paradigm C++ IDE Debugger: This is a good tool and Paradigm is working to make it better. It is based on the Borland IDE, but Paradigm has already extended it for embedded use. In particular, they have done a nice target interface and have added a couple new windows for displaying registers and memory dumps, separately from the dreaded CPU window. Some features for embedded debugging work only with hardware support (emulator or JTAG) and not with the software target monitor. Most unfortunate is the inability to debug hardware isr's (without emulator or JTAG). Attempting to break in our keyboard or tick handler, for example, causes the debug session to end. This is handled fine by VisualProbe and Soft-Scope. As one would expect, C/C++ support is excellent since the compiler and debugger are a matched set, both under Paradigm's control (as is the rest of the toolchain). Breakpoint support is quite good, as discussed in the notes above. The source code and other displays are nice. However, it would be a big improvement in usability if Paradigm would integrate the mixed mode source/disassembly view into the code window, as other debuggers do, so the CPU window would no longer be needed. The "speed menus" (context-sensitive menus accessible by right-clicking) are a nice way to present the subset of options relevant to a particular window and make it easy for the user to see the actions that are possible for it. The fact that it is part of the IDE makes for convenient operation when switching between editing, compiling, and debugging. A surprising amount of effort was apparently put into scripting (macro) support, judging from the full manual provided on the subject, so those who wish to customize the environment should be in luck. The debugger has a few rough edges to work out. For example, when new code windows (and some others) are

initially opened, they are too small. It would make sense for new code windows to be created the same size as the one that was just active, or to create them at some easily specified size. Another example, is that the "Run to" command doesn't work, for some reason. Unfortunately, this is one we frequently use. All in all, it's a good tool. Even better is the fact that Paradigm is actively improving it and is open to suggestions. Some improvements to features discussed may be made in the near future.

Developer Studio: One of this debugger's greatest strength is its symbol handling. Since the debugger and compiler are a matched set, there are not problems with data types as often occur in 3rd party debuggers, such as VisualProbe, whose development teams are not aware of changes made to symbolic formats made by the compiler vendors. It follows all of the user interface conventions of a modern Windows application (context-sensitive menus, docking windows, expanding windows, sliding toolbars, etc), and its interface is generally well done, but it is somewhat wasteful of screen space. Scroll bars and borders between panes could have been made thinner. The Variables window is particularly clever; when the Auto tab is selected, it rotates the list so that variables of most relevance at the current execution point are shown at the top of the list. Every step shuffles the view to present the most relevant data. The result is that more often than not the variable you are interested in is at the top of the list.

It's biggest disadvantage is that it is not intended for debugging embedded software. It offers no access to protected mode registers or structures. It does not display descriptor tables, so it is not possible to ensure, for example, what isr is hooked to a particular interrupt vector. Also, it provides no target monitor for embedded targets; the only monitor provided runs only on a Win32 PC. Beyond this, there are some annoyances in the debugger. For example, there is no Run To command that will let you easily run to a function by name. There is also no View At function. Instead, you must know what module the routine is in, open it, scroll down to find the routine and then do a "go to here" — a major distraction from debugging. Documentation is provided only in on-line form via the Help menu and is very brief; it is part of the Visual C++ User's Guide in the online help. If this is the complete documentation, then this debugger has very little functionality. If there are more advanced features that are not covered in this documentation, then this tool is probably best learned from others already expert in using it.

The best use of this debugger for embedded purposes would be to debug real-time systems that can run as EXE's on a PC, where the BIOS and a protected mode environment such as *pmEasy* take care of the initialization and startup details, so that all that is being debugged is high-level application code. For traditional embedded targets (i.e. located code), a true embedded debugger is the better tool.

Turbo Debugger (real mode): This debugger has had a loyal following in the DOS application community. Paradigm's variants of it were widely used in the embedded field for real mode systems. Because of this, many people are familiar with it, so it serves as a basis for comparison in our chart. We recommend that new projects use Paradigm C++, unless the target is a DOS PC (or a PC running our real mode bootloader), where an EXE is the desired program format. Some commands, such as Goto (the View At command), are accessible only by right-clicking the mouse. They aren't available from the main menu, which is a problem if the user is not in the habit of right-clicking. This is a good tool, but its text mode interface is quite unwelcome to those who have used a debugger with a GUI.

SMX and *pmEasy* are registered trademarks of Micro Digital Inc. *smxAware* and *WinBase* are trademarks of Micro Digital Inc.

Micro Digital Inc sales 1-800-366-2491 support (714) 437-7333 fax (714) 432-0490 sales@smxinfo.com smxd\tech\dbgchart.doc 11/8/00