

pmEasy[®]

Protected Mode Environment

pmEasy offers a middle-ground between Windows- or Linux-based systems and real-time systems which are burned into EPROM or flash.

Like a desktop OS, *pmEasy* starts in real-mode, switches the processor to protected mode, loads the application into RAM, and starts it running. The application is in the form of a *relocatable* exe file. Unlike desktop OS's, *pmEasy* is designed to support hard real-time requirements. It is also compact and fast.

Unlike DOS-extenders, which have also been used for embedded systems, *pmEasy* does not greatly increase interrupt latency nor task switching time. (See sidebar on page 3 for more discussion of DOS extenders.)

Hence, *pmEasy* is appropriate for embedded systems which have some form of disk, and for which, locating and burning code into ROM is not appropriate or not desired. (e.g. The application is large and complex or is in a state of flux.)

Application Loading

An important benefit of *pmEasy* is that it avoids the complexity of using a linker/locator. There is no worrying about addresses, descriptor tables, or how to start from system reset. The application is linked with the native linker of the Borland or Microsoft C/C++ compiler and *pmEasy* loads the resulting exe file from disk into RAM.

The following devices are supported:

- Floppy Disk (FAT12)
- IDE Hard Disk (FAT16, FAT32)
- LS-120
- CDROM (ATAPI)
- M-Systems DiskOnChip[®] Flash Disk
- DOS devices (*see chart*)

Features

- Easy access to protected mode
- Loads protected mode executables from:
 - Floppy
 - IDE Hard Disk
 - DiskOnChip[®]
 - CDROM
 - LS-120
 - IOmega Zip
 - DOS Devices
- No need to locate code.
- Support for either 16-bit or 32-bit protected mode.
- *pmScope* version supports debugging.
- No increase in interrupt latency.
- No increase in task switch time.
- Allows temporary switch to real mode for hardware initialization and other purposes.
- Boot loadable. DOS not required.
- DOS loadable. Exits cleanly back to DOS.
- With *unDOS*, provides an excellent upgrade path from DOS. (See *unDOS* brochure for details.)
- Source code included (except for exe loaders and device drivers).

There are two versions of *pmEasy*: (1) *pmEasy16* for 16-bit, segmented protected mode, and (2) *pmEasy32* for 32-bit, flat protected mode.

pmEasy16 is particularly advantageous when upgrading from DOS. It loads exe files in the NE (New Executable) format, as used by Windows. This type of exe file is generated using the same compiler and linker as for real mode, but with the PROTMODE module statement.

pmEasy32 loads exe files in the PE (Portable Executable) format used by Windows NT. The exe file is a console-mode executable generated by a 32-bit, flat-mode compiler and linker, such as Borland C++ or Visual C++.

Loading *pmEasy*

pmEasy can operate with or without DOS present. It can be loaded and run from the DOS prompt. When running from DOS, the drive letter and file name of the application program can be specified on the *pmEasy* command line (e.g. C:> pme c:myapp.exe). Also, *pmEasy* can be bound to the application so that there is a single executable (e.g. C:> myapp). The ability to run from DOS is especially convenient during development. It is also useful for targets shipping with DOS. In such systems, *pmEasy* will exit cleanly back to DOS when the application is stopped.

For systems, with only a BIOS, *pmEasy* can be bootloaded from disk. The exe2bin (or equivalent) utility is used to generate a binary file for *pmEasy*. A utility is included with *pmEasy* to replace the disk boot sector in order to load the binary image of *pmEasy*. *pmEasy*, in turn, loads the application. DOS is not required in this process.

pmEasy does not depend upon DOS or upon BIOS to run. These are simply convenient ways to start it and to do hardware initialization. It is possible to locate *pmEasy* to run from ROM. In this case, the application code must include its own hardware

initialization code — something the BIOS would normally take care of.

pmEasy includes a simple disk reader and drivers for loading from all of the devices listed on page 1. Any combination of drivers can be selected from the makefile used to make *pmEasy*. Typically, only one driver would be linked, but if the system has more than one type of drive, as many drivers, as necessary, may be linked to *pmEasy*.

Note: Device drivers included with *pmEasy* are modified for *pmEasy* and provided in object form, only. They are not suitable for use with *smxFile*.

In addition, a special version of the disk reader is provided that uses DOS int 21H calls and DPMI 300H to load the program. *This provides the ability to load from any device that has a DOS device driver, if DOS is present!*

Sample applications and makefiles are provided. (If also using SMX, the Protosystem is the sample application.)

Protected Mode

When *pmEasy* is started, it initially runs in real mode. It is usually a DOS-style, real-mode exe, unless it is being bootloaded, in which case it is a binary file or unless it is being run from ROM, in which case it is a binary image. *pmEasy* sets up protected mode structures such as the global descriptor table (gdt) and the interrupt descriptor table (idt). It also hooks default exception handlers to the processor exception interrupt levels. Then *pmEasy* switches the 386-or-higher processor into protected mode. Next it loads the application into RAM and starts it running.

While the application is running, *pmEasy* provides a reduced DPMI¹ server. Services such as heap (i.e. RAM) allocation, local descriptor table management, and interrupt descriptor table management are provided.

¹DOS Protected Mode Interface — an API developed by Intel and others

The heap can be configured to span multiple, disjoint areas of memory using a simple table (see Flexible Heaps, below). Also, some specialty functions are provided, such as DPMI 300H. This function allows invoking a real-mode software interrupt (e.g. int 21H (DOS) or int 10H (BIOS)) after *pmEasy* has switched to protected mode. This function is for use during initialization, such as to initialize a display controller to a particular graphics mode. (This ability can be a real life saver when the manufacturer of the display or other hardware provides only a real mode (e.g. BIOS int 10H) driver.)

EXE Debugging

For debugging the application code, *pmScope* and *pmLoc* are used. *pmLoc* is used to generate the debug symbolics for *pmScope* from the exe file. A special version of *pmEasy* that has been linked with the debug monitor, *pmMon*, is run on the target platform. It can be loaded the usual ways and, in turn, it loads the application from disk. Meanwhile, the user loads the symbols into the debugger on the host. When the exe load is complete, debugging can begin.

Another option is to use the VisualProbe debugger. This requires locating the application with Link & Locate 386. With this approach, Embedded *pmEasy* is used during debugging. For normal execution, the application is built as an exe, as usual. For *smx* users, this option is provided in the Protosystem makefile.

Flexible Heap

pmEasy's heap can span multiple blocks of available RAM. The user specifies a table of starting and ending addresses of free memory blocks and *pmEasy* initializes the heap accordingly. This allows using all available RAM. For example, memory from the top of *pmEasy* to 640K and above 1M can be included in the heap.

The Problems with DOS Extenders

DOS extenders work ok for desktop applications. A DOS extender does what *pmEasy* does, but it also accepts DOS calls. When a DOS call is made by the application, the DOS extender switches to *virtual 86* mode and invokes DOS. When DOS is done, it switches back to protected mode with the results of the DOS operation. Sounds simple? It isn't.

Due to the complex architecture of 386-class processors, the DOS extender must do several hidden functions such as copying and switching stacks. While involved in these lengthy endeavors, interrupts are disabled for long times.

Also, while in virtual 86 mode, hardware interrupts cannot be processed by protected mode interrupt service routines. Enter the "umbrella interrupt handler" which switches back to protected mode (again copying and recopying stacks) so the protected mode isr can run!

The above hidden features seriously impact interrupt responsiveness.

Unlike DOS, a task switch cannot be performed while any part of the DOS extender is running — including the umbrella interrupt handler. This makes preemptive multitasking almost impossible to implement.

Finally, because of the foregoing complexities and because source code is not available, debugging a real-time application using a DOS extender often becomes a nightmare. We have encountered many projects, which are unable to find and fix serious bugs.

Automatic memory sizing is also supported. This is accomplished by specifying a special value for the ending address of the last region in the table. *pmEasy* searches for the end of RAM, in user-specified increments, and sizes this block of heap as large as possible. This is useful for systems that are shipped with variable amounts of RAM or which may be upgraded in the field.

OMF Debugging

For those who prefer to use a debugger, which accepts only omf files (e.g. VisualProbe), a reduced version of *pmEasy* called *Embedded pmEasy* is included in the *pmEasy* package. See the *smx86* brochure for discussion of this option. Basically, it permits debugging with omf files, but running with exe files.

Hex Loader

The exe file loader, in *pmEasy*, can be replaced with a hex file loader. Hex files are one type of file produced by linker/locators. A hex file is not relocatable. Also a hex file is typically twice as large as the corresponding exe file. However, since each record is check-sum protected, a hex file is more secure for downloading via LAN or serial link.

The hex file loader can also be used with *Embedded pmEasy* to permit faster application loading into the target, during debugging. (Either via LAN to target hard drive or via floppy to target floppy drive.)

pmEasy Sizes²

	<u>Code</u>	<u>Data</u>	<u>Total</u>
<i>pmEasy</i> ³ (add drivers below)	21K	10K	31K
Floppy Driver	4K	1K	5K
ATA/ATAPI Driver (Hard Disk, LS-120, and CDROM)	6K	1K	7K
CDROM Driver (not incl. IDE driver)	16K	7K	23K
M-Systems DiskOnChip Driver	11K	0K	11K
<i>pmEasy</i> ³ with DOS disk reader (drivers not necessary)	18K	13K	31K

²Sizes are with standard settings. Disabling video output and other features reduces *pmEasy* size significantly. Also some features use buffers of configurable size. If memory is tight, please contact us to determine an accurate size for the exact configuration you plan to use.

³Size of *pmEasy*³², including PE loader and disk reader. *pmEasy*₁₆ + NE loader size is similar. *pmEasy* is about 2K smaller with hex loader instead of PE loader.

Supported DPMI Calls (C API)

Memory Management Services

errCd	pmiAllocMem (size, linearAddrPtr, handlePtr);	Allocates a block of memory, returning a pointer to the block and a handle to its control block
errCd	pmiAllocMemTop (size, linearAddrPtr, handlePtr);	Allocates a block of memory at the highest address available in the heap (non-standard)
errCd	pmiFreeMem (handle);	Frees the specified block of memory and merges it with adjacent free blocks
errCd	pmiBaseToHandle (base, handlePtr);	Returns handle of block at base address in <i>*handlePtr</i> (non-standard)
errCd	pmiHeapStats (statStructPtr);	Returns amount of heap remaining, number of free blocks, and size of largest free block in <i>*handlePtr</i> (non-standard)
errCd	pmiSizeMem (handle, sizePtr);	Returns size of allocated memory block in <i>*sizePtr</i>

LDT Descriptor Management Services

errCd	pmiAllocLD (num, selPtr);	Allocates one or more consecutive descriptors in the local descriptor table (LDT)
errCd	pmiAllocLDTop (selPtr);	Allocates one descriptor from the highest slot of the local descriptor table (LDT) (non-standard)
errCd	pmiFreeLD (sel);	Frees the specified descriptor in the LDT
errCd	pmiGetBaseAddr (sel, linearBaseAddrPtr);	Gets the segment base address in the specified descriptor
errCd	pmiSetBaseAddr (sel, linearBaseAddr);	Sets the segment base address in the specified descriptor
errCd	pmiSetLimit (sel, limit);	Sets the segment limit in the specified descriptor
errCd	pmiSetRights (sel, rights, xrights);	Sets the segment type and access rights in the specified descriptor
errCd	pmiCreateAlias (sel, aliasPtr);	Creates a new descriptor in the LDT identical to the one specified, but set as a read/write data segment
errCd	pmiGetDescr (sel, bufPtr);	Copies the specified descriptor into an 8-byte buffer pointed to by <i>bufPtr</i>
errCd	pmiMapRealSeg (seg, selPtr);	Allocates and initializes a descriptor to map a real mode segment
errCd	pmiGetSellnc (incPtr);	Returns the segment descriptor size (8) in <i>*incPtr</i>

Interrupt and Trap Management Services

errCd pmiGetPMIntVect (intno, (**isrPtr()));	Gets isr address from interrupt descriptor table entry at level <i>intno</i> (levels 0-255)
errCd pmiSetPMIntVect (intno, (*isr()));	Loads isr address into interrupt descriptor table entry at level <i>intno</i> (levels 0-255)
errCd pmiSetExcepVect (exno, (*esr()));	Loads esr address into interrupt descriptor table entry at level <i>exno</i> (levels 0-31)
errCd pmiSimRealInt (intno, *regs);	Invokes a real mode software interrupt. First sets registers as specified in structure pointed to by <i>regs</i> . (DPMI 300H)
errCd pmiGetIntDescr (sel, buf);	Copies interrupt descriptor into 8 byte buffer (non-standard)
errCd pmiSetIntDescr (sel, buf);	Sets interrupt descriptor from 8 byte buffer (non-standard)

errCd is 0 if ok, else specifies nature of error.

Summary of Features

- 386 or higher
- *pmEasy16* supports 16-bit PM applications
- *pmEasy32* supports 32-bit PM applications
- Enters protected mode and stays there — no switching back and forth into and out of real or V86 mode, which severely hurts performance.
- Subset of DPMI services, compatible with DPMI specification version 1.0 (Intel Order No. 240977-001)
- ROM'able, bootable, or DOS loadable
- Standard *pmEasy* loads from many types of disk with the provided drivers. Other disks can be read using DOS file i/o, if DOS present.
- Will exit cleanly to DOS, if started from DOS
- Allocates available memory below and above 1MB (uses free memory table created by user)
- Standard *pmEasy* supports *pmScope* debugger
- Embedded *pmEasy* supports VisualProbe debugger
- Low-cost PC tool integration already done
- Built with MASM and Microsoft 16-bit C compiler, both available cheaply.
- Supports both PC and non-PC compatible targets
- Simply designed, easy to understand, well-structured code
- Source code is provided. Promotes understanding and allows customization
- Compatible with the SMX product family
- Works with or without SMX.

pmEasy and *smx* are registered trademarks of Micro Digital, Inc.
SSI, Link&Locate 386, and VisualProbe are trademarks of Embedded Power Corporation
Soft-Scope is a registered trademark of US Software