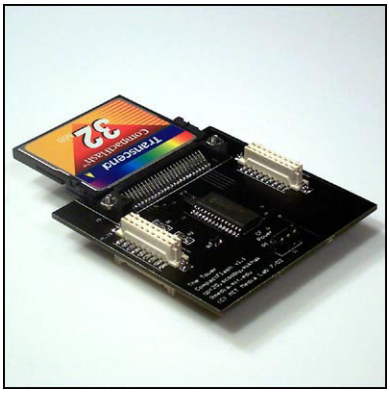
CompactFlash Layer



Description

The CompactFlash layer is used to communicate with CompactFlash I and II cards, as well as Microdrive units, to provide a vast amount of data storage when needed for storing significant sensor data, binary data files, or even providing full filesystem support to the Tower.

Hardware Detail

A reduced-IDE interface is used to communicate with the cards, minimizing the complexity required for hardware connectivity. Data can be written to and read from the card on a sector-by-sector basis. It is necessary to read and write full sectors, as partial operations will leave the card itself in a bad state. When using the higher capacity Microdrives, it may be necessary to switch the card power to the secondary power bus, as spinning up a Microdrive uses a significant amount of power that may be unavailable from the primary battery power source.

Layer Code

The include file for the CompactFlash layer contains eight functions, one checking to see if a card is in the slot, one each for turning the card power on and off, one for initializing a card to communicate with, one each for starting a sector write or a sector read, and one each for actually writing or reading a byte within the sector. *(All of these functions are written for the PIC Foundation. The include files used with other foundations differ slightly, due to the presence of local variables.)*

The **cf-is-there** function does not take any arguments, and will return a value of either 1 or 0 depending on whether or not a card is currently inserted in the slot. The function since a single value to the layer, a "0" indicating that a query is being made. The result is read out and returned, ignoring the first byte back, which is simply the number of arguments to follow, since it is known to be one.

```
to cf-is-there
      i2c-start
      i2c-write-byte $10
      i2c-write-byte 1
      i2c-write-byte 0
      i2c-stop
      i2c-start
      i2c-write-byte $11
      ignore i2c-read-byte 1
      seti2c-byte i2c-read-byte 0
      i2c-stop
      output i2c-byte
end
```

The **cf-power-on** function takes no arguments, and is used to power on the CompactFlash card itself. Since the cards can draw a lot of power, it is advantageous to keep them powered down when not actively accessing them. The function sends a single value of “1” to the layer, indicating the desired command.

```
to cf-power-on
      i2c-start
      i2c-write-byte $10
      i2c-write-byte 1
      i2c-write-byte 1
      i2c-stop
end
```

The **cf-power-off** function takes no arguments, and is used to power off the CompactFlash card itself. Since the cards can draw a lot of power, it is advantageous to keep them powered down when not actively accessing them. The function sends a single value of “2” to the layer, indicating the desired command.

```
to cf-power-off
      i2c-start
      i2c-write-byte $10
      i2c-write-byte 1
      i2c-write-byte 2
      i2c-stop
end
```

The **cf-init** function takes no arguments, and is used to initialize the compact CompactFlash card after it has been powered on. The initialization routine sets up internal buffering and cylinder information, and prepares the card to be accessed. The function sends a single value of “3” to the layer, indicating the desired command.

```
to cf-init
      i2c-start
      i2c-write-byte $10
      i2c-write-byte 1
      i2c-write-byte 3
      i2c-stop
end
```

The **cf-start-write-sector** function initializes a sector-write operation, taking one argument, the sector number to begin at. The first argument sent to the actual layer in this case however, is a “4”, signifying a sector write operation. When the layer receives the request to begin the write, it sets the cylinder and head positions as needed to point to the desired sector. It is important to note that sectors must be written in their entirety, so 512 bytes must be written following this command, or the card itself will be left in a bad state.

```
to cf-start-write-sector :sector
      i2c-start
      i2c-write-byte $10
      i2c-write-byte 4
      i2c-write-byte 4
      i2c-write-byte 0
      i2c-write-byte :sector / 256
      i2c-write-byte :sector % 256
      i2c-stop
end
```

The **cf-start-read-sector** function initializes a sector-read operation, taking one argument, the sector number to begin at. The first argument sent to the actual layer in this case however, is a “5”, signifying a sector read operation. When the layer receives the request to begin the read, it sets the cylinder and head positions as needed to point to the desired sector. It is important to note that sectors must be read in their entirety, so 512 bytes must be read out following this command, or the card itself will be left in a bad state If only a specific byte within the sector is needed, it’s still necessary to issue a fixed number of dummy-reads to increment the read pointer to the desired position, read that byte, and then finish off the sector with the necessary number of dummy-reads to bring the total to 512 bytes.

```
to cf-start-read-sector :sector
      i2c-start
      i2c-write-byte $10
      i2c-write-byte 4
      i2c-write-byte 5
      i2c-write-byte 0
      i2c-write-byte :sector / 256
      i2c-write-byte :sector % 256
      i2c-stop
end
```

The **cf-write** function is used following a **cf-start-write-sector** command to actually write data to the CompactFlash card. The function takes a single argument, the data byte to be written, and then sends a “6”, signifying a data write operation, followed by the byte itself.

```
to cf-write-byte :data
      i2c-start
      i2c-write-byte $10
      i2c-write-byte 2
      i2c-write-byte 6
      i2c-write-byte :data
      i2c-stop
end
```

The **cf-read** function is used following a **cf-start-read-sector** command to actually read data from the CompactFlash card. The function takes no arguments, and just sends a “7” to the layer itself, signifying a data read operation. After the request is sent to perform the read, the result is read out of the layer’s transmit buffer as a single byte, and returned to the calling function. Actually two bytes are read out, but the first is ignored, as it is known to be a “1”, representing the number of arguments to follow.

```
to cf-read-byte
      i2c-start
      i2c-write-byte $10
      i2c-write-byte 1
      i2c-write-byte 7
      i2c-stop
      i2c-start
      i2c-write-byte $11
      ignore i2c-read-byte 1
      seti2c-byte i2c-read-byte 0
      i2c-stop
      output i2c-byte
end
```

Examples of Use

To use the CompactFlash layer, a good place to start it to make sure that a card is present in the slot. We can check for a card by using the **cf-is-there** function like this:

```
waituntil [cf-is-there] print-string "|Card is present.|
> Card is present.
```

In the code above, we simply wait until the **cf-is-there** function returns true, and then print the desired string to the terminal, letting us know that a card has been inserted. Before we can write to or read from the card, we need to power it on and initialize it. That's as simple as:

```
cf-power-on
cf-init
```

Don't forget to power on the card before you try to initialize it, or it won't initialize properly. As soon as the card is initialized, lets write some data to it:

```
cf-start-write-sector
setn 0
repeat 512 [cf-write n setn n + 1]
```

We've just written an entire sector worth of data. All we had to do was call the **cf-start-write-sector** function, and then write 512 bytes. The data we wrote is an incrementing sequence from 0 to 255 repeated twice. Even though the variable n will continue past 255 to 511, **cf-write** only writes a single byte, which by default will use just the lowest byte of the number given to it, which ranges from 0 to 255. Let's try reading that data back out to make sure it looks okay:

```
cf-start-read-sector
repeat 512 [print cf-read]
> 0
> 1
> 2
> 3
(lots more data)
> 254
```

```
> 255
> 0
> 1
(lots more data)
> 254
> 255
```

The code to read out is pretty straightforward. Almost like writing a sector, we just call **cf-start-read-sector**, and then read out 512 bytes. Finally, let's power down the card since we're not going to be using it for awhile, and don't want to waste those precious batteries:

```
cf-power-off
```

Currently, only low-level reads and writes as shown above have been implemented. We hope to soon implement a full FAT-32 filesystem on the layer, to make it even easier to transfer large amounts of data between Towers, desktop computers, and other CompactFlash-enabled devices.