# Extension Shell

**Extension Shell 1.3**

Extension Shell is an 'INIT' code resource that minimises the amount of rewriting that you, as a programmer, will have to do when creating Extensions. If you're not a programmer, Extension Shell can't do anything for you.

Extension Shell contains all the common code that your Extensions may need, and installs only what you require in memory. It provides the leverage your code will need to get called at start-up, correctly display an icon, animate a sequence of icons, post notification messages to the user, install Gestalt selectors, install Shutdown tasks, install VBL tasks, install low memory filters (such as jGNEFilter), leave blocks of code in the System Heap, install Time Manager tasks, and patch out traps with your own routines. It requires System 7. It's 68K based for now, largely because I don't have any experience/access to PPC tools.

This archive contains both a compiled version, and full source code. Extension Shell requires THINK C 6.0. If you use another development environment, please read the "If you don't have THINK..." file for details of the project settings.

**How and Why?**

For a more detailed explanation, read 'Extension Shell Source ReadMe' (in the :Extension Shell 1.3 (Source) folder) and 'Generic ES Handler ReadMe' (in the :Sample Extensions:Generic ES Handler *ƒ* folder). These cover the main principles behind Extension Shell. Then take a look at 'ExtensionShell.c' and 'ES Handler.c' (in the same folders). These should give you a feel as to what goes on at start-up.

Basically, the process goes like this:

- Extension Shell starts up, and initialises itself.
- Extension Shell calls a code resource you provide (the 'ES Handler' ). This code resource decides what should be installed, what icons to show, etc.

- Extension Shell does what your ES Handler requests. The Handler may be called again to do any further initialisation, if requested.

- Extension Shell cleans up, and exits.

To get a better feel, browse through the rest of the source code. Take a look at the sample Extensions, and compare their ES Handler code with the Generic ES Handler.

If you're just starting out with Extensions, you may find it a bit overwhelming. Don't panic. Even if you decide not to use Extension Shell, hopefully the source code will help explain some of the 'black-art' of writing INIT code. It's all well commented, and should be pretty clear. If you're not sure of the terminology of stand-alone code resources, there are the locations of some useful documents at the end of this file. Do try and get hold of them.

If you've written Extensions before, you'll probably have your own template - and the chances are Extension Shell won't teach you anything new.

Either way, the main advantage of Extension Shell is its modularity. Having written and debugged one module (e.g., a trap patch), you can move on and develop another routine. Since each 'thing' (trap patch, gestalt selector, etc.) is a separate code resource, it's a lot easier just to paste in an update when need be, without having to recompile the whole Extension. It also means that only the minimum is left in memory - you won't have to include code for drawing icons, or posting notification messages. OK, so there is the overhead of Extension Shell itself - particularly as it's in C, not assembly. But it's only a few bytes over 4K, and it's a lot more flexible. Hopefully, you'll get some mileage out of it.

Don't worry if things look too complex - they're actually quite straightforward. The best way to start is to take the sample Extension that's closest to what you're trying to write, and clone it. Change the #defines to suit your details, and compile it up.

**What makes up an Extension Shell Extension?**

At least seven resources. Three of them are supplied by the generic Extension Shell, and the others are supplied, where needed, by you. The Extension Shell resources are generic, and hold the routines to get things done. The code resources you supply are responsible for the Extension specific details.

Specifically, the three generic Extension Shell resources are:

- An 'INIT' resource that is executed by the System at boot time. This is the heart of Extension Shell. It contains all the routines to do the actual work. This is the that part that calls your code.

- A 'CODE' resource for destroying Notification Manager messages. This is optional, but the advantage is that the messages will remove themselves from the System Heap. If this resource is missing, the message is stuck there until the next reboot. This is the CODE 5001, ‘RESP Code’, resource.

- A 'CODE' resource for a Gestalt Selector. The actual selector you pass to Gestalt is defined by your ES Handler code resource, but this piece of code is the default Address Table’ that Extension Shell provides. This can be replaced by a routine of your own, provided it can interface with Extension Shell correctly - see ‘Extension Shell Source ReadMe’ for more details. This is the CODE 5002, ‘Address Table’, resource.

The sample Extensions have all three of these resources. Compiled versions are also included in the Extension Shell 1.3 (Source) folder, in the ResEdit file ‘Extension Shell’. The source code for these three resources is also in the same folder.

The four+ resources you should provide are:

- An ES Handler 'CODE' resource. The Extension Shell 'INIT' code resource loads this resource into memory and calls it to find out what it is to do. The ES Handler decides what should be installed, what icons to show, if any Notification Manager messages should be posted, etc., etc. It must be the CODE 5000, ‘ES Handler’, resource.

- A 'STR#' resource. When your ES Handler has decided that Extension Shell should not try and install your code, it can pass the resource ID and index of the relevant 'STR#' resource. This string will be displayed as a Notification Manager message to the user when the Finder has started up. If the ‘RESP Code’ resource (see above) was present in the Extension, the message will unload itself from memory automatically.

- An icon family - with at least 'ICN#', 'icl4', and 'icl8' icons. Your Extension will probably need at least one icon to indicate a successful installation, and one for a failure, however Extension Shell also supports the display of an animated sequence of icons. Your ES Handler should pass the resource ID of the relevant family to Extension Shell.

- Further 'CODE' resources. These resources are the ‘installables’ of your Extension. Your ES Handler fills in a template for Extension Shell, specifying their resource type and IDs, as well as what they are (a trap patch to DrawMenuBar, a Gestalt selector of type 'Moof', etc.). Extension Shell then loads these resources in, and installs them correctly. In this way, all the patches are kept separate from the rest of the Extension, making your Extension more modular and faster to develop.

**What can it do?**

Extension Shell can help your Extensions install the following things:

- **Trap Patches**. Trap patches replace System code with your own routines. Your routines could call the original System routines, or override their behaviour completely. For example, a patch to DragWindow could drag the actual window around the screen, rather than an dotted outline.

- **Gestalt Selectors**. Gestalt selectors can be accessed through the Gestalt call. They can return a long's worth of information. System Gestalts commonly return information about the Macintosh. 3rd party selectors are often a handle/pointer to a structure in memory, which contains things like status flags, strings, pointers, handles, etc.

- **Shutdown Tasks**. Shutdown tasks are queued until the Macintosh is shutdown, or restarted. When a shutdown or restart occurs, the tasks are executed.

- **VBL Tasks**. VBL tasks are executed during the 'old style' VBL interrupt, that are not tied to a particular monitor's refresh rate. They execute at interrupt time, so are limited in what they can do.

- **Low Memory Global Filters**. Some low memory global variables are used as entry points to a chain of procedures. By hooking into this chain, your code will be called whenever the chain is activated. The most commonly used low-memory filter is jGNEFilter. This processes events before application's receive them.

- **Blocks of code**. If you are installing several things (e.g., two trap patches and a jGNEFilter), you may find yourself duplicating code. By putting the common code into its own code resource, you can arrange to have it loaded and locked into the System Heap by Extension Shell. Your other routines can then share this code to do their work.

- **Time Manager Tasks**. Time Manager tasks are executed periodically by the Time Manager. They execute at interrupt time, and can not call anything that might move memory.

To see how to install an example of each type, see the sample Extensions. Make sure you read 'ParamBlock.h' (in the :Extension Shell 1.3 (Source):Extension Shell #includes folder). This is the file that holds the information your ES Handler uses to let Extension Shell know what to install.

**Legal Stuff**

Extension Shell is freeware. Basically, it tries to tie together some of the code/ideas related to INIT writing that have been floating around for years, and combine them in an Extension-independent way. All the techniques it makes use of are, to the best of my knowledge, public domain or freeware themselves. However, Extension Shell is still Copyrighted © 1993-1994, Dair Grant. You may not redistribute it in any modified form.

You may not charge for Extension Shell itself, or redistribute it as part of a commercial package without my prior permission. Shareware houses, BBS Sysops, or CD-ROM Distributors may include Extension Shell in their collections, on the understanding that they do not charge for access to Extension Shell specifically. I would appreciate a copy of any CD-ROM that Extension Shell appears on.

Extension Shell may be used in any extension (freeware, shareware, or commercial) without charge.

**General Information**

Extension Shell is compiled with THINK C 6.0. The .c and .h files are all BBEdit files. If you don't use an external editor with THINK C 6.0, you will need to toggle off the 'Uses external editor' check-box (under Edit:Options:THINK Project Manager...:Editor) if you want THINK C 6.0to open the source files. You'll also need to use something like File Buddy to turn the source back into THINK C files.

If you don't have THINK C 6.0, the "If you don't have THINK..." file contains a description of each project's settings. You should be able to get things up and running using that.

The source files have been divided into two types. Information Files are labelled 'Cool', set in Geneva 9, and should be read before compiling the contents of a folder. Source code is labelled 'None', set in Monaco 9, and holds the source code for whatever is in the folder.

**Contacting the Author, and Credits...**

Please don't redistribute modified copies of the Extension Shell package - if you've got an idea that you would like incorporated into Extension Shell, email me and we can work it into the next release. My email address will be changing during the summer of 1994 after I

graduate and (attempt to) enter the world of gainful employment. For now, you can reach me as grantd@dcs.gla.ac.uk. My postal address, which should last a bit longer, is:

Dair Grant,
11 Garrioch Quadrant,
North Kelvinside,
Glasgow, G20 8RT
Scotland.

I wrote Extension Shell when I was starting to get into writing Mac INITs. I found it quite hard to get hold of samples that I could use to compare with my code, or with documentation explaining exactly what was going on. Basically, I was spending most of my time hacking around with old code, trying to convert it into a new INIT by osmosis. :-( The code loaders I found didn't seem very flexible, and they were all a bit single-minded (e.g., no support for icons, or only installed trap patches) in what they did. Hopefully, Extension Shell is about as flexible a code loader as it can be.

I do plan to support this code. If you have any problems, or see any errors in the code, let me know, and the source will be updated. If you don't like how something is implemented, and can think of a better way - again, let me know. Obviously, bug reports are more than welcome. :-)

Thanks to the people who helped with beta testing, sample code, and suggestions for Extension Shell, namely: Ray Arachelian, Jonathan D Baumgartner, Jordan Pallack, Mark Pilgrim, Felix Potter, Johnathon Suker, and James Thomson.

**Further Reading**

Although Extension Shell makes it relatively easy to get some code installed at start-up, you might like to read some of these if you want some more information on what's going on. If you're just starting with writing Extensions, please take a look at these documents before getting stuck in to Extension Shell. If you notice that any of these site names/paths are out of date, please let me know.

If you're just starting off with stand-alone code, **please** read the "Standalone Code, ad nauseam" Tech. Note - it's a good summary of what you'll need to know.

"Code Resources (Inits, Cdevs, VBLs, ...)", Usenet Macintosh Programmer's Guide
<sumex-aim.stanford.edu>
/info-mac/dev/info/usenet-mac-prog-guide-msw.hqx

"DeskHook and INIT Evils", Pete Helme
New Technical Notes
<ftp.apple.com>
/dts/mac/tn/operating.system.os/os-02-deskhook-and-init.hqx

"Standalone Code, ad nauseam", Craig Prouse, Keith Rollin, Keithen Hayenga
New Technical Notes
<ftp.apple.com>
/dts/mac/tn/platforms.tools.pt/pt-35-stand-alone-code.hqx

"Another take on Globals in Standalone Code", Keith Rollin
develop #12
<ftp.apple.com>
/dts/mac/docs/develop/develop.12.code/globals-in-standalone-code.hqx

"The Mac Extended", Eric Shapiro & Tom Thompson
BYTE, July 1993
<ftp.uu.net>
/published/byte/93jul/belltest.sea
/published/byte/93jul/keytest.sea

"C.S.M.P Digest" archives
<ftp.cs.uoregon.edu>
/pub/mac/csmp-digest
<ftp.dartmouth.edu>
/pub/csmp-digest