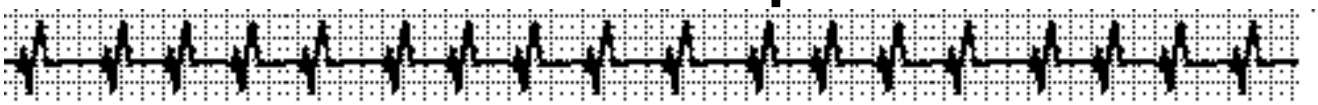
#p

INIT-Scope



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 a few useful utilities.

Who Can Use INIT-Scope?

Basically, INIT-Scope is a highly technical tool, and is of most use to a skilled programmer. However, it is still of considerable value to others as well.

1. Even if you do not understand everything in the INITInfo report that INIT-Scope produces, you will most likely still be able to get some idea of what your INITs 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 INITInfo report, (s)he can help you figure out any kinds of problems that you might have.

In fact, if you are having a problem, and can't make sense out of the data that INIT-Scope provides, send me a copy of the report by E-Mail, and I will try and help you decipher it.

Using INIT-Scope

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

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.)

The INITInfo Report

INIT-Scope produces a text file report called "INIT Info". 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 start-up process. This includes each resource type and ID.

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

8. The system heap expansion caused by the INIT.

9. 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.

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

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

In addition to all of this information, INIT-Scope also makes using INITs easier. By setting an appropriate option in the associated cdev (via the Control Panel desk accessory), you can skip any INIT by holding down the Shift key just before it would ordinarily load (i.e. hold down the Shift key just after you see the icon for the previous INIT).

If you are technically inclined, INIT-Scope will also let you intercept any INIT by holding down the option key at the time the INIT is about to load.

Format of the INIT Info File

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

An example such file begins like this:

INIT-Scope’s INIT Info Report
INIT-Scope is Copyright 1990 by David P. Sumner

Date: 5/22/90
Time: 9:46 AM

All addresses are in hex.

All sizes, quantities, and id's are in decimal.

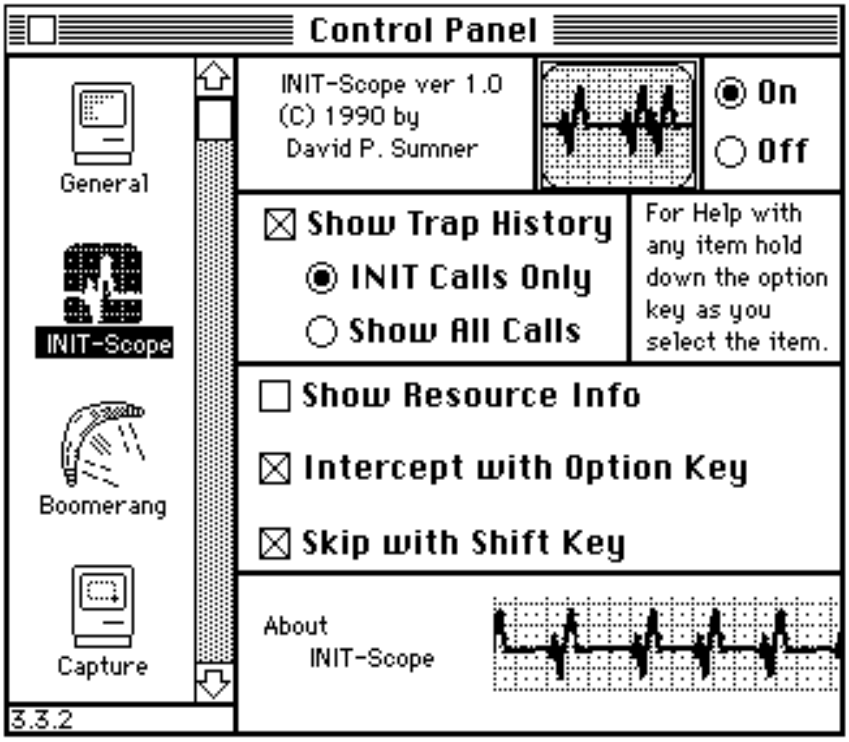
Computer: Mac Plus
Processor: 68000
Does not have MC 68881 Floating Point Coprocessor
Does not have Color QuickDraw
Mac Plus Keyboard
System Volume RefNum is: -32733
BufPtr: $E2DFA
APPL Zone $21400
Heap End: $253F4
AppLimit $7DBFC
System Zone $1400
Free in System Heap (bytes): 18580
Top of Memory: $100000 (1048576 Total Bytes of Ram).
Screen Base: $FA700
Sound Buffer: $FFD00
System Version: $602
Currently Active VBL Procedures: $7D20 $15EA8

After this system information, the INITInfo file provides detailed information about each INIT in turn as it is loaded in 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 rest of the file looks like will depend upon the options you select in the INIT-Scope cdev in the Control Panel.

#p



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 and much more.

An example of this portion of the INIT Info file is:

```
Res Type  STR
Res Type  SHRW
Res Type  PROC
   $1EA  Patch:    $2681A  Pack3 (StdFile)
Res Type  PROC
Res Type  Data
Res Type  ICN#
$A02F  Patch:    $2B512  PostEvent
$A970  Patch:    $2B502  GetNextEvent
$A971  Patch:    $2B50A  EventAvail

jGNEFilter (GetNextEvent Patch) installed at:    $332E0
```

This would tell you that this INIT loaded a resource of type STR , then one of type SHRW, then a PROC resource.Then it patched the toolbox trap Pack3 with a routine of its own at address $2681A (the ‘$’ indicates a hex value). After this, the INIT loaded in resources of types PROC, data, and an ICN#. It then patched the traps PostEvent (at $2B512), and EventAvail (at $2B50A). We note next that the INIT effectively patches the GetNextEvent toolbox trap as well by setting the jGNEFilter hook to the address of a routine installed in the system heap by the INIT.

There are a great variety of other kinds of information that this portion of the INIT Info 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 patches the traps Eject and PostEvent.

The INIT Info report can provide much more information than what we have indicated here. The sections that follow will elaborate on this.

Example:
Here is an example of the report generated for a couple of simple INITs

Easy Access -- Note that this file contains three distinct INITs

```
Easy Access  ( INIT)
Size of this INIT in bytes: 832
Requested System Heap space (bytes): 0
System Heap Expansion (bytes): 0
System Heap Used (bytes): 632
Application Heap:    $40298 -   $44A98
Free in System Heap (bytes): 16748

   $137  Patch:    $2AFEC  DrawMenuBar

jGNEFilter (GetNextEvent Patch) installed at:    $2AFBE


Trap History
  $A51E  NewPtr   (Size (bytes): 776  )
  $A02E  BlockMove  #Bytes: 776  From:    $40414  To:    $2AF68


Easy Access  ( INIT)
Size of this INIT in bytes: 572
Requested System Heap space (bytes): 0
System Heap Expansion (bytes): 612
System Heap Used (bytes): 468
Application Heap:    $404FC -   $41CFC
Free in System Heap (bytes): 16892

    $2F  Patch:    $2B26A  PostEvent

Trap History
  $A51E  NewPtr   (Size (bytes): 532  )
  $A02E  BlockMove  #Bytes: 532  From:    $40754  To:    $2B1D0
```

LaserFixLaserFix (INIT)
Size of this INIT in bytes: 1096
BufPtr: $1E7272
High Ram Used (bytes): 388
Requested System Heap space (bytes): 0
System Heap Expansion (bytes): 468
System Heap Used (bytes): 4
Application Heap: $406D0 - $41ED0
Free in System Heap (bytes): 17356

$8 Patch: $1E7272 Create
$0 Patch: $1E72F6 Open

Trap History
$A997 OpenResFile
$A9A0 GetResource DITL #-8191
$A86E InitGraf
$A86F OpenPort
$A9A0 GetResource ICN# #128
$A029 HLock $407D8
$A8EC CopyBits
$A8EC CopyBits
$A02A HUnLock $407D8
$A9A3 ReleaseResource $407D8
$A87D ClosePort
$A146 GetTrapAddress
$A146 GetTrapAddress
$A02E BlockMove #Bytes: 388 From: $40AD2 To: $1E7272
$A047 SetTrapAddress
$A047 SetTrapAddress

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* [Begin Footnote] ---If you haven't already tried this neat INIT, get it off of just about any online service.--- [End Footnote] (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 INIT Info 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
$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**
```

Now, the two asterisks are INIT-Scope's way of 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.

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*.

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 trap that is central to every Macintosh application. This is effectively a patch to the GetNextEvent trap, but it does not appear as such.

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. We'll have more to say about debuggers in a minute.

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 INIT Info 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 <u>all</u> 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 INITInfo report is:

```
Trap History
          .
          .
```

```
          .
$A9A0  GetResource   BrtZ  #128
$A51E  NewPtr   (Size (bytes): 512  )
$A992  DetatchResource  (   $1D544  )
$A9A0  GetResource   ICN#  #128
$A86E  InitGraf
$A86F  OpenPort
$A029  HLock     $483A0
$A8EC  CopyBits
$A8EC  CopyBits
$A02A  HUnLock     $483A0
$A9A3  ReleaseResource  (   $483A0  )
$A87D  ClosePort
$A146  GetTrapAddress
$A02E  BlockMove  #Bytes: 68  From: $4854C  To: $1D847C
$A047  SetTrapAddress
          .
          .
```

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.

4. NewHandle and NewPtr. In each case the size of the requested handle or pointer is shown.

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

6. Pack Managers are displayed by name.

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 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.

Changing the Report's Creator Type

The INIT Info 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 your 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).

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 (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 2 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.

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 will 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 cdev. After this, INIT-Scope will not load again until you turn it back on from the cdev.

If INIT-Scope is on, but you would like to bypass it in the startup process, just lock down the Caps Lock key but do not depress any other modifer keys.

#p

For the Highly Technical - Using INIT-Scope and a Debugger

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 the instruction

```
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.

It is usually better to load the debugger before INIT-Scope. It will work the other way around, but it's not recommended.

One last thing, you may find it useful to know that the name of the INIT is stored at the location $16 bytes (22 decimal) past the address in the program counter at the moment you fall into the debugger.

Final Comments

INIT-Scope has been tested extensively on a Mac Plus, Mac SE, Mac SE/30, Mac II, Mac IIcx, and a Mac IIci. In addition, a great many INITs were tested with INIT-Scope. The only debugger that INIT-Scope has been test with is TMON. 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!

Version 2.0

Your comments, suggestions are welcome and indeed solicited. A far more powerful version of INIT-Scope is in the works and will be out shortly. If you would like to be informed of the release of version 2.0, send me an E-Mail.

#p

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

CIS: 75515,1507
America Online: David Sumn
Internet: sumner@sc.scarolina.edu