



How to ensure your data gets written

Copyright © 2002 Persistor Instruments Inc. All rights reserved.

Problem – file system writes are seldom holistic or immediate

Writing data to a Compact Flash card in C using the CF2 is a simple process. Open a file, write to the file and then, when you are done, close the file. It sounds simple enough. But what happens if power is removed between the time the file is opened and before it is closed? A directory of the card may show the file you were writing to is present but its size may be zero.

Has the data been lost? Well, not necessarily, although some of it may be lost. It may be possible to recover the data but sometimes it cannot. This all begs the question, why does this happen? During data logging using a CF2, data is typically written to the Compact Flash using a C call to fwrite. The call to fwrite may not necessarily write data to the card. The file system 'buffers' the data and when the buffer becomes full, it is written to the Compact Flash card. Each time the data is written to the Compact Flash card, the File Allocation Tables need to be updated to reflect the increase in the size of the file, the sectors it occupies and so on. If power is interrupted during a write, there is no way to guarantee that the FAT gets updated.

Before you think this is just a problem with Compact Flash remember that any modern PC that uses a hard disk suffers from the same problems. Today, everyone has learned to 'Shutdown' their PC before removing power. If you did not do this you would be forced to run a 'Scan Disk' on the hard drive the next time you turned on the PC. Same problem: files may have not closed properly and data may be 'lost' on the disk.

Solution – trick the directory system with low-level writes

So what do we do? How can we make the CF2 more robust? There is a slick little trick we can play on the CF2 and the Compact Flash. The first step is to format a Compact Flash card. We would also need to make sure that we erased every sector of the card to all ones (FFs).

Next, we need to open a file on the Compact Flash card and essentially make it look like it occupies the entire Compact Flash card. If we were to ask PicoDOS for a directory of the card, we would see one big file and there would be no spare bytes left on the card.

Using some low-level function in the CF2 we have the ability to write directly to the sectors on the Compact Flash. We can also use other functions to determine the starting and ending sector numbers for the entire Compact Flash card. These sectors are numbered sequentially starting with some low number like 95 all the way up to some large number that is dependent on the size of the Compact Flash card you are using. In our logging program, we just write data directly to the sectors that our file occupies. If the power is interrupted during writes we may lose the last buffer of data in RAM but we do not lose what we have recorded so far. We have avoided disaster because the FAT does not need to be updated while we write because we already know that the file uses the whole card!

We may not have necessarily filled up the entire card during logging though. So how do we determine where the end of our data file is within this one large file? Just open the file for binary read access and look for the FFs that signify an erased sector. This works as long as the data that you store is not all FFs to begin with (which of course is usually the case).

How to ensure your data gets written

DirectWrite

DirectWrite is written using Codewarrior and the Persistor 'ToPico' stationery. The ToPico stationery creates a custom version of PicoDOS. When you compile and run DirectWrite you will see the **BB)C:/>** prompt. Here are the commands that you will run.

PREP

Prep creates a file on the Compact Flash card which will be where we store data.

Syntax: PREP <filename>

Prep will ask you 2 questions: the first is reminding you that this will erase all Compact Flash data and the second is a similar question from the format command. The format command, in addition to formatting the Compact Flash card, will erase all the data sectors on the card. It looks like this:

```
PREPping this drive will erase all of its data!
Are you sure [N] ? Y←
C:
Formatting this drive will erase all of its data!
Are you sure? Y←
Formatting C: ...
Erasing NNNNN Sectors
```

Where NNNNN will be different depending on the size of the card you are using. Notice that you will see the sector numbers scroll by as they are being erased. Next, this message will appear:

```
Creating file DATA.DAT with fake length of nnnnnnnnnn bytes
Be patient, this takes about 1s/MB...
```

Where nnnnnnnnnn will also be different, depending on the size of the card. Once, the file has been created you are ready to record data.

DD

Dump Data displays sector data in a hexadecimal format.

Syntax: DD <Start sector> [<Endsector>]

FL

FL displays the first and last logical sector numbers that the file created using PREP occupies.

Syntax: FL



Startad

Startad will start data collection and write the data to the Compact Flash card starting at the first logical sector (see FL) and will continue until it writes to the last logical sector. It is very simple and it does not know if a previous collection operation has occurred. It is left to the user (you) to modify this code for your application. If you wanted Startad to append to the data it had already collected, you would need to write a small piece of code that search for the next available erased sector. New data would be stored starting at this new sector.

Summary

DirectWrite is meant to serve as an example. You will likely remove from it those pieces of code that suit your application. As stated in the beginning, it is still possible to loose data with DirectWrite. But the only data that is lost is the data in RAM that was not written to the card before power was lost.

The best solution would be to provide a power source that cannot be interrupted or which can be controlled in such a way that data in RAM can be saved before power is removed. When this is not possible, I hope that the techniques demonstrated in this example provide an alternative that minimizes data loss in your application.