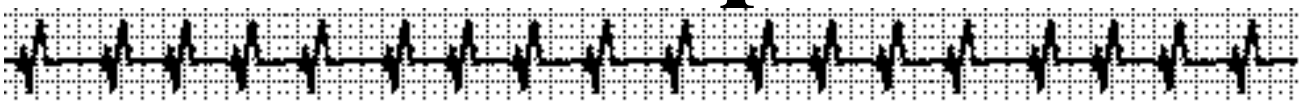
INIT-Scope 2.0

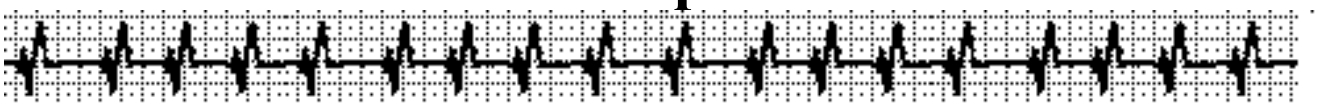


by

David P. Sumner

INIT-Scope 2.0 is Shareware
If you decide to keep it send
$15 to
David Sumner
1009 Walters Lane
Columbia, SC 29209

INIT-Scope 2.0



Introduction

INIT-Scope is a cdev/INIT combo that meshes with the startup mechanism at the lowest levels of your Macintosh. The INIT portion of INIT-Scope monitors the entire loading process, and provides a great deal of information concerning what happens to your computer during this critical phase. The cdev portion allows you to choose among several options, and also provides some useful utilities.

What Good is It?

INIT-Scope will provide you with a highly detailed report of the startup activities of your Macintosh. If you are experiencing any mysterious problems with your computer,
INIT-Scope may provide the essential clues that you need to understand what is happening. Moreover, if you are an INIT-Developer, you will find INIT-Scope of great value in debugging your INIT.

Maybe you are just curious as to what all those INITs are doing to your computer, or you want to know how they work; INIT-Scope is just the tool to provide the information you need.

INIT-Scope tells you:
1. What the environment of your computer is like at startup.
2. Which traps are patched by each INIT.
3. Which low-memory global variables are changed by each INIT.
4. What resources are loaded in by each INIT (and the order in which they were loaded).
5. A Trap History[1] of each INIT. This includes the parameters for many of the traps.

…and MUCH more.

And INIT-Scope does this *without patching any traps itself*!

Installing INIT-Scope

System 6

If you are using system 6.x, then simply place INIT-Scope into the system folder on your startup hard drive.

[1]Virtual Memory must be off to get the trap history portion of the report.

System 7

In system 7.0 lingo, INIT-Scope is an extension/cdev combination. To work properly, you must place INIT-Scope into the Extensions folder, and place an alias of INIT-Scope into the Control Panels folder. If you simply drag INIT-Scope into the system folder, the Macintosh will ask you if you want to place it into the Control Panels folder. When this happens, just say, “No.” It is important that you place the *alias* into the Control Panels folder, and INIT-Scope itself into the Extensions folder.

Once installed, you can adjust the operation of INIT-Scope by opening it up as a control panel. The possible options are described later in this document.

Using INIT-Scope

Who Can Use INIT-Scope?
Basically, INIT-Scope is a technical tool, and is of maximum use to a skilled programmer. However, it is of considerable value to others as well.

1. Even if you do not understand everything in the *Startup Report* that INIT-Scope produces, you will most likely still be able to get a fairly good idea of what your INITs (or extensions) are doing to your system.

2. You can use INIT-Scope's INIT skipping feature to skip loading particular INITs at startup time.

3. By providing a technical person with the data provided in INIT-Scope's Startup Report, (s)he may be able help you figure out any kinds of problems that you might have.

Using INIT-Scope
INIT-Scope is easy to use. Just install it into your system folder as described previously, open up the control panel, and choose the options you want. Then reboot your computer. The rest is automatic.

Note: If you are using Virtual Memory, you will find that most of the information in the startup log is missing. If you want to get a full report from INIT-Scope, power-up with Virtual Memory turned off. There is no problem with 32-bit memory settings.

What's in a Name?
Note that INIT-Scope's name has a leading space character. This is done so that it will load earlier than (most) of the other INITs in your system folder. INITs/cdevs/RDEVs load in alphabetical order. Thus you may rename INIT-Scope by placing a different leading character if you want to change its position in the startup process. If you redistribute
INIT-Scope to a bulletin board or other source, please leave its name as ‘ INIT-Scope’ with a leading space (also be sure to include this documentation.)

Skipping INITs
If you check the 'Skip INITs on Shift Key' option in the INIT-Scope cdev, then you

can skip any INIT in your system folder by simply depressing the Shift key[2] (but no other modifier keys) just before it loads in. Generally, this amounts to depressing the Shift key right after the icon of the previous INIT appears. When INIT-Scope detects the Shift key it will abort the loading of the next INIT, emit a short beep as a signal, and delay for two seconds to give you time to release the Shift key. Of course, to use this feature you must know the order in which your INITs load.

If you are having a system bomb at some point during the startup process, it might be a good idea to use this feature to simply skip all the INITs in your system folder—a lot easier than booting with a system disk on an external floppy diskette.

[2]Note that if you hold down the Shift Key at the beginning of the start-up of a machine running system 7, then *all* INITs will be disabled.

One reminder—some INIT or cdev files may contain several distinct INITs. For example Easy Access has three. Thus to completely bypass Easy Access in this manner would require leaving the Shift key down for all three INITs.

Turning Off INIT-Scope

Although it provides you with a great deal of information, you do not generally need to keep INIT-Scope turned on (unless maybe you want to use one of the 'Skip INITs' or 'Intercept INITs' feature).

You can turn INIT-Scope off from the control panel. After this, INIT-Scope will not execute again until you turn it back on.

When INIT-Scope intends to load and execute, it displays the icon  .

If INIT-Scope is resident, but does not execute, it will display the icon 

instead.

If INIT-Scope is turned on in the cdev, but you would like to bypass it in the startup process, just press down the Shift Key as INIT-Scope is about to load.

The Startup Report[3]

INIT-Scope produces a text file report called *Startup Report*. You will find this file on the root level of your startup disk. This report contains the following information:

1. The names and trapwords of all traps patched and the address of the patches.

2. The basic information about your system—such as type of computer, keyboard, amount of RAM, and the values of important low memory locations.

3. Information about the resources used by the various INITs during the startup process. This includes each resource type, ID and handle.

[3]The report is best read using size 12 Times as the Normal font.

4. Addresses of all VBL routines. This includes the address of VBL routines loaded prior to the execution of INIT-Scope itself. Along with the address of each VBL is the phase and count values associated with it.

5. Addresses of all shut-down routines installed. Along with the address of each such routine is an indication of what stage of the shut-down procedure calls the routine.

6. Addresses of all installed time manager routines.

7. Addresses and associated strings of Notification requests.

8. The start and end of the application heap at the time each INIT loads.

9. The system heap expansion caused by the INIT.

10. The actual system heap RAM used by the INIT. (This is based upon the amount of free memory in the system heap before the INIT executes compared with the free space afterwards. In addition, the size of a largest free block of RAM is provided as well.

Finally, one of INIT-Scope's finest features,

10. A *detailed* description of the trap history of the loading process.

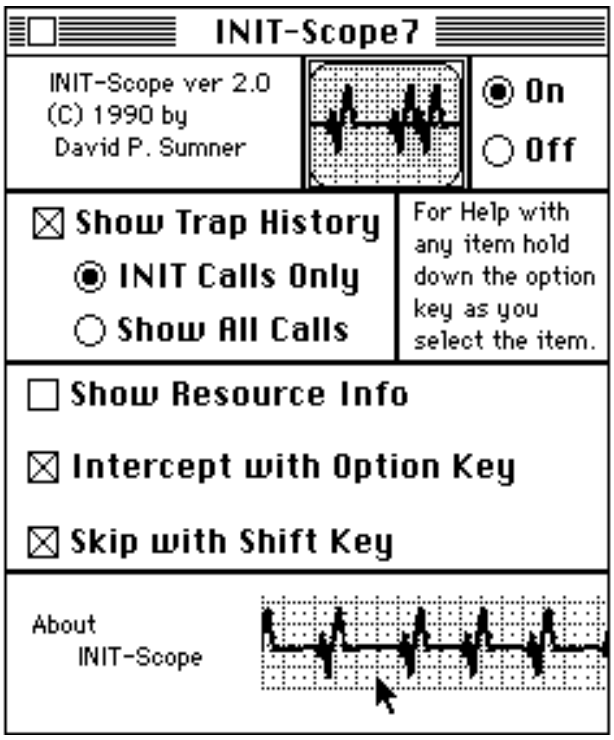
Format of the Startup Report

INIT-Scope's output file begins with a brief header and description of the environment of the machine in which it resides.

After this system information, the Startup Report provides detailed information about each extension as it is loaded and executed by the system. For each INIT, the INIT-Scope report shows the system environment in which the INIT loads and also the effect that the INIT has on this environment. In particular, INIT-Scope shows the application heap at the time the INIT loads (the system heap is an immediate derivative of this—it lies directly below the application heap). Additionally, this portion of the report shows how many bytes of system RAM and High RAM the INIT uses. Also the amount of RAM the INIT requests (through its 'sysz' resource) is also noted—requests for less than 16K are treated by the system as requests for 16K). Note that the system heap need not expand in response to this request if there is already sufficient memory available in the system heap.

This section of the report also shows the true size of the INIT in bytes.

Exactly what the file looks like will depend upon the options you select in the INIT-Scope cdev in the Control Panel.



Note that on-line help is available by clicking on any

button while holding down the option key.

If you have checked the *Show Resource Info* box, then for each INIT you will see all the Resource Types accessed by the INIT (either directly by the INIT or indirectly through calls made by toolbox traps that are used by the INIT). These Types will appear in the order in which they occur in the loading process.

Interspersed with the resource types are a number of other useful pieces of data. This includes the addresses and names of all traps patched by each INIT, all VBL (clock interrupt) routines installed, Shutdown manager routines installed, Notification requests, and much more.

For example, if the *Show Resource Info* is checked, then this portion of a *Startup Report* file might look like this:

```
          ----------------------------------
          >File Sharing Extension ( INIT) <
          ----------------------------------
Size of this INIT in bytes: 2560
BufPtr: $726E90
High Ram Used (bytes): 0
Requested System Heap space (bytes): 0
System Heap Expansion (bytes): 44968
System Heap Used (bytes): 2616
Application Heap:   $F4164 -   $F5964
Free in System Heap (bytes): 66752
Largest Free Block in System: 39724  Diff= 27028
INIT Handle:    $61814

            Patches / Installations
          ---------------------------
  ResType STR
  ResType GBRL
  $A063 Patch:    $A9732  MaxApplZone
  $A065 Patch:    $A9BA0 StackSpace
VBLProc installed at:    $A9BE4  Count:      $C  Phase:   $0
   $1C9  Patch:    $A9716  SysError
  $A094 Patch:    $A9ADC            xxxx

          Low Memory Globals Altered:
          -----------------------------------
(    $3E2  )        FSQueueHook     Changed from:    $2B696   To:    $A9B74
(    $B50  )        Twitcher1           Changed from: $FFFFFFFF   To:   $0
```

This would tell you that this INIT loaded a resource of type STR , and then one of type GNRL. Then it patched the toolbox trap MaxApplZone with a routine of its own at address $920C2 (the '$' indicates a hex value). It also patched the StackSpace trap, and then installed a VBL procedure at address $92574. It then patched the SysError trap and the trap with trap word $A094.

There are a great variety of other kinds of information that this portion of the Startup Report file provides. for instance, consider this segment of the output for the INIT *Soundmaster*.

```
VBLProc  $417C82  Count:      $1  Phase:   $0
Removed VBLProc:  $417C82
Removed VBLProc:    $792E
  $A9C8  Patch:    $17B80  SysBeep
Shut Down Routine at:    $17B94   Called before: Restart
Shut Down Routine at:    $17B98   Called before: Power Off
Removed VBLProc:    $792E
  $A017  Patch:    $17B8C  Eject
  $A02F  Patch:    $17B88  PostEvent
```

Here we see that SoundMaster Installed a VBL routine and then shortly thereafter removed it. It then patched the trap SysBeep. After this, it installed two shutdown routines; one to be called before the computer restarts, and another to be called just before the power is turned off.

Finally, SoundMaster patched the traps *Eject* and *PostEvent*.

As one last example, here is the initial portion of the report on *SndControl*.

```
          ----------------------------------
          >SndControl  ( cdev) <
          ----------------------------------
Size of this INIT in bytes: 14758
BufPtr:  $6D3070
High Ram Used (bytes): 0
Requested System Heap space (bytes): 40960
System Heap Expansion (bytes): 24616
System Heap Used (bytes): 29586
Application Heap:    $98778 -   $99F78
Free in System Heap (bytes): 16952
Largest Free Block in System: 13744  Diff= 3208
INIT Handle:    $1D7E4

            Patches / Installations
          ----------------------------
  $A00F Patch:    $4A23A MountVol
  $A9C8          Patch:    $4A15A SysBeep
  $A017 Patch:    $4A1AE Eject
  $A00E Patch:    $4A1F4  UnMountVol
  $A9F4 Patch:    $4A318  ExitToShell
  $A9F2 Patch:    $4A328  Launch
  $A9A0          Patch:    $4A398  GetResource
  $A92D          Patch:    $4A35A CloseWindow
  $A9C9          Patch:    $4A338  SysError
  $A97B          Patch:    $48F86  InitDialogs
Shut Down Routine at:    $4A510   Called before: Power Off
Shut Down Routine at:    $4A528   Called before: Restart
Shut Down Routine at:    $4A540   Called before: Closing Drivers

          Low Memory Globals Altered:
          ------------------------------------
(    $29A  )        JGNEFilter          Changed from:    $36D0E   To:    $48DFE
```

The Startup Report file can provide much more information than what we have indicated here. The sections that follow will elaborate on this.

The appendix at the end of this documentation discusses the Startup Report in much greater detail.

Patches and Hooks

Obviously, it is very useful to know which traps are patched by the various INITs that reside in your system. If two INITs patch the same trap(s) then there is a potential for trouble (although well-written INITs can generally patch the same traps without stepping on each other's toes). Moreover, this information gives you at least a rough idea of how the INIT performs its magic.

For instance, it should come as no great surprise that *Boomerang* patches the StdFile trap. This is how it can get that little boomerang icon into the Standard File dialog box every time. You will not be shocked to discover that the virus protection INITs patch traps that modify resources. By patching such traps, these utilities can intercept a virus that is trying to add or modify a system resource and refuse it access. On the other hand, you might be surprised to discover that the INIT *the Grouch* [4] (previously *Oscar*) patches the traps MenuSelect and CopyBits. Well, MenuSelect is pretty obvious since the Finder's 'Empty Trash' menu item is changed to 'About the Grouch', but why CopyBits? Well, I'm guessing because I haven't looked more closely at it, but probably the idea is based around the fact that when you throw something into the trash, the trash icon changes and it is a call to CopyBits that causes the icon to change.

In spite of this, there are a number of pitfalls that you must avoid.

Delayed Patches (or Don't Believe Everything You Read)

It is naive to believe that just because a patch occurs during the time that a particular INIT is loading, it is the INIT that is doing the patching. This may not be the case. It may not even have anything at all to do with the INIT!

For example, when the INIT *Suitcase* executes, it patches a whole slew of traps. In fact the *Startup Report* file will show that the following traps are all patched by *Suitcase*. (Of course the actual addresses o the patches would vary from machine to machine.)

```
$A996  Patch:      $3589E  RsrcZoneInit
$A9A1  Patch:      $35952  GetNamedResource
$A9A8  Patch:      $359FA  GetResInfo
$A9A0  Patch:      $35F10  GetResource
$A9A2  Patch:      $35DF6  LoadResource
$A9B0  Patch:      $35E3E  WriteResource
$A999  Patch:      $36D66  UpDateResFile
$A9AB  Patch:      $363DE  AddResource
$A9AD  Patch:      $36424  RmveResource
$A99A  Patch:      $36DD0  CloseResFile
$A99D  Patch:      $35BCC  GetIndResource
$A80E  Patch:      $35BDE  Get1IndResource
$A998  Patch:      $35C2E  UseResFile
$A94D  Patch:      $3660E  AddResMenu
$A951  Patch:      $36616  InsertResMenu
```

[4] If you haven't already tried this neat INIT, get it off of just about any online service. It provides some nice special effects whenever you empty the trash.

$A93D	Patch:	$36BDC MenuSelect
$A9B6	Patch:	$35CD0 OpenDeskAcc
$A9B7	Patch:	$35D90 CloseDeskAcc
$A023	Patch:	$35DE4 DisposHandle
$A995	Patch:	$357DC InitResources

```
$A935  Patch:      $3659C  InsertMenu
$A932  Patch:      $36566  DisposMenu
$A9A3  Patch:      $36578  ReleaseResource
$A997  Patch:      $36232  OpenResFile
$A000  Patch:      $362D0  Open
$A00C  Patch:      $36332  GetFileInfo
```

Well, that's all very well and good, and in fact all of these patches really are caused by *Suitcase* itself. However, no matter what INIT loads next, you will see (essentially) the following in the next INIT's portion of the report:

```
$A99D  Patch: GetIndResource    $413CA4 (Patched by previous INIT)
```

Now, INIT-Scope is telling you that it has detected that this patch is not really due to the current INIT, but is actually caused by an earlier INIT. It is not terribly unusual for an INIT to patch a trap, which then in turn patches other traps when it is next executed.

This is one example of a delayed patch. In fact it does not matter what INIT executes after Suitcase; you will always see this reference to a patch to GetIndResource. The reason for this is that the system code that is responsible for loading INITs at startup time calls a trap that triggers the patch to GetIndResource. So although this patch occurs very close to the time that *Suitcase* executes, it does not occur while *Suitcase* is executing.

If you run INIT-Scope on a system that contains *Strtscrn* and *Black Box* and if Strtscrn executes after Black Box, then you will see a patch to the toolbox trap PaintOne that appears to be due to *Strtscrn*, but is really caused by a delayed patch of *Black Box's*. (*Black Box* patches InitDialogs which (apparently) in turn patches PaintOne when it is called.) Since *Strtscrn* uses the toolbox trap InitDialogs, it triggers the delayed patch by *Black Box*.

In fact, there will be some patches that will not be made until after the startup process has terminated. Naturally, INIT-Scope will miss these. A later version of INIT-Scope will continue to monitor the system even after the Finder takes over, and these patches and other extensions/alterations will not escape unnoticed.

Hooks

There is more than one way to patch the system. The Macintosh contains a variety of low memory vectors that allow a user to install patch code that will be called at a prescribed time. Perhaps the most frequently used such hook is JGNEFilter. Any routine whose address is placed in this vector will be called at a special point during the operation of the crucial GetNextEvent (or WaitNextEvent) trap that is central to every Macintosh application. This is effectively a patch to the GetNextEvent trap, but it does not appear as such. You can determine just which low-memory globals INIT-Scope Checks by modifying the 'HOOK' resource (described later).

A Word About Monitoring Low Memory Global Variables

The 'HOOK' resource mentioned above, contains the names and addresses of all the low memory global variables that INIT-Scope can recognize.

This resource purposely does not contain all the possible low-memory globals. Only certain ones of these are likely to be of any significance to any INIT. For example there are low-memory globals that monitor the location of the cursor, but it is not likely the INIT will alter this location, and even if it did we would probably not be able to determine that it was

in fact the INIT that did so and not just the user moving the mouse. Many variables such as MenuHook, JGNEFilter, and others are sometimes altered by INITs as another way of patching the system. All these variables are contained in the default 'HOOK' resource.

---- INIT-Scope does not display the values of spurious global variables. ----

Moreover, there are many low-memory values that get changed during the INITialization process that are changed by the ROM, and not explicitly by the INIT. It is unlikely that these values will be of any interest. For example, the RMgrHiVars global holds the resource type during many calls to the Resource Manager. Thus there is no point in checking to see if it changed before and after the exectution of the INIT.

However, in the event that you do want to monitor the changes to such variables, you need only add their names and addresses to the 'HOOK' resource.

The GloB Resource—Following the Progress of a Particular Variable

Now, all of the variables in the HOOK resource are monitored only at the end of each INIT's actions. If the variable's value changed, then INIT-Scope reports the new value. For the variables contained in the HOOK resource, this is all that is generally needed. Such variables will usually only be changed once, if at all, by any INIT.

However, you may want to monitor the changes to a variable more closely than this. INIT-Scope lets you do this via its 'GloB' resource. This resource contains a single long word. If the value in 'GloB' is $FFFFFFFF, then INIT-Scope will not monitor any variable. Otherwise, it closely monitors the long value at the address stored in 'GloB.' The (long) value stored at this address is monitored after every trap call (from any source), and if it has changed since the last trap call, the change is noted in the trap history portion of the *Startup Report*.

The Trap History

INIT-Scope intercepts the INIT process at its very roots, and hence can closely monitor the proceedings. One consequence of this is that INIT-Scope can display, as part of its report, a detailed account of the toolbox traps used during the INITializing process. In fact, if you select the 'Show Trap History' option, then the *Startup Report* will contain the name of every trap that the INIT calls during its execution. For many traps the values of the parameters passed to the traps is provided as well.

If you select the 'INIT Calls Only' option, then only traps used by the INIT will be reported.

If you select the 'Show All Calls' option for trap history, then all trap calls will be reported no matter whether they came from the INIT or not. Needless to say, this will produce a L-O-N-G report and probably it is best not to select this option unless you really need the information.

An example of a portion of the trap history segment of an *Startup Report* is:

```
        Trap History
        ----------------
$A746  GetTrapAddress  Unimplemented
$A346  GetTrapAddress  SysEnvirons
$A81F  Get1Resource   sNot  #0   ResHandle:    $4C120
$A9A3  ReleaseResource     $4C120
$A746  GetTrapAddress  Unimplemented
```

```
$A346  GetTrapAddress  Gestalt
$A746  GetTrapAddress  Unimplemented
$A346  GetTrapAddress  NMInstall
$A031  GetOSEvent
$A260  HFSDispatch  GetFCBInfo
$A036  MoreMasters
$A036  MoreMasters
$A036  MoreMasters
$A036  MoreMasters
$A81F  Get1Resource   InaP  #130   ResHandle:    $1D8C8
$A992  DetachResource     $1D8C8
$A064  MoveHHi     $1D8C8
$A029  HLock     $1D8C8
$A20C  GetFileInfo
$A974  Button
$A1AD  Gestalt  $612F7578  a/ux
$A81F  Get1Resource   sysz  #0   ResHandle:    $4C120
$A9A3  ReleaseResource     $4C120
$A522  NewHandle   Size (bytes): 30720
$A522  NewHandle   Size (bytes): 25600
                   :::::
                   :::::
$A1AD  Gestalt  $612F7578  a/ux
$A81F  Get1Resource   sysz  #0   ResHandle:    $4C120
$A9A3  ReleaseResource     $4C120
$A522  NewHandle   Size (bytes): 30720
$A522  NewHandle   Size (bytes): 25600
$A99B  SetResLoad
```

Note that you can tell a great deal from the trap history report. Not only can you surmise the logic of the INIT at a glance, but the values of the parameters allow you to locate important data inside the system heap or above BufPtr. Also, the trap history report can be used like a street map to trace through the INIT with a debugger, or as an aid to reconstructing the INIT's code with McNosy.

The kinds of traps whose data is provided falls into these categories:

1. Memory manager calls that deal with handles. The value of the handle is provided.

2. BlockMove (INIT-Scope tells you how many bytes were moved, from where, and to where.)

3. Resource Traps - the ResType and ID of the requested resource are both provided, as well as the handle returned by the resource call.

4. NewHandle and NewPtr. In each case the report shows the requested size of the handle or pointer.

5. For calls by HFSDispatch, the type of call is provided.

6. Pack Managers are displayed by name.

7. The parameters for other special traps such as Gestalt, GetTrapAddress, and SetTrapAddress.

The trap word of each trap is also provided—and often this is valuable. For instance, both A11E and A51E are legitimate trap words for the NewPtr trap. However, A11E will allocate the resulting block of RAM in the application heap, while A51E will allocate the block in the system heap.

Some Caveats

There are a few problems with the Trap History portion of the report. If you select to only see the calls made by the INIT, then INIT-Scope attempts to determine for each trap call whether or not it was made by the INIT. It does this by simply checking to see if the call lies in the range of memory beginning at the address of the INIT and extending up to the address of BufPtr at the time the INIT was loaded. Any call in this range will almost surely be from the INIT. A few anomalies creep in however, and you should be aware of them.

First, consider files such as *Easy Access*. This INIT File actually loads (at least the system 6 version) three distinct INITs, and you will miss most of the calls by easy Access unless you turn on the 'All Calls' option. The reason for this is as follows. The first thing that each of the INITs loaded by *Easy Access* does is to allocate a block of memory in the system heap, and then load some code resource into it. Then the INIT jumps to a location in that new block—which is generally lower in the system heap than the INIT itself. Consequently, INIT-Scope does not recognize these calls as being from the INIT. Note that had the allocated memory resided in high RAM, this problem would not occur.

Using INIT-Scope as a Debugging Aid

INIT-Scope makes it easy to intercept an INIT just after it is loaded by the system and just before it actually executes. INIT-Scope provides a lot of useful information that makes going back through the INIT with a debugger a lot easier. You can plan for breakpoints more intelligently and have better idea of what's going on.

To intercept an INIT during the startup process, simply hold down the option key (but no other modifier keys) just before the INIT is about to load (or, just after the previous INIT loads.)

You must have a debugger installed in order to use this feature.

INIT-Scope does check for the presence of a debugger, and ignores your request to intercept if no debugger is found.

When you fall into the debugger, you will see a note that tells you the name of the INIT that you are about to enter, and the next instruction (the one pointed to by the PC) will be:

```
jsr (A1);
```

The address of the INIT's code is in register A1, and you can follow its execution by stepping through the code at this point. The next step takes you to the first instruction in the INIT.

Aside from the comments made earlier (in the section on trap history), there are a few other things to be aware of if you want to use a debugger with INIT-Scope.

For one thing, if you generate a report, and then want to use that report as a guide to ferreting through the INIT with a debugger, then make sure that you keep INIT-Scope turned on! Otherwise the addresses and data will not be the same the second time through as they were in the report.

Beware the Debugger Patches
If you use INIT-Scope's intercepting capability to drop into a debugger just before an INIT is called, you must be prepared for two things; first, there will be no trap history provided for this INIT (the trap history option is explained in the next section), and secondly you will likely see some unusual patches that have nothing to do with the INIT, but are in fact caused by the debugger. For example, in the case of TMON, you may see something like:

```
$2F  Patch:    $A3B2  PostEvent
$1C9  Patch:    $A3AC  SysError
Low Memory Vector at   $8  altered to: $1EE0C4
```

Note that TMON must patch vectors such as $8 to be able to intercept system errors when they occur.

Modifying INIT-Scope

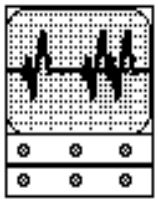
Changing the Report's Creator Type
The *Startup Report* is a text file, and consequently any word processor can read it. However, it is convenient to be able to open the file directly from the Finder by double-clicking it. For this reason, the report has been given the creator type of 'MSWD' so that you can open it directly if you own Microsoft Word. If you use another word processor, then you might prefer to change the creator type to something else—like MACA if you use MacWrite. You can do this by using ResEdit to change INIT-Scope's 'Fcrt' resource (this resource is a long word representing the creator type of the report, and is by default MSWD).

Changing the HOOK Resource
INIT-Scope contains a special resource of type 'HOOK'. There is a template for this resource included in INIT-Scope's resource file. It is a fairly simple matter to alter this resource, and its structure is pretty self-explanatory. You can edit the 'HOOK' resource and add or delete the names and addresses of low-memory globals that you want to check.

> **Note**: Do not distribute any version of INIT-Scope in which you have changed this or any other resource.

Virtual Memory

This version of INIT-Scope will not provide an accurate report if Virtual Memory is on. It will run under virtual memory ,but will not provide the trap history and patch data portions of the report. So if you want a complete report, reboot with virtual memory temporarily turned off. Future versions of INIT-Scope will avoid this problem.



Final Comments

INIT-Scope has been tested extensively on a Mac Plus, Mac SE, Mac SE/30, Mac II, Mac IIcx, PowerBook 170, LCII, Macintosh IIsi, and a Mac IIci. In addition, a great many INITs were tested with INIT-Scope. The only debuggers that INIT-Scope has been tested with are TMON and MacsBug. Also INIT-Scope has not been tested with Systems below 6.0.

INIT-Scope's INIT is written entirely in assembler. Although it provides you with a great deal of information about the patches to the toolbox traps, INIT-Scope *does not patch any traps itself*.

Future Versions

Your comments, suggestions are welcome and indeed solicited. INIT-Scope will be continually improved, and your suggestions for further enhancements or (gasp!) bug reports will help.

Also, please make sure that any distributed copy has the name of the file as " INIT-Scope" (with a leading space), and I'd prefer that the 'Fcrt' resource remain 'MSWD' (so that the report will be a Microsoft Word file). Similarly, do not distribute any copy of INIT-Scope in which the 'HOOK' resource or 'GloB' resource has been altered. (The 'GloB' Resource should be $FFFFFFFF in any distributed version of INIT-Scope.)

If you decide to keep INIT-Scope, send $15 to David Sumner at one of the addresses below. Technical help and upgrade notices will be provided to those who

register.

David P. Sumner
1009 Walters Lane
Columbia, SC 29208
(803)-783-2980

Dept of Mathematics
University of South Carolina
Columbia, SC 29208
(803)-777-3976

I can also be reached electronically at any of these addresses:

Compuserve: 75515,1507
America Online: David Sumn
Internet: sumner@milo.math.scarolina.edu

Addendum
Changes to INIT-Scope 1.0

1. INIT-Scope 2.0 is system 7.0 compatible.
2. All new traps up to 1992 are recognized.
3. Information is provided for more special traps (such as gestalt).
4. Additional information is provided about the initial startup status of the machine.
5. INIT Intercepting is better behaved and much more convenient to use.
6. All low-memory hooks are monitored regularly.
7. The user can modify the low-memory globals that are checked by INIT-Scope.
8. The format of the output file has been improved.
9. Notification requests and associated strings are included in the report.
10. Handles returned to resource traps are displayed in trap history.
11. The user may optionally observe the changes to a particular global variable after every trap call.
12. The handle to each INIT is supplied.
13. A rudimentary measure of system heap fragmentation is provided with the report on each INIT.