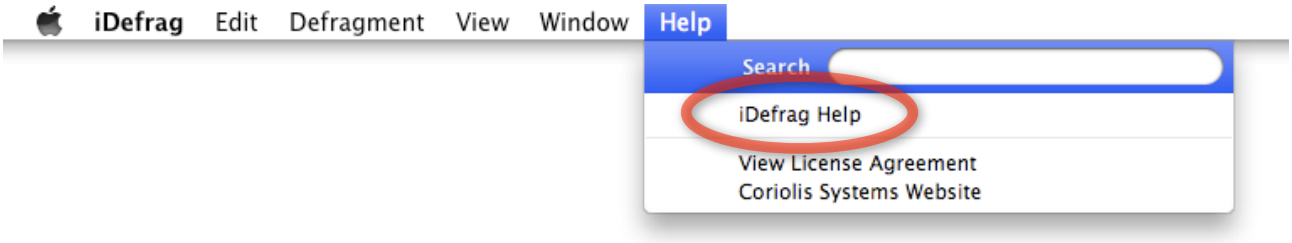
iDefrag

Quick Start Guide

coriolis

Introduction

This guide provides a brief overview of iDefrag but does not cover every feature of iDefrag. The help within iDefrag provides a more comprehensive reference guide.



If, having read this guide and looked in the help, you still can't figure out how to use the product to achieve a particular goal, please contact our support team who will be happy to assist.

Contents

Before you start

Before letting iDefrag go to work, you should back up any important data that you have.

While we don't know of any bugs in iDefrag that could cause data loss, defragmenting is inherently risky and the sustained access to the disk that you get whilst defragmenting can aggravate existing hardware problems with your disk.

You should also check your filesystem for errors. To do this, you can either select Check for Errors from the Defragment menu, or you can use Disk Utility; they both perform exactly the same checks. If you use Disk Utility it's the "Verify Disk" button that you need to select, *not* the "Verify Permissions" button.

If you don't keep regular back-ups, your data is at risk *even without* using disk utility software.

Disks fail all the time; as a mechanical component they are one of the most unreliable parts of your computer, and given that you are probably storing all of your photos, music and maybe even home movies on your machine it makes sense to keep your data safe.

Fortunately, Apple provides an easy-to-use back-up system called Time Machine that you can use to keep your data safe. More advanced users may prefer other software, but for many people Time Machine provides set-and-forget back-up and peace of mind.

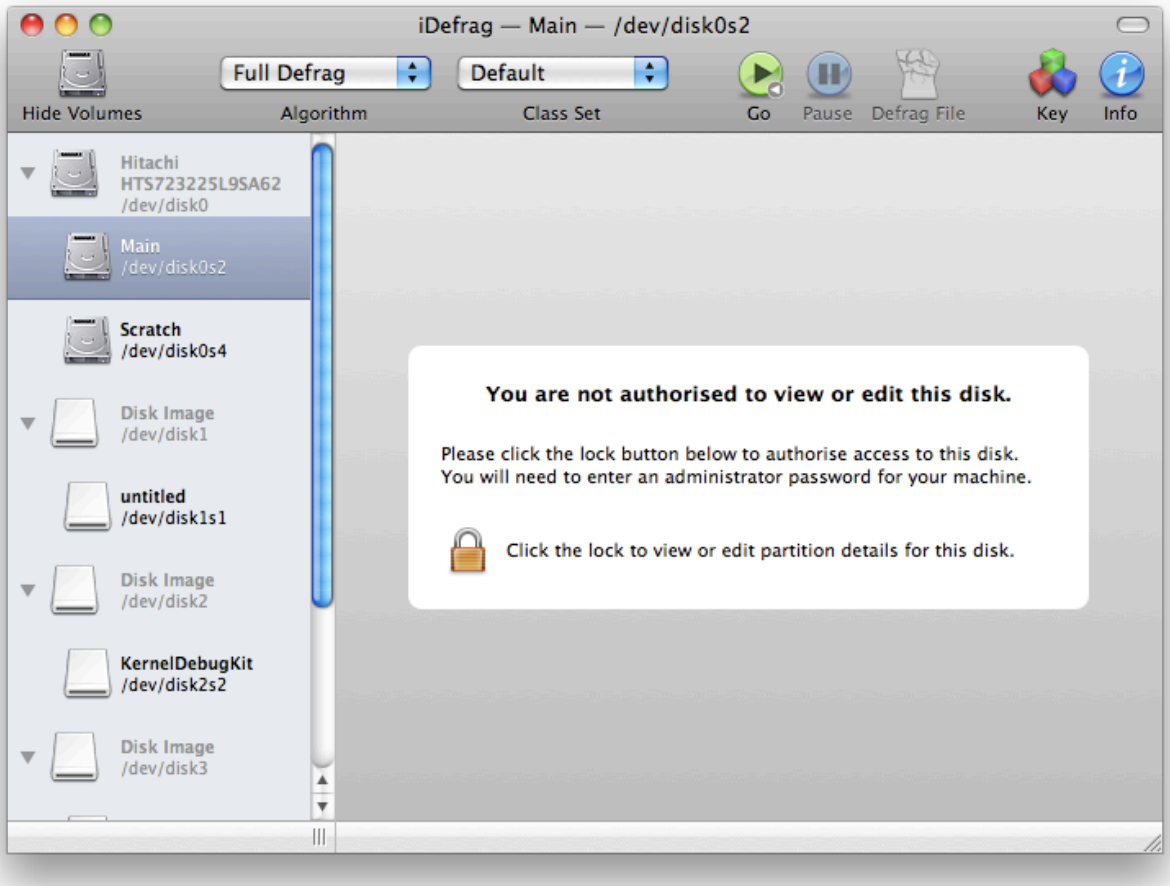


Figure 1 The computer is asking for your permission to access this disk.

Authorization

Mac OS X has built-in security features to prevent applications (and other users) from accessing the disk directly. As a result, iDefrag sometimes needs to ask you for permission when you select a disk. When this happens, you'll see a gold colored padlock like that in Figure 1, above.

Click on the lock button and then enter *your OS X username and password* when prompted. It is *not* asking you for the username and password that you might have been given when you purchased iDefrag.

Whenever you see a gold padlock, your computer is asking for your permission to perform some operation.

You should read the window being displayed on the screen carefully to understand which program is asking for permission and why, and only enter your OS X administrator log-in details if you are sure you know what will happen.

Main Display

In Figure 2, opposite, you can see the major features of iDefrag's main window:

1. Using this pop-up list, you can choose which of iDefrag's algorithms you want to run. Algorithms are discussed below.

2. When you're ready, click the Go button to start defragmenting. You may be able to make out a small gray circle containing a white left-pointing arrow at the bottom corner of the Go button. This is the restart badge, and indicates that iDefrag will need to restart your computer to defragment this disk; some disks can be defragmented without a restart, and in that case the badge will not appear.

3. Each square in this area of the window represents a single block in your file-system. The colors indicate what kind (or *class*) of file is using the block — for example, green is used to represent Applications. Note that iDefrag shades adjacent runs of blocks of the same color in an alternating light/dark pattern, so you can see the ends of your files or fragments. Anything that's red is fragmented.

   If the default colors are not to your taste, you can change them from the Key panel

   Your custom colors will be saved in iDefrag's preferences for future sessions. Just like the default set, each color you set will be drawn in a darker and a lighter version so that you can see individual fragments on your disk.

4. This part of the window shows a representation of the entire disk. The white parts show what is free space. In this particular example, you can see that there are a lot of fragmented files (they're red) and there's not much free space (white).

5. At the bottom you'll see some tabs that you can select that will show you more information about the volume. You'll find more information about the other tabs in the Help.

6. On the left, you'll see a list of volumes that you can choose from. iDefrag will only list volumes that it supports: the Mac filesystem format (HFS+); not Windows formats such as FAT or NTFS.

   Many external drives come pre-formatted with FAT

   If you purchase an external disk, it will probably have been formatted for use with a PC.

   In order to provide the best performance, and to allow the use of all of the features of Mac OS X, we recommend re-formatting external disks before use with the Mac OS Extended (journaled) option in Disk Utility.

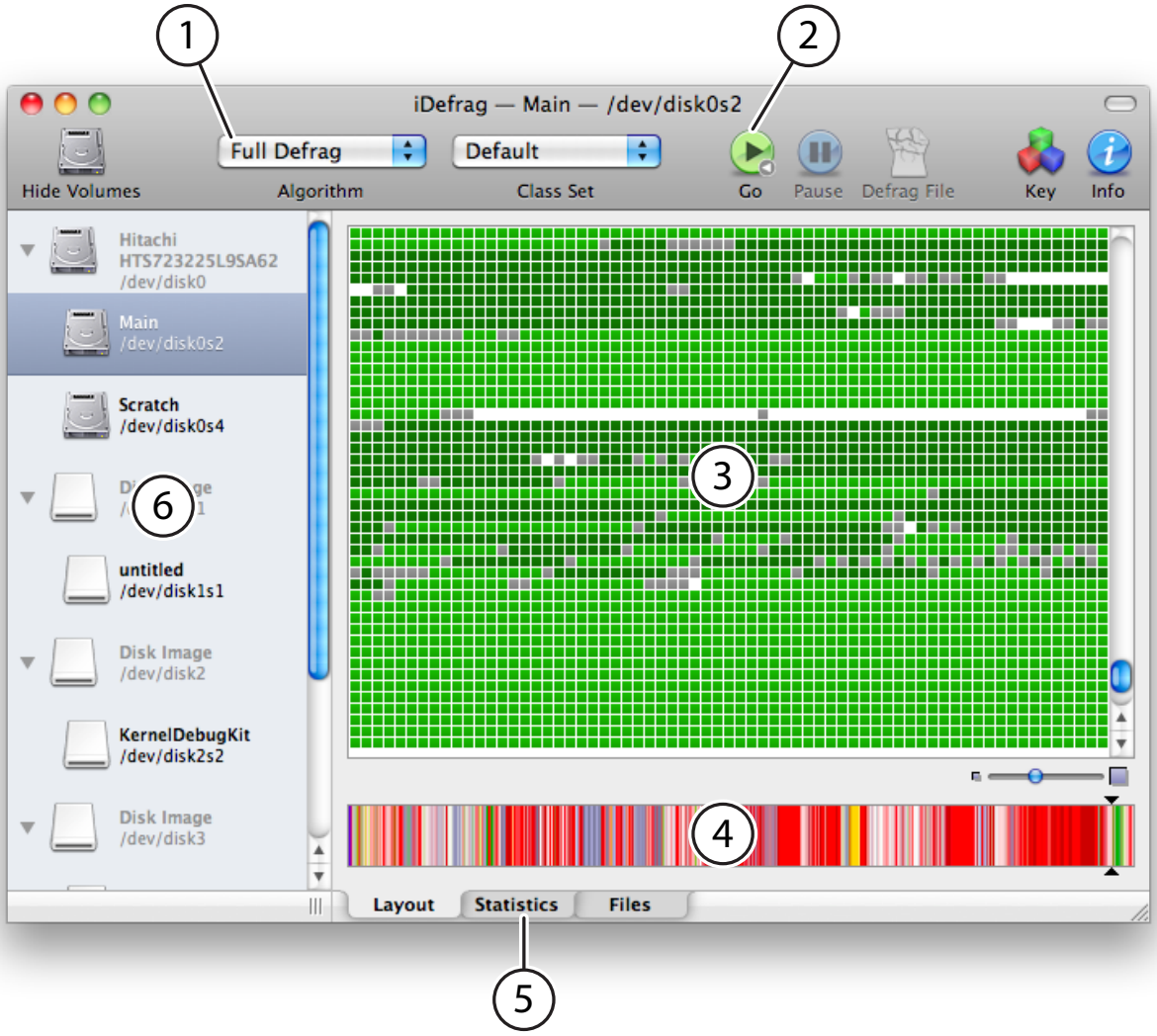


Figure 2 iDefrag's main window

What does the Class Set pop-up do?

Advanced users can create new categories (or *classes*) of file and can specify what iDefrag is supposed to do with them, by defining their own classes in a "class set" file.

You can find the details in the iDefrag Help.

Defragmenting your Startup Volume

Except when using the Quick (on-line) algorithm, iDefrag needs exclusive access to the volume you want to defragment.

If other programs are using the disk, it may not be possible to gain exclusive access. Sometimes you may be able to terminate other programs to resolve this problem, but for the start-up volume the operating system itself is using the disk and so you will need to use one of the options listed below.

Note also that background software such as anti-virus programs can prevent you from giving iDefrag exclusive access. The options below may be useful in that case also.

If you just want to have a play with iDefrag first without actually defragmenting, skip this section and come back to it later.

To defragment your startup volume, you can try one of the following:

- You can let iDefrag restart into a mode where it has exclusive access to your volume. iDefrag will offer to do this when necessary. You will need about 1 GB of free disk space and you will not be able to run any other programs whilst iDefrag is running.
- You can create a bootable DVD and use that. To do that, select Create Boot Disk from the iDefrag menu and follow the instructions. You will need a DVD burner and some recordable media. Note that booting from a DVD can take considerably longer to boot (several minutes) than your normal startup volume.
- If you have another Mac, you can use Target Disk Mode. See this page on Apple's support site for instructions about how to do this: http://support.apple.com/kb/HT1661.

Some older PowerPC (G4) Macs have bugs in their implementation of FireWire Target Disk Mode.

On affected machines, the FireWire logo may stop moving from place to place on the display; this may also cause iDefrag to display the beach ball cursor.

We recommend that you use a different method for older PowerPC systems.

- If you have an external drive or you have partitioned your main hard drive, you can install OS X on it and boot from that. We do not discuss the details of doing that here but you should be able to find guides on the Internet if you are interested in doing this.

Algorithms

iDefrag comes with a number of different algorithms and one of the first things that most people ask is which one should be used. In this version of iDefrag, the available algorithms are:

Full Defrag

If you don't want to think about which algorithm to choose, *choose the Full Defrag algorithm* (which is actually a combination of the Metadata and Optimize algorithms). The Full Defrag algorithm is the most comprehensive of all the algorithms and will be the right thing for most people. After running this algorithm to completion, all your files should be defragmented and in an optimum position on your disk.

Compact

This algorithm compacts your free space but won't defragment files. It is much faster than the Full Defrag algorithm. Use this algorithm if you're having trouble with the Boot Camp assistant or if you don't have the time to run the full defrag option. Even though it doesn't defragment files, it can still be quite effective because it can re-enable OS X's built-in defragmentation (which will stop working when your free space is too fragmented).

Quick (on-line)

This algorithm is useful if iDefrag is unable to get exclusive access to the volume that you want to defragment and you don't have the time to reboot. This algorithm will only work on files that are not currently in use (which rules out all system files).

Optimize & Metadata

Please see the Help for detailed descriptions of these algorithms. Unless you have a specific reason not to, you should use the Full Defrag algorithm which is a combination of these two algorithms.

Figure 3 A heavily fragmented disk

How can I tell if my disk needs defragmenting?

iDefrag offers a number of ways of measuring this but perhaps the simplest way is to look at the representation of the whole disk at the bottom of your screen. You can see a screen shot of a heavily fragmented disk in Figure 3, above.

Note the lack of big chunks of white, indicating free space, and a fair bit of red coloring, showing fragmented files.

The other option is to look at the Statistics tab. Here you'll see a lot of information regarding the state of your volume. Please see iDefrag's built-in help to find out what exactly the numbers mean. We don't provide advice as to when you should or should not defragment, but if you make a note of the numbers on the statistics tab, you can see how they change over time and before and after defragmenting and make a judgement from that.

What next?

When you're ready to start defragmenting, click the Go button and iDefrag will start. The process can take a long time although it depends on the algorithm. The Full Defrag algorithm can take many hours and we advise you to leave it running over night. You can choose what iDefrag does when it has finished, e.g. turn your machine off—select Preferences from the iDefrag menu.

For the off-line algorithms, you will not be able to access the volume that iDefrag is working on, but you can continue to work with any other volumes that you might have mounted; your options are obviously limited if iDefrag is running from a boot disk.

You can pause defragmentation at any time, but this will not give you access to your volume—it prevents iDefrag from using any processor resources, which might be useful if you want to dedicate your machine's processor to some other task for a while.

Thermal Monitoring

During operation, iDefrag will monitor the temperature of most disks (it will warn you if it can't) and pause if the temperature exceeds thresholds defined in the Preferences. By default, iDefrag is set to stop at 55°C which is appropriate for many disks, but some disks operate at a higher temperatures. If you find iDefrag is frequently stopping, it's worth checking to see what temperature your disk operates at and you can usually find this information by searching the Internet for the model number of your disk. The model number of your disk can be found by using the System Profiler application that comes with OS X.

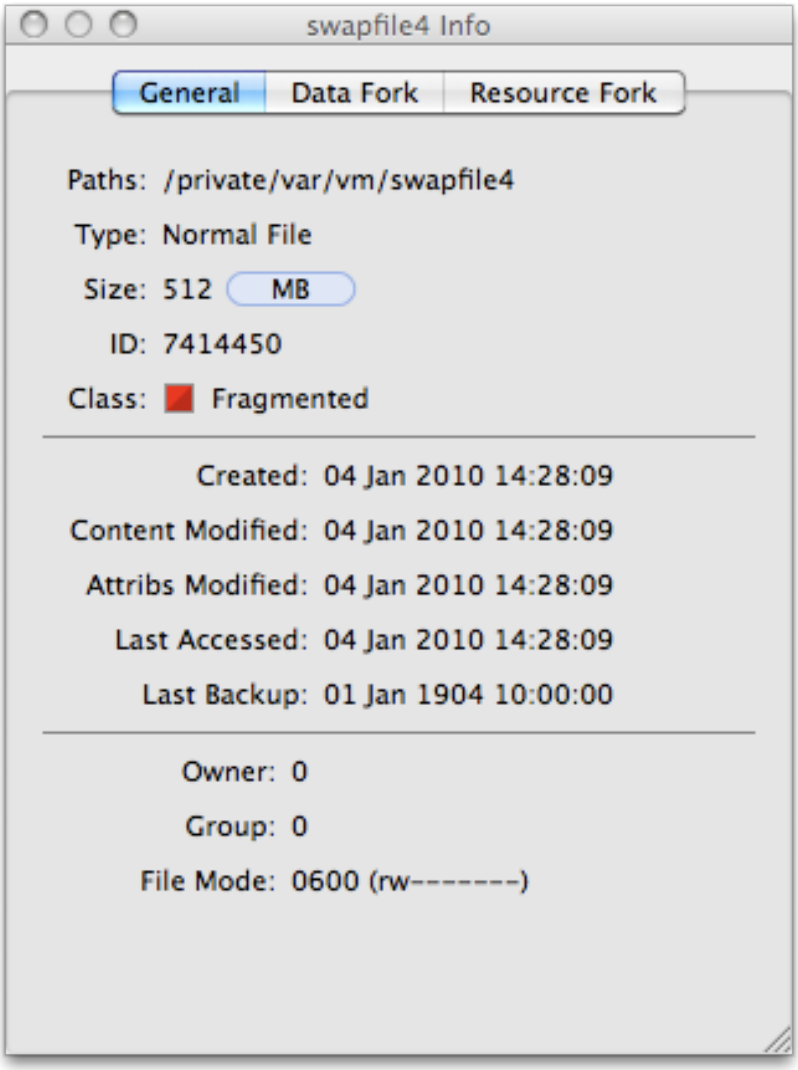


Figure 4 The Info panel

The Info Panel

Click on the Info button on the toolbar to display the Info Panel, shown in Figure 4, above.

The Info panel updates automatically as you hover over the blocks in iDefrag's main window, and will show you detailed information about whatever it is that is currently under your mouse pointer.

If you click on an block, iDefrag will select all of the blocks that make up the file on which you have clicked and the information for this file will be displayed in the Info panel when you aren't hovering over anything else.

If the selected file is fragmented, you can click on the Defrag File button in the toolbar to defragment just this selected file. You won't able to do this for system files or files that are in use by other applications (same as for the on-line algorithm), you'll have to use one of the off-line algorithms to do this.

Troubleshooting

If you are having problems with iDefrag, check the help within iDefrag, particularly the "Troubleshooting" sections. A copy of this help can also be found here:

http://www.coriolis-systems.com/help/iDefrag-2/

Also, check the frequently asked questions page for iDefrag:

http://www.coriolis-systems.com/iDefrag-faq.php

If you still cannot find an answer to your question you can contact our support team:

support@coriolis-systems.com

If you want to report a bug, request a feature or suggest an improvement in iDefrag you can do so via the menu option within iDefrag or you can do it on-line here:

http://www.coriolis-systems.com/bugreport/

Appendix - Filesystems and Disks

Introduction

The information in this appendix is intended to give a non-technical reader some idea about the workings of hard disks and the ways in which computers store files.

You don't need to read this appendix in order to use iDefrag, but it may help to understand what the program is showing you and why you might want to defragment your disk in the first place.

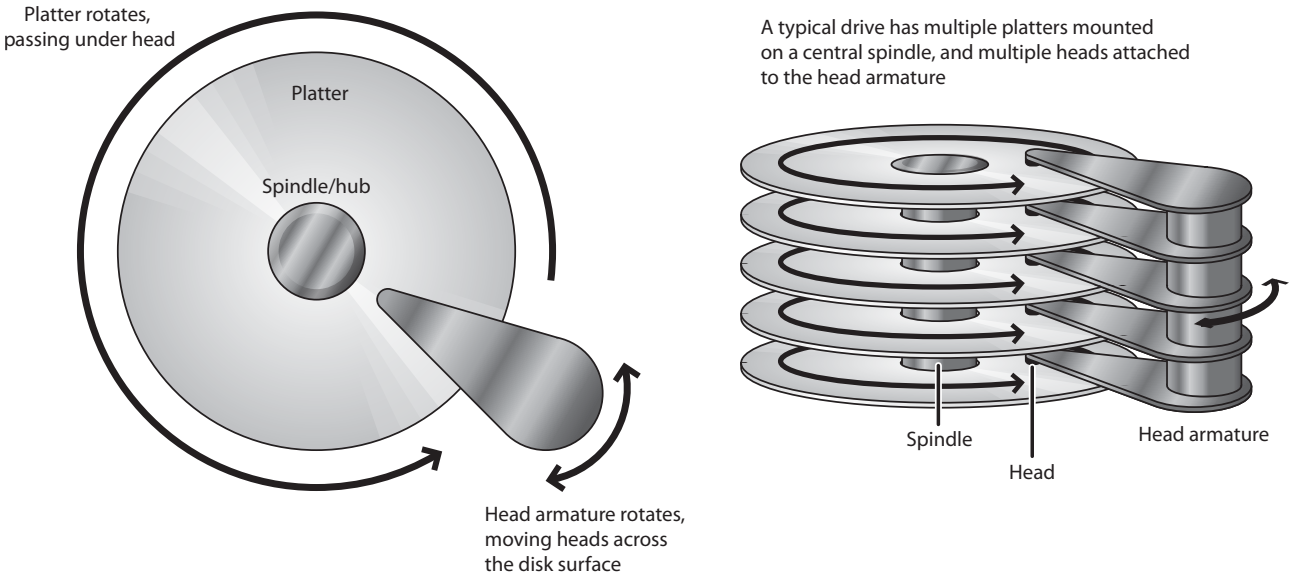


Figure 5 Inside a typical hard disk you will find a set of platters mounted on a rotating spindle, each with one or more disk heads floating just above the surface.

About your hard disk

Everyone is familiar with the notion that a hard disk can store data in *files*, but quite how this is achieved is, for most users, largely a mystery. Indeed, many people these days probably don't even give it a second thought.

All of the explanation that follows is talking about magnetic disks; the details for other storage devices, including tapes, optical discs, solid-state storage devices and so on are different. A magnetic disk, whether a floppy disk or a hard disk, consists of

- one or more circular *platters* (the disk *media*) coated in a magnetic medium,
- one or more disk *heads*, which are able to replay and record a signal on the magnetic surface of the platter,
- a *spindle motor*, which is responsible for spinning the platters at high speed (typically for a hard disk 5,400 rpm or higher), and
- some kind of actuator that can move the disk heads in and out (this might be a stepper motor for floppy disks, or more typically on hard disks a voice coil might be used)

Figure 5 shows a simplified drawing of the inside of a typical hard disk. In practice there would be two heads per platter, one for each surface, the platters and armature would be much closer together, and the disk heads themselves would be very small indeed.

In order to read data from the disk, the disk drive must position its heads at the correct distance from the spindle, then wait for the data to pass under the head for the platter on which the data is stored. The location of the head relative to the spindle is defined by the *cylinder number*. A combination of a cylinder number and a *head number* defines a circular *track* on a particular side of a particular platter. Each track is further divided up into *sectors*, each of which can usually hold 512 bytes of data, and so the combination of a cylinder number, a head number and a sector number identifies a unique sector on the disk's media.

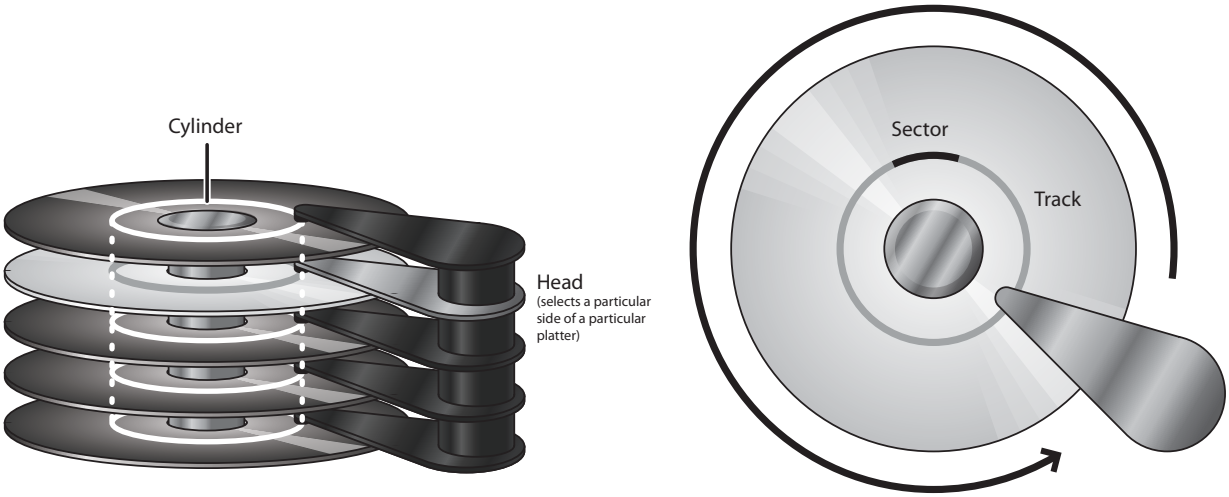


Figure 6 Selecting a particular sector by cylinder, head and sector numbers.

In terms of the mechanics of the disk drive, selecting a head is done electronically, and takes essentially no time at all. The time spent waiting for a particular sector to pass under the disk head obviously depends on the rate of rotation of the drive and the size of the sectors; this is one reason why hard disks that spin at a higher rate are preferable, though the high rate of rotation *also* means that individual bits of data pass under the head faster, resulting in a higher rate of data transfer.

Changing cylinder is comparatively expensive, taking of the order of several milliseconds on a typical modern drive. Moving the heads a single cylinder's width in or out is relatively quick as the positioning mechanisms typically used in modern drives are accurate enough to move one cylinder in either direction reliably. For longer movements, however, the mechanism may not be sufficiently accurate to guarantee that it will move the heads to the correct location; as a result, each track contains information allowing the drive to identify the track over which the head is currently located. If the drive does not move to the correct track the first time, it may need to move again, typically by successively smaller distances, until it finds the right location.

The most expensive movements are normally "full-stroke" movements, where the disk heads are moved from the innermost cylinder to the outermost cylinder or vice-versa.

Figure 6 shows the process of locating a given sector on the disk using cylinder, head and sector numbers.

Older disks, with the exception of SCSI disks, typically expect the host computer to provide them with co-ordinates for data in this form, which is also referred to as CHS or Cylinder Head Sector. Unfortunately this scheme has a number of downsides, notably that different disks will have different numbers of cylinders and heads, and may even choose to have different numbers of sectors on each of their tracks; the arrangement of cylinders, heads and sectors for a given disk is referred to as its *geometry*. Using real CHS values also prevents the disk from easily re-assigning sectors where it discovers that the magnetic media is damaged, and instead the computer's software must deal with this problem itself.

Finally, as disks grew in size, it became apparent that the fields reserved for cylinder, head and sector numbers, both in software and in hardware, were too small to cope. As a result, IDE disks started to lie to the host computer about their geometry, both for compatibility with the PC BIOS and so that they could start to manage bad sectors automatically themselves.

Modern disks are not generally used with CHS values; instead, sectors are given a number starting from zero or one. This is often referred to as a *Linear Block Address* or LBA. When accessed in this way, the disk manufacturer is free to choose the locations of each sector on the physical media in any way they please so as to optimize the performance of their drive mechanism; normally the assignment of blocks is done in such a way as to guarantee good performance if the blocks are accessed in LBA order.

Filesystems, volumes and partitions

So we know now that disks can be regarded as a numbered array of sectors, each of which can store 512 bytes of data. There are disks where the sector size differs from that, but almost all hard disks used today stick with these 512 byte blocks.

If that was all that we had, life would be very tedious indeed. Even on a 1.44MB floppy disk, there are two heads (corresponding to the two sides of the disk), eighty cylinders and each track holds eighteen sectors, giving a total sector count of $2 \times 80 \times 18 = 2{,}880$ sectors. Keeping track of which of them are in use and where your files are on the disk would be a real pain.

A modern 500GB hard disk, on the other hand, has nigh on a *billion* sectors and can store hundreds of thousands or even millions of files. How are we supposed to locate that photo that we saved last week? You'd need a building full of filing cabinets!

Fortunately this is exactly the kind of tedious job that computers are good at, and the pieces of software that they used to keep track of all of your precious data are called *filesystem drivers*. Apple's HFS+ (aka Mac OS Extended) is one example, and you have probably also heard of Microsoft's FAT and NTFS filesystems as well.

Different filesystems use different approaches to store the information about where your files are on the disk, about the sectors that are or are not currently in use, and so on, but what they have in common is that given a disk formatted for use with that file system, and the name of a file on that disk, they can work out which sectors hold the data for that file and return the data to the program that is attempting to access the file.

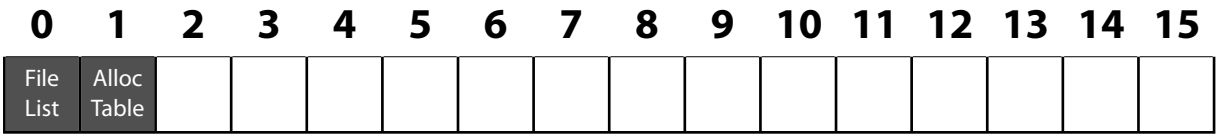
It is important at this point to note that the word *filesystem* is used interchangeably by some people to mean several different things; sometimes they mean the computer code responsible for locating your files on the disk; other times they mean a particular formatted disk; and on still other occasions they mean the abstract design of the data structures themselves.

This is all very confusing, and so Apple very sensibly tends to use the following words:

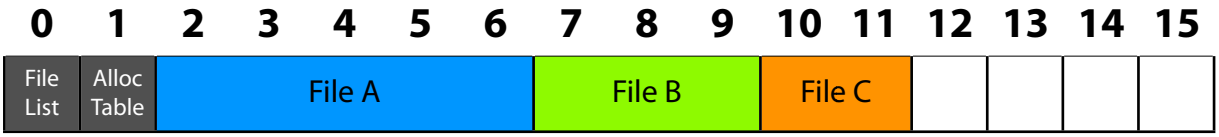
filesystem	The abstract design, including data structures and algorithms; the information presented in the filesystem specification – essentially an instruction book that tells you how the data is stored.
filesystem driver	The computer program that implements the abstract design, thereby allowing your Mac to read a disk formatted according to the filesystem specifications.
volume	The contents of a particular disk, formatted according to the filesystem specifications.

In addition, rather than the whole disk being used to hold a single volume, disks can be divided into pieces called *partitions*, each of which might contain a separate volume.

(a) SimpleFS Version 1.0



(b) After saving some files



(c) And deleting File B

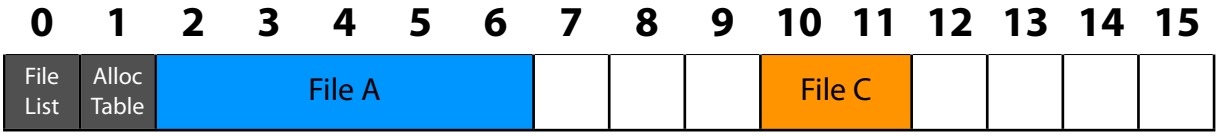


Figure 7 SimpleFS version 1.0

To help us understand what the filesystem does, let's take a look at an idealised filesystem, which we'll call SimpleFS.

SimpleFS version 1 supports storage of files in contiguous chunks on disk. We'll imagine that we have a very small disk with sixteen blocks total. We'll allocate one block to hold a list of the files on the disk, and in case we want to cope with bigger disks in future, we'll allocate another to hold information on which blocks are allocated, as shown in Figure 7 (a).

We'll skirt over exactly how this information is stored in these reserved blocks, as it doesn't matter for the purposes of our explanation.

So, let's save a file to our disk; our file is 3,000 bytes in length, so it takes up 3,000 ÷ 512 = 5.9 blocks. SimpleFS only lets us allocate entire blocks, so we'll use 6 blocks to store this file — call it "File A". We'll also save "File B", 1,300 bytes long (3 blocks), and "File C", 1,000 bytes long (2 blocks). The result is shown in Figure 7 (b).

Easy so far, right?

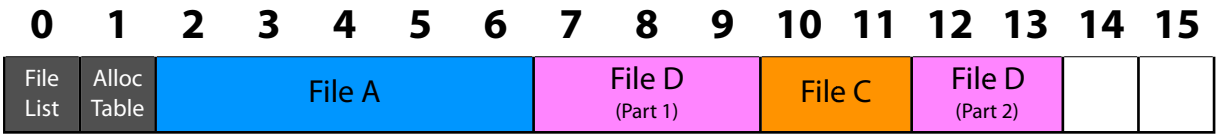
Next, let's delete "File B" from our disk. The result is shown in Figure 7 (c).

Imagine we now wish to save a new file, "File D", and that this new file is 2,400 bytes in length (5 blocks). We have *seven* blocks free, but the largest *contiguous* space is only four blocks long! That's no good — we can't save our new file even though we have 50% of the usable blocks marked as not allocated!

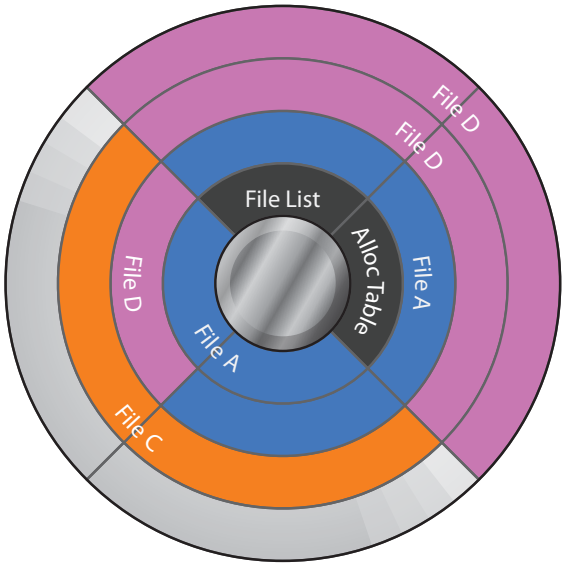
Let's imagine for a moment that we can somehow upgrade our disk to a new SimpleFS 2.0 that allows us to use all of these blocks to store our data. We still want to save "File D", which takes up five blocks, so we store the first three in blocks 7, 8 and 9, and the remainder in blocks 12 and 13. Figure 8 (a) shows the result.

This is great! We can use every block on the disk if we want.

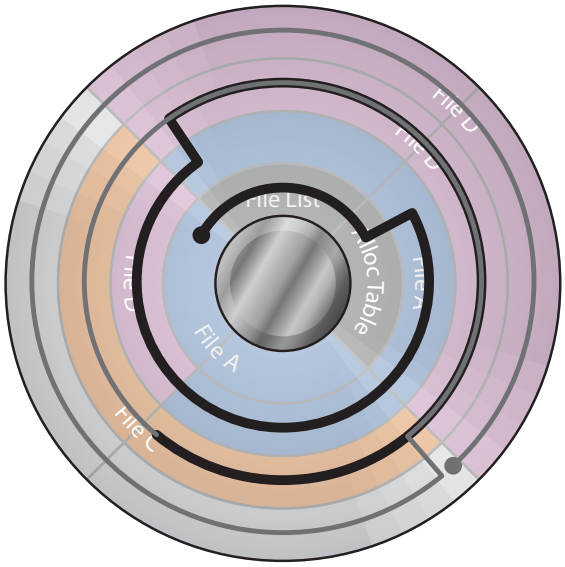
(a) SimpleFS Version 2.0 with File D



(b) On-disk layout



On-disk layout of the blocks, assuming a very simple mapping of linear block address to physical disk blocks.



The path taken by the disk head to read File D

Figure 8 SimpleFS version 2.0

Unfortunately, we have created a problem for ourselves. Let's assume that the disk we've formatted for use with SimpleFS is very old and slow. For sake of argument, let's give it a single platter with only a single side, so we have one head, and let's give it four cylinders, with each track holding four sectors.

For the next part, you can follow what is going on by starting at the black spot in the right hand part of Figure 8 (b) and tracing the line as it goes clockwise around the disk.

Assume we want to read "File D" from the disk, we first have to go and find it in the file list, which means reading block 0. The file list tells us where "File D" is on the disk, so the next sector we want to read is block 7, which on our disk is on track 1; we're currently on track 0, so we move the head out out one track. But we aren't in the right place to read block 7, so we have to wait for the disk to revolve until the head is over block 7 so we can read it.

Having read block 7, we now need block 8; this is on track 2, and our disk head is over track 1, so we move the head outwards again. Yet again, we aren't in the right place to read block 8 (we've missed the beginning of it), so we need to wait for the disk to rotate again.

(If you're following along at this point, switch to the grey line.)

We can now read blocks 8 and 9 one after the other, without moving the disk head. The next block, though, is block 12, and that's on the outer track, track 3, so we yet again need to move the head out one track, and since yet again we aren't in the right place yet, we need to wait for the disk to rotate.

Finally, we read blocks 12 and 13 one after the other, and we're done.

During this process, the disk has revolved three and a half times, and we've had to move the head three times, not counting any initial movement to get to the starting point on track 0.

Let's say this disk is very slow indeed, and takes half a second to move the head, and rotates at 20rpm. We've spent a total of $3 \times 0.5 + 3.5 \div 20 \times 60 = 12$ seconds reading this file.

Now let's consider what might happen if we swapped the positions of the end of "File D" with "File C". Well, this one simple change will put the final blocks of "File D" on the same track as the previous two blocks, so we could read 8, 9, 10 and 11 one after another, without moving the disk head and without waiting for the disk to rotate. We save one head movement and one and a half rotations!

In that case, we would only spend $2 \times 0.5 + 2 \div 20 \times 60 = 7$ seconds.

That's a 42% time saving, just for moving the blocks of "File D" so that they're next to one another on the disk!

When a file is split into more than one piece like "File D" was here, it is said that that file has become *fragmented*, and the pieces are typically referred to as *fragments*. On the Mac filesystem, HFS+, you may also see them referred to as *extents*, which is a reference to the way the filesystem maintains the information about the fragments of a file.

As you can hopefully see from this (heavily simplified) example, allowing the filesystem to fragment files is a trade-off. Without it, you might not be able to use all of the space on your disk, but on the other hand, a fragmented file might take a lot longer to read or write than it would if the file was in a single piece.

Clearly, therefore, what you don't want is for a file to be fragmented when it didn't have to be.

What does my Mac do to avoid fragmentation?

You've probably heard some people say "You don't need to defragment Macs", and it's true that the Mac filesystem, HFS+, includes a number of features designed to avoid fragmentation and thereby improve the performance of your machine.

The simplest of these is the way it chooses which blocks to allocate on the disk. In general, computer programs don't tell the computer the size of the file they intend to save before saving it. Even if the computer delays writing to the disk, it might not know the full size of the file within a reasonable amount of time. As a result, picking a space of exactly the right size is difficult.

At first sight, it might seem that this will be a huge disadvantage when our goal is to fragment files as infrequently as possible, but it turns out that computer scientists have shown that choosing a space of exactly the right size (or *best-fit*) is no better in practice than other ways of choosing a space to allocate.

There are various other allocation strategies that have been investigated; for instance, we might simply pick the first available space (this is called *first-fit*). Another rather surprising choice is to pick the largest space available, and always use that (*worst fit*). Again, it has been shown that in general neither of these is worse than best-fit!

Many filesystems use first-fit (or variations on it), partly because they don't know the size of the file when they have to pick the first block, and partly because it avoids searching through all of the free spaces to find the best or worst option.

Apple's filesystem engineers, though, spotted something about the way people use disks. Until very recently, the files you saved on your disk were very likely to be much smaller than the disk you were using. As a result, it would take a long time for a typical user to use their disk completely. Imagine, rather than picking the first space, or trying to find the "best" space, just choosing the *next* space. If the user has yet to use every block on their disk, we know there is a good chance that, no matter how large the file we need to save, we won't need to fragment it, and the best part is that keeping track of the next space to use is easy — we just keep a bookmark, or *pointer* to it. This approach is known as a *roving pointer* algorithm.

This clever choice worked well when most people used their computers to store word-processor documents, spreadsheets and other simple data files; it took a long time for a typical user to get through every block on a disk, and so it was a long time before the computer had to fragment a file. However, modern media files, such as photos, iTunes or MP3 music files and particularly video files and movies are *much* larger and make it much easier for an ordinary user to defeat this simple approach.

Under Mac OS X, HFS+ *also* includes some limited automatic defragmentation. If you have journaling turned on, *and* the fragmented file is smaller than twenty megabytes, *and* it has more than eight fragments, Mac OS X will defragment it for you when you access it, assuming that there is sufficient contiguous free space.[1].

[1] Correct at time of writing.