



Development Hardware for Jaguar Console

## **Operating Manual**

*6 May 2009*

**Changelog:**

13 Sep 08 – 1.1.0 - Added ABS support to JCP flash, minor grammatical clarifications

16 Sep 08 – 1.2.0 – Added basic use page for 'up' command, improved command line description and examples.

4 Oct 08 – 1.3.0 – Added description of 'word' flash option for marginal power supplies

17 Oct 08 – 1.4.0 – Added new BIOS upgrade in JCP and 'jcpauto' experimental tool

3 Mar 09 – 2.0.0 – Updated for revision 2.0. Fixed references to micro-B USB to read mini-B.

7 Apr 09 – 2.0.1 – Updated for Jaguar CD tweaks and bypass

6 May 09 – 2.0.2 – Updated for JCP 2.02.05 adding the -h switch

Software and Documentation © 2008-2009 Mike Brent. All rights reserved.

Specifications are subject to change without notice. All efforts have been provided to provide accurate documentation, however, errors may exist. No warranty is provided for this board or the documentation. SkunkBoard is not an approved Jaguar product and no person or company makes assurances that it will function correctly, or at all, with the Jaguar system. User assumes all risk and responsibility regarding use of the SkunkBoard. Skunkboard is not authorized to be used in a manner that violates any local laws.

User Support forum: <http://www.harmlesslion.com/phpBB2>

## Skunkboard Basics

Skunkboard is an add-on board for the Jaguar home video game system, primarily intended to be used for development of new software.

Skunkboard Rev 1 has the following specifications:

- 4MB of 16-bit flash organized as 2 meg x 16 bit, mapped at \$800000, rated for 100,000 erase cycles.
- Xilinx 9572XL CPLD with 1600 gates for glue logic
- Cypress C67300 USB Controller and 16-bit RISC co-processor
- USB Mini-B device connector for communication with PC
- Two USB Type A host connectors for future use. Note that at this time there are no plans in place to develop these ports.



SkunkBoard Rev 2 has the same specifications, except for the flash:

- 8MB of 16-bit flash, software configurable to appear as either 2 banks of 4MB, or a single bank of 6MB, always mapped at \$800000

SkunkBoard is connected to a PC running Windows, or, though unsupported, Linux or OSX. All interaction takes place through a program known as JCP, short for Jaguar CoPy.



**Warning:** SkunkBoard does not ship with a shell, and is built using static sensitive, low-voltage parts. Always use safe anti-static practices when handling Skunkboard. Ground yourself by touching a metal object before touching any part of Skunkboard, and handle it only by its edges. Keep it in its anti-static bag when not in use, and refrain from moving around when holding it bare.

Likewise, use care when Skunkboard is inserted into the Jaguar. Do not pull on the USB cord or apply horizontal pressure, this may put stress on the cartridge port and possibly damage it.

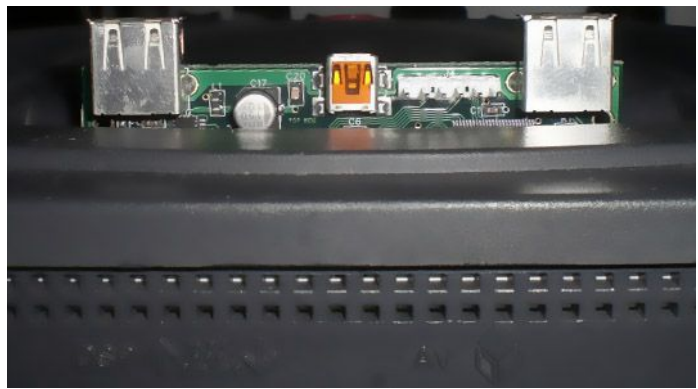
Never insert your SkunkBoard into the Jaguar with the power switch on – this may damage the card. If you do, immediately power off the Jaguar and wait 10 seconds before powering it on again. We recommend only official Jaguar power supplies be used with the console.

SkunkBoard rev 2 is built with tabs that *may* fit standard Jaguar cartridge shells. However, do not attempt to force the board into a shell. The USB ports require substantial clearance and standard shells will require substantial modification to be usable. Do not crush the USB ports.

## ***Installing Skunkboard***

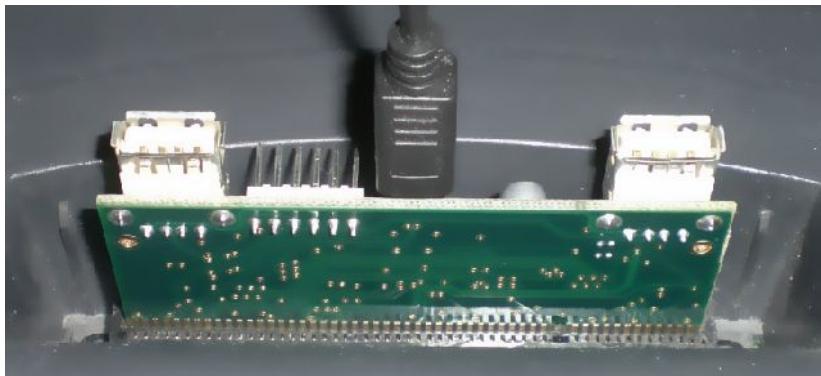
### **Jaguar**

- 1) Ensure your Jaguar is installed and functioning as documented in the Jaguar owner's manual. If you have not used it for a while, you may need to clean the cartridge port. Test with a known working game cartridge.
- 2) Turn the Jaguar power OFF.
- 3) Remove the Skunkboard from its anti-static envelope, and gently insert it into the Jaguar's cartridge port. It only installs in one direction, with the chips facing the rear of the machine. Line it up carefully and do not force it. Do not rock it in the slot, apply gentle downward pressure only.



*Ensure the Skunkboard is inserted fully into the cartridge slot, the USB ports should nearly line up with the case as shown here. Skunkboard 2 looks similar.*

- 4) Obtain a USB-A to Mini-B connector cable. The Mini-B connector used by SkunkBoard is shown here. Attach one end to your PC, and the other end to the Skunkboard's center USB port, as shown here.

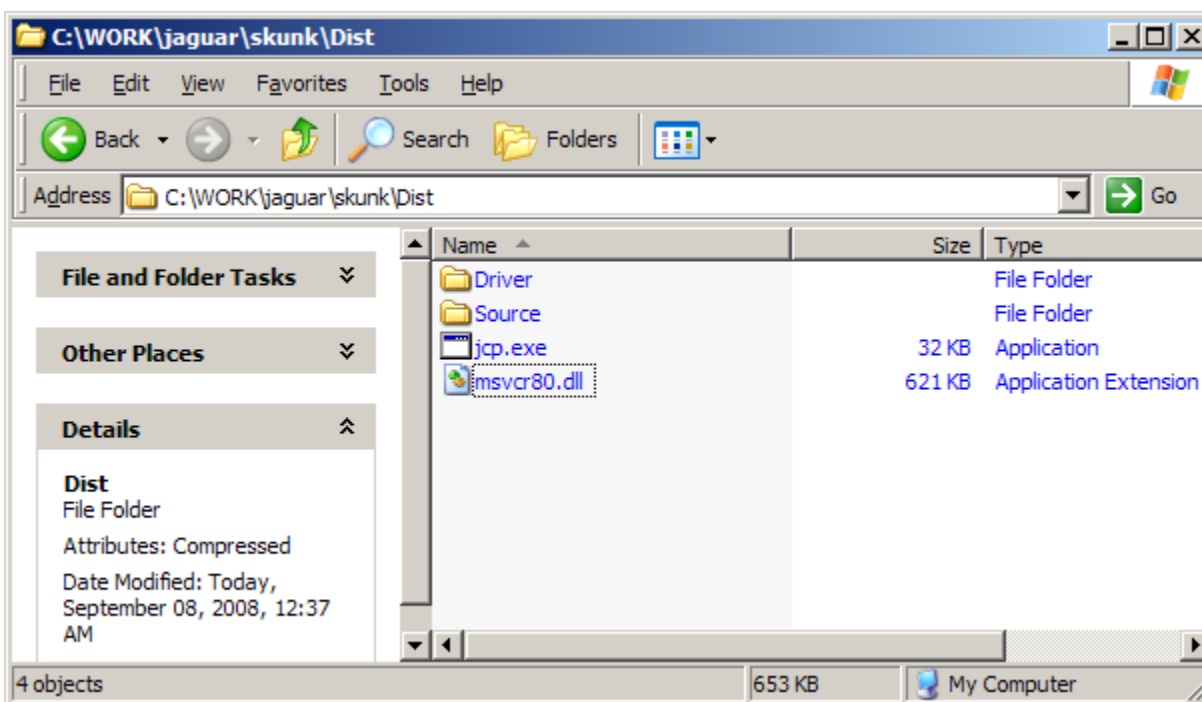


- 5) Now prepare to install the software in the next section.

## Windows Installation

- 1) Before powering on the Jaguar, obtain the latest software package from <http://harmlesslion.com/software/skunkboard>
- 2) Unzip the file to your working folder.
- 3) You should see four items in the root: a folder named **Driver**, a folder named **Source**, a DLL called **msvcr80.dll** and an executable named **jcp.exe**.

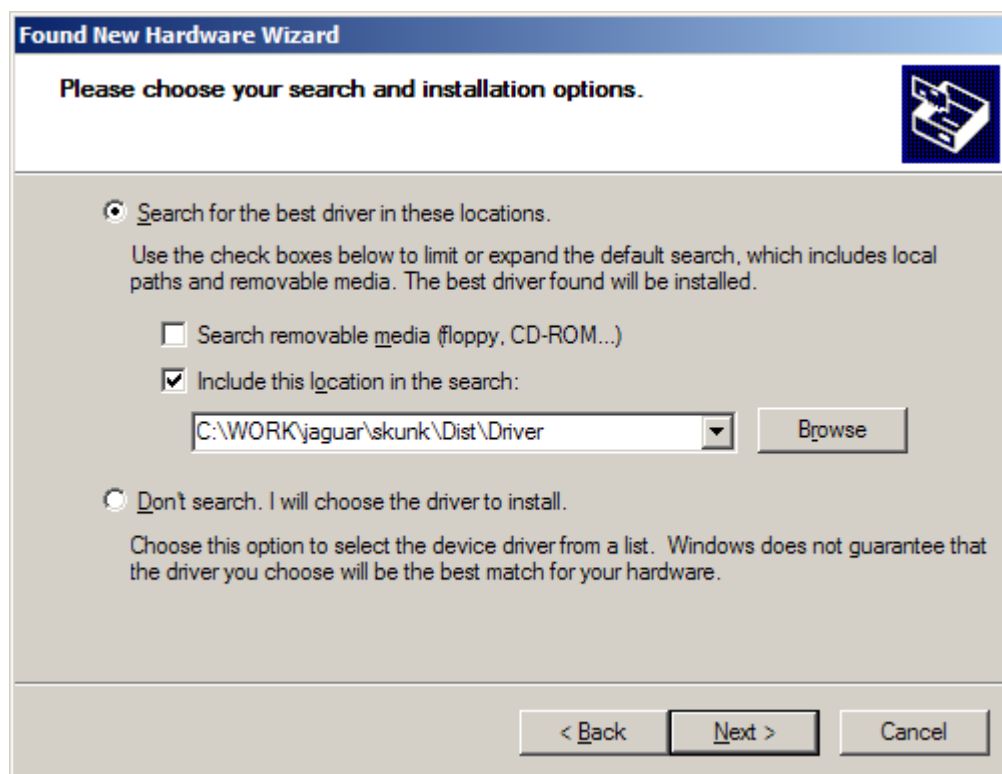
*Note: If you have a **jcpauto.exe**, you may ignore that.*



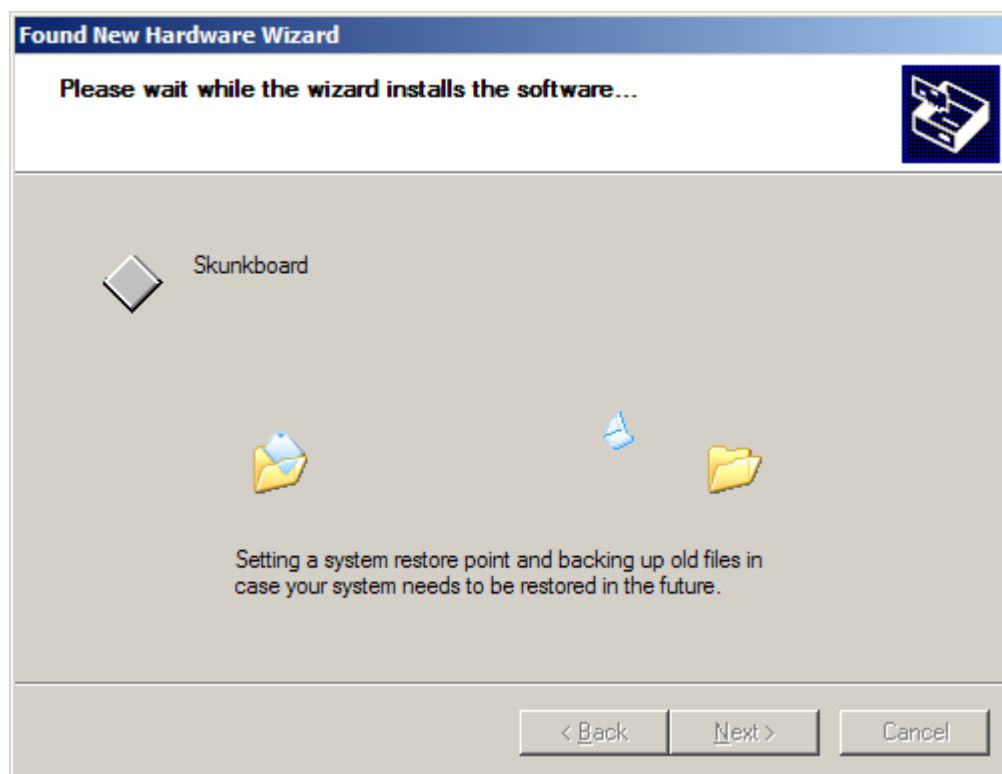
- 4) Ensure the USB cable is connected, and turn on the Jaguar. Windows should detect the device and ask for a driver. Select to install from a list or specific location. *Some versions of Windows may show a slightly different screen or list of options. Ultimately you want to select 'Have Disk', or a similar option to tell Windows where to get the driver.*



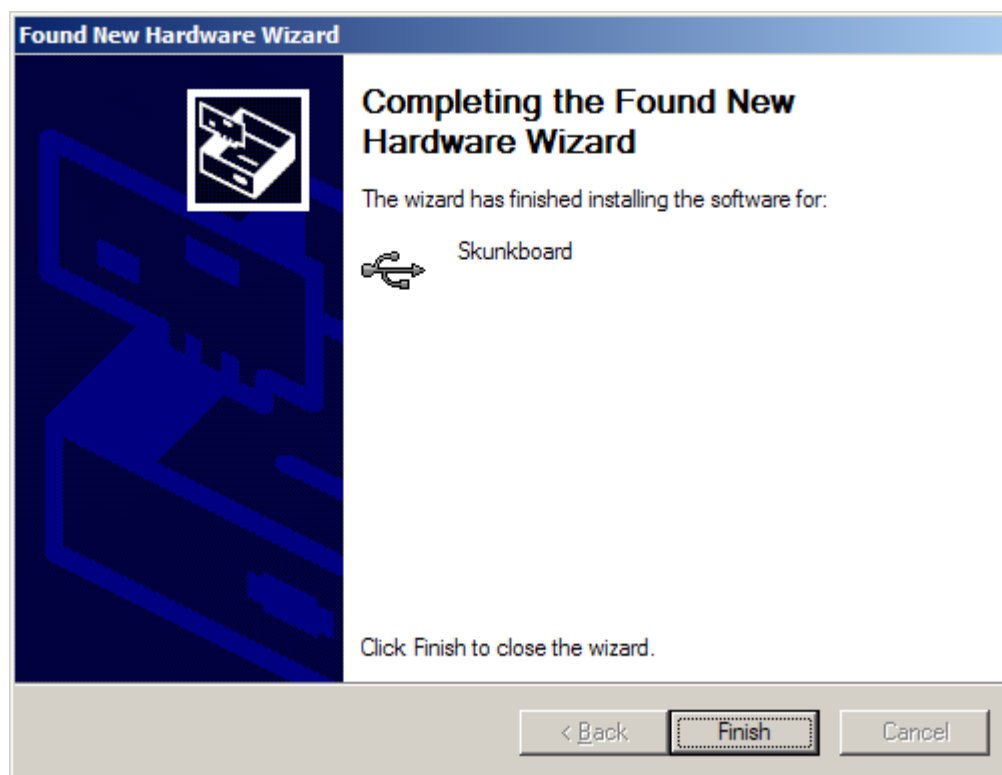
- 5) On the next tab, let Windows search, but specify the location that you saved the drivers at as well, as in this screenshot:



6) Windows should find and install the driver:



7) When done, click Finish:



**Note:** Skunkboard communicates through a standard Cypress EZ-OTG chip. Your system may detect and install a default driver instead. This is okay, the system will still work. The driver included with Skunkboard is a simple LibUSB stub and does not contain any Skunkboard specific code.

## Linux Installation

Under Linux, Skunkboard is not officially supported. However, we've made an effort to provide the files you need to build JCP, and JCP has been tested under Linux successfully, using Debian *etch* and kernel 2.6.18, with libUSB 0.1 and gcc 4.1.2. You must have all this (or compatible versions) installed and running properly on your Linux box before attempting to use Skunkboard.

- 1) Obtain the latest software package from <http://harmlesslion.com/software/skunkboard>
- 2) Unzip the file to your working folder.
- 3) You should see four items in the root: a folder named **Driver**, a folder named **Source**, a DLL called **msvcr80.dll** and an executable named **jcp.exe**.

```
raccoon:~/jcp/Dist# ls -l
total 40
drwxr-xr-x 2 root root  4096 2008-09-08 00:37 Driver
-rw-r--r-- 1 root root 32768 2008-09-07 03:22 jcp.exe
drwxr-xr-x 5 root root  4096 2008-09-08 00:29 Source
-rw-r--r-- 1 root root 479232 2008-09-07 03:22 msvcr80.dll
raccoon:~/jcp/Dist#
```

- 4) You may discard the *jcp.exe* (and *jcpauto.exe* if present), *msvcr80.dll* and the *Driver* folder, as these are only useful under Windows.
- 5) Change directory into Source/JCP, and execute 'make':

```
raccoon:~/jcp/Dist/Source/JCP# make
gcc jcp.c -ojcp -lusb -lrt

raccoon:~/jcp/Dist/Source/JCP# ls
dumpver.h    jcp      jcp.vcproj  Makefile    romdump.h  univbin.h
flashstub.h  jcp.c    libusb.lib  ReadMe.txt  turbow.h   winusb.h
raccoon:~/jcp/Dist/Source/JCP#
```

- 6) 'jcp' must run as root. You may want to chown and sticky it – if so, use these steps (with *sudo* if necessary):

```
chown root ./jcp
chmod +s ./jcp
```

- 7) The output executable 'jcp' should be ready to go, you can test with *./jcp* (your option list may vary):

```
raccoon:~/jcp/Dist/Source/JCP# ./jcp
jcp v02.00.00 built on Mar 3 2009
```

```
* Usage: jcp [-rewf] file [$base]
-r = reset (no other args needed)
-f = flash (pass filename + opt base)
-wf= word flash (slow, only if 'f' alone fails)
-ef= erase whole flash
```

### ***Using the Command Prompt***

JCP is a command line tool. Therefore, if you are using a graphical user environment, you first need to open a command-line – in Windows this is either *Command Prompt*, *MS-DOS Command Prompt*, *command.exe*, or *cmd.exe*, depending on your operating system. Under Linux, it is usually called “*terminal*”.

You will need to know the path to JCP, and specify this path when you wish to invoke it. For example, if you have stored JCP at C:\Work\Jaguar\Skunk\Dist, as in the installation example, then you would invoke JCP like so:

```
c:\work\jaguar\skunk\dist\jcp.exe
```

or in the Linux example (where 'user' is your login, if you are working in your home directory):

```
/home/user/jcp/Dist/Source/JCP/jcp
```

Since it is often convenient to change paths, and these commands can get quite long typing the whole path every time, shells include a setting called the 'search path' which you can add your JCP folder to, to make it easier. Under Windows, type the following, substituting the path you actually stored JCP at:

```
path=%path%;c:\work\jaguar\skunk\dist
```

and in Linux, it varies some, but is usually:

```
export PATH=$PATH:/home/user/jcp/Dist/Source/JCP/
```

If done correctly, then JCP can now be invoked on either system by simply running *jcp* with no path specified at all.

It is not the intent of this manual to describe how to use a command line interface, you must be somewhat familiar with navigation and how paths and folders work to be comfortable using this tool.

## ***Using Skunkboard***

When you turn on your Jaguar with Skunkboard plugged in, you should immediately get a solid green screen, *without* the Jaguar logo coming up first.

If you get the Jaguar red screen, power cycle the Jaguar by turning it off for four seconds , then check to ensure the board is securely seated in the cartridge port, then turn the Jaguar back on.

The green screen is your indication that the board is ready to go. You may now communicate with the board using JCP.

## ***Starting Already Loaded Software***

If you have previously loaded software onto the flash chip, you may start that software at this point using the first (left) controller.

For the first software bank (and the only bank on Rev 1 boards), press **UP**.

For the second software bank, press **DOWN** (Rev 2 only).

To launch a program in 6MB mode (Rev 2 only), hold '**A**' while pressing **UP**.

If no software is loaded in the selected bank, the Jaguar will probably crash, and you will need to power cycle it to get back to the green screen.

If unauthorized software is loaded on the flash when you select a bank, even if not the bank you selected, the screen will go red to indicate that the flash image may not be started. You will need to flash a new image to proceed. (*See the JCP section below for details on how to flash a file.*)

## ***Jaguar CD***

To bypass Skunkboard startup when it is in the Jaguar CD unit, hold '**C**' while powering on or resetting the unit. This will permit the Jaguar CD to start normally.

## ***Known Incompatibilities***

Skunkboard is known NOT to work in the following configurations. In both cases trying to start the Skunkboard leads to a crash of the system.

- In a developer Jaguar with the "Stubulator" BIOS. This BIOS assumes the cartridge is 32-bit wide, and starts at \$802000, ignoring the actual header. (The width problem means that even if a valid file is flashed at \$802000 it will still crash with this BIOS)
- Rev 1 Skunkboard with a BIOS earlier than 1.2.0 with the Jaguar CD attached. The Skunkboard boot process is not compatible with the Jaguar CD boot on these earlier revisions. Rev 1 boards should be updated with JCP (*see the JCP update section*), then they will work.

This crash is not permanent and does not harm the Skunkboard, but it does prevent Skunkboard from being used in these scenarios.

## **Using JCP**

JCP stands for Jaguar Copy, and is named after similar tools in Unix. Unlike those tools, JCP is a complete interface to your Jaguar and Skunkboard, and performs all interaction.

To use JCP, you execute JCP and pass optional switches to control its behavior, as well as, in some cases, a file to load.

The following sections describe the operations you can do with JCP. For these examples, it is assumed your command prompt is in the folder that you extracted the zip to (the same folder as *jcp.exe*). The text you would type is in [blue](#).

## **Standalone commands**

### **jcp -r**

This is the simplest command, -r will perform a hard reset of the Jaguar. This function will work in all cases so long as the Skunkboard itself is still reading the USB – it does not require any of the Jaguar's processors to be running. If it fails, then you must power cycle the Jaguar.

```
C:\skunk>jcp -r
jcp v02.02.01 built on Feb 28 2009
Resetting jaguar...
```

If you reset the Jaguar during a flash operation, or with certain crashes, you will get the red Jaguar failed boot screen instead of the Skunkboard green screen. In that case you must also power cycle the Jaguar.



To power cycle the Jaguar, turn it off, wait at least 5 seconds, then turn it on again.

### **jcp -s**

Another simple command, this will dump your Skunkboard's serial number and firmware version, like so:

```
C:\skunk>jcp -s
jcp v02.02.01 built on Feb 28 2009
Sending...
Boot version 02.00.01, Serial 2002
```

## **Commands that load to RAM**

### **jcp <filename>**

In this second-simplest implementation, JCP will load a file to the Jaguar's RAM and start it. This is used for 'BJL' style uploads, and files are usually in COF or BIN format. A COF, or *Common Object Format* file specifies all the necessary information for JCP to load it to the correct address. Files without this information will default to \$4000. If that is incorrect, they may require further information. For that, use the next format:

### **jcp <filename> \$address**

This incarnation loads a file, but specifies the load address for BIN and other files without a header. The address is an absolute hexadecimal value in memory. For example, most BJL files load at \$4000 – if you do not specify an address, this is the default. But some files load instead at \$5000. To use this address instead, load like so: *jcp spooky.bin \$5000*

### **jcp -o <filename> \$address**

Some files have information inside them about where to load, such as COF. If you try the override method, and JCP appears to ignore your command and loads to a different address than you specified, adding '-o' tells JCP you are overriding whatever the files says, and forces it to use your address. Loading a file at a different address than it was built for will usually cause the Jaguar to crash, requiring a reset. Therefore, it is rare you need this option, but it provided just in case. For example: *jcp -o myfile.cof \$5000*

### **jcp -h <cnt> <filename>**

Similar to the -o option, -h lets you override JCP's detection of the number of Header bytes to skip. This is very rarely needed, only if a misdetection occurs. It must be immediately followed by a space, and then the count as a decimal number. For example: *jcp -h 512 myfile.cof*

### **jcp -n <filename>**

This option permits you to load a file to Jaguar RAM, but *not* execute it immediately. This permits you to load several data files, for instance, then load the program and let it take over.

### **jcp -c <filename>**

This option loads the file to memory as per normal. After the Jaguar accepts control of the program, JCP will enter *Console Mode*. Console mode lets the Jaguar continue to communicate with the PC after the program starts, and is useful for debugging and development. The Console Mode is documented in another section, and is primarily useful to developers. This switch should not be combined with -n, since the Jaguar program will not be running.

#### **Example:**

```
C:\skunk>jcp JAGLION.COF
jcp v02.02.01 built on Feb 28 2009
COFF File:  Skip 168 bytes, base addr is $4000, length is 124029
bytes
Sending.....
Jag accepted start request at $00004000.

Finished in 563 millis, 220KB/second.
```

See also the Miscellaneous command “-x”.

## Commands that load to Flash

### **jcp -f <filename>**

The simplest way to flash a file, this works for most files. This version can parse most files with the extension ABS, ROM, JAG, and PRG, and if it can not, it will default to a load address of \$802000, which is usually correct. The flash is preceded by an erase phase which is fairly colorful. *In this example, note that quotes are needed around the filename as it contains spaces.*

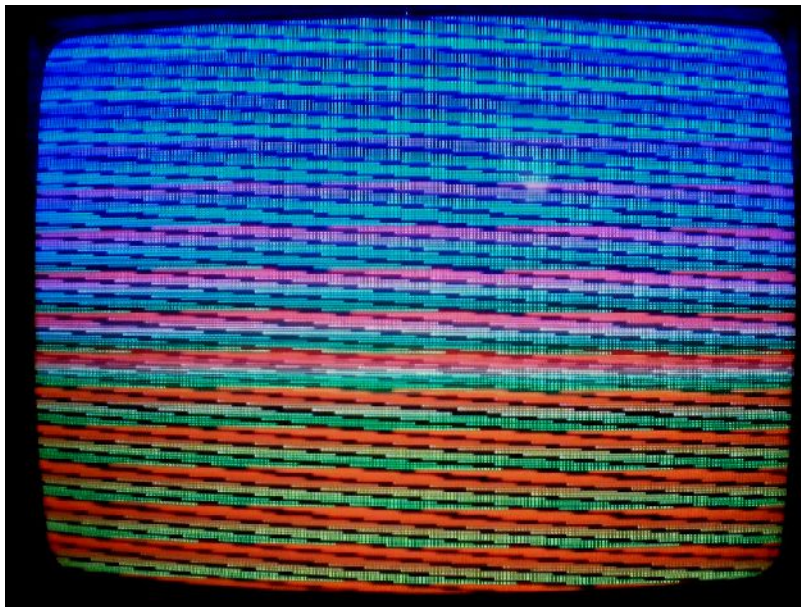
```
C:\skunk>jcp -f "Alien vs Predator (Alpha).jag"
```

```
jcp v02.02.01 built on Feb 28 2009
```

```
Sending...
```

```
..
```

```
Waiting for erase to complete (about 9s)..
```



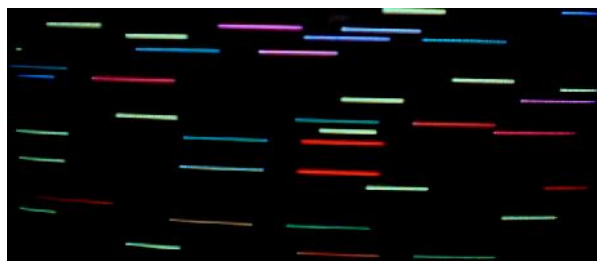
the next phase actually writes the flash:

```
Cart ROM: Skip 8192 bytes, base addr is $802000, length is 1627284 bytes
```

```
Sending.....
```

```
Jag accepted start request at $00802000.
```

```
Finished in 7265 millis, 223KB/second.
```



### **jcp -ef <filename>**

JCP will normally look at the file you are flashing, and only flash half the board if the file is under 2MB. If you are having problems with this detection, including the 'e' option will make skunkboard flash the entire 4MB, regardless of the size of the file being flashed.

### **jcp -f <filename> \$address**

This incarnation loads a file, but specifies the load address for BIN and other files without a header. The address is an absolute hexadecimal value in memory. It is very rare you will need this option for a flash file, as most load at the default of \$802000 and will be detected.

### **jcp -of <filename> \$address**

Most flash address files have information inside them about where to load, such as JAG. If you try the override method, and JCP appears to ignore your command and loads to a different address than you specified, adding '-o' tells JCP you are overriding whatever the files says, and forces it to use your address. Loading a file at a different address than it was built for will usually cause the Jaguar to crash, requiring a reset. Therefore, it is rare you need this option, but it provided just in case. For example: *jcp -of myfile.jag \$802100*. **Note:** *If you are relocating a file smaller than 2MB, you may also want to include the e parameter to erase the entire flash, as only the size is used to determine this.*

### **jcp -fh <cnt> <filename>**

Related to '-of', this lets you override the automatic detection of the number of Header bytes to skip, in the event that JCP misdetects it. This is rarely needed, but may be used to skip header padding on ROM files more often than RAM. This should only be used if you know JCP misdetects the number of bytes to skip. Note the order: 'h' must be the last entry in the group of switches, the others may be in any order. For a normal 'rom' header, the padding size is 8192. For example: *jcp -fh 8192 myfile.jag*

### **jcp -wf <filename>**

This writes the flash using a slower 'word' mode, rather than the fast 'double-word' mode. This may be necessary if your Jaguar power supply is marginal. The Jaguar runs on a 9VDC, 1200mA (or 1.2A) power input. The Skunkboard requires this full 9VDC in order to power the fast write mode. If your power supply is marginal, or you are using a third party power supply that provides less than 1200mA, your Jaguar may still function but flash writes will fail. (Often you will also see symptoms like faint lines or bands across the screen). Using this option may allow the flash to succeed, but if it does, you should replace your power supply as soon as possible. Requires boot version 01.01.00 or later.

**Note:** Skunkboard uses the flash memory range from \$800000 to \$801FFF for its own BIOS, and therefore, this area is not available for flash software to be loaded.

## Rev 2 – second bank and 6MB mode

All of the above commands may be prefaced with a '2' to indicate bank 2, or a '6' to indicate six megabyte mode. For example:

```
C:\skunk>jcp -2f "Alien vs Predator (Beta).jag"
```

This command will flash the specified program to the second 4MB bank on a revision 2 Skunkboard.

```
C:\skunk>jcp -6f "SixMBSlideShow.cof"
```

This command will flash the specified program in 6MB mode on a revision 2 Skunkboard. Note that the 6MB flash occurs in two separate cycles of erase then flash.

## Miscellaneous Commands

### **jcp -d <filename>**

This command dumps the current contents of the flash across the console to the file specified. There are some limitations with this.

First, JCP does not know the size of the data on the flash, therefore, the entire 4MB space is always dumped to the file.

Second, because the Skunkboard BIOS resides from \$800000 to \$801FFF, and is not useful to a normal cartridge, this area is not included in the dump. Instead JCP prepends the standard "TypeAB" cartridge header which will normally be correct for software on the board. If you are writing your own software, you will want to be aware of this behaviour so you can correct the header, if needed, before transferring your software elsewhere.

You may dump the second 4MB bank on a rev 2 Skunkboard by using **jcp -2d <filename>**.

```
C:\skunk>jcp -d test.rom
```

```
jcp v02.02.01 built on Feb 28 2009
Beginning dump to 'test.rom'...
Sending...
```

```
Receiving flash...
Closing file...
```

```
Dumped 4MB in 24s - 170KB/s
* Dump complete.
```

### **jcp -x <executable>**

The -x option may be used in any place that -c may be used, to trigger an external, third party console to interface with your program. For example, the Removers library may (as of the time of writing) contain an external shell with more capability than the Skunklib shell. If you have such a shell, this is the option that you will use to interact with it. Please follow the instructions included with that application.

## jcp -u

If you have a Rev 1 Skunkboards, identified with the serial (-s) command as having a boot version or BIOS of 01.01.00 or earlier, you can use this command to upgrade to version 01.02.xx. If your BIOS is 01.00.00, this update is *strongly recommended*, as it includes numerous reliability fixes and updates. In addition, the *word flash* (-wf) option requires this boot version.

The most valuable update in the 01.01.00 boot version is protection against accidental overwrite of the Skunkboard boot itself. Although very unlikely, it was observed in some situations that the boot could be damaged, rendering Skunkboard unusable. The boot version 01.01.00 adds additional locking to protect against this.

Boot version 1.02.xx adds fixes so that it will boot with a Jaguar CD attached, and minor fixes to the startup, as well as tweaks to ensure compatibility with Skunkboard rev 2 software.

If you already have boot version 01.02.xx or later, this call will not make any changes. Otherwise, the upgrade is very fast and should take less than 2 seconds.

The update is not required for Skunkboard Rev 2, as it already has all the fixes built in.

```
C:\skunk>jcp -u
jcp v02.02.01 built on Feb 28 2009
```

```
Sending...
Skunkboard BIOS Update 1.02.02
```

```
Current ver: 01.01.00, Serial 1003
```

```
Preparing...
```

```
Writing...
```

```
Verifying...
```

```
Current ver: 01.02.02, Serial 1003
```

```
Operation complete.
```

## *Programming With Skunkboard*

Programming software with the Skunkboard is very similar to developing for BJL. You build and link your files in the same way, then upload them to Jaguar RAM (or Flash) using JCP. Due to the speed of flashing, it is usually preferable to develop to RAM first, and then relink the software to run from cartridge space. However you can perform these operations however you like.

To aid with debugging, Skunkboard includes a programmer's library designed to interface with the JCP Console Mode. This library makes it simple to interface with the PC for text and files, both read and write.

This library communicates with the PC using a double-buffered system. Most functions will automatically write to one buffer or the other. If both buffers are full, the system will wait a short time for one to free up. If this times out, the console is marked unavailable. The timeout is fairly long, but if you use the system heavily you may need to increase the timeout to ensure the PC can keep up.

To use the library, add *skunk.s* to your project. This file is located in the \Source\SkunkLib folder. This library is free to use however you wish, including commercially, however, it's recommended you only use it for your own testing work as JCP specifications may change.

This library uses one DWORD of BSS space to track whether the console was last communicated with successfully. The rest is 68k program code. Should you choose to modify the library, it's recommended to make a backup first.

All console functions will delay if buffers are full, and will time out if they stay full. If the console reconnects then they should resume but previous accesses are lost. If your program \*relies\* on the console input then it should test `skunkConsoleUp` after a call.

You should not use these functions in your production cartridge, rather create dummy stubs that do nothing or comment the calls out in your code. Without the Skunkboard they may work, they may not, there are no guarantees.

None of these functions are 'thread safe' so you should resist calling them from interrupts.

All addresses must be on a word boundary. All lengths must be even, except text writes may be an odd count (but an even number of bytes are still read and transmitted).

The following functions are available in the SkunkLib:

### **skunkRESET()**

Resets the library, marks the console as available, and clears both buffers. This will wait to handshake with the PC console, and if that times out, the console is marked unavailable to all subsequent functions.

```
jsr skunkRESET
```

**skunkNOP()**

No action, no options. Can be useful for synchronization of the buffers with the PC console. For example, sending two skunkCONSOLEWRITE commands one after the other may lead to random ordering on the PC side. If there is concern about ordering, you can insert a skunkNOP to fill the alternate buffer.

```
jsr skunkNOP
```

**skunkCONSOLEWRITE(a0)**

Write text to the console, if it is active

a0 - address of the text to be written, terminated with a 0 byte

Will write to either buffer, whichever is free first. Note that if the length is not an even number, one additional byte will be sent to the PC, though it should not be printed. If two lines are written quickly, the order at which they come out is not guaranteed. Use NOPs if you want a guaranteed order. Note: the terminating 0 byte is very important! You will get garbage on the console or a crashed Jaguar otherwise. The '13' in the example is a carriage return, you might also need a line feed (10) for proper line wrap, depending on your console.

```
move.l    #HELLOWORLD, a0
jsr       skunkCONSOLEWRITE
```

HELLOWORLD:

```
; the 0 byte at the end is critical!
.dc.b     'Hello, world!',13,0
```

**a0=skunkCONSOLEREAD(d0)**

Reads text from the console keyboard and returns it at a0, which is a buffer d0 bytes long. The maximum return is 4064 bytes. Note the returned string is NOT necessarily NUL terminated, so you can't just turn around and print it! One way to ensure this is safe is to put your buffer in the initialized data section, and explicitly terminate the buffer after its length, or to specify a buffer one longer than you want, and explicitly write a 0 at the end.

```
move.l    #BUFFER, a0
move.l    #128, d0
jsr       skunkCONSOLEREAD
```

BUFFER:

```
.ds.b     128
```

**skunkCONSOLECLOSE()**

Instructs the console to close. No arguments. Expects the console is closed after the fact. No text nor file operations will work again after this call, unless the console is reattached. Ensures both buffers are clear before executing.

```
jsr skunkCONSOLECLOSE
```

### **skunkFILEOPEN(a0,d0)**

Opens a file on the PC in the current folder (paths are not supported)

a0 - points to the filename, terminated with a 0 byte

d0 - 0 for write mode. 1 for read mode.

This function will wait until the PC acknowledges the buffer to reduce ordering problems.

```
move.l    #FILENAME,a0
moveq     #0,d1          ; write mode
jsr       skunkFILEOPEN ; ordering guaranteed
```

FILENAME:

```
; the 0 byte at the end is critical!
.dc.b     'myfile.txt',0
```

### **skunkFILEWRITE(a0,d0)**

Writes a block on the PC to the currently open file (if open to write)

a0 - points to the data - will be updated to point after the data

d0 - number of bytes to write, up to 4060. (must be even)

This function will wait until the PC acknowledges the buffer to reduce ordering problems.

There's no guarantee of actual write, only that it was sent to the PC.

```
move.l    #BUFFER,a0
moveq     #14,d0         ; length of data to write
jsr       skunkFILEWRITE
```

BUFFER:

```
; note no 0 - this is because the data may be binary!
.dc.b     'Hello, world!',13
```

### **a0=skunkFILEREAD(d0)**

Reads data from the currently open file (if opened for read) which is a buffer d0 bytes long.

The maximum return is 4064 bytes. d0 is updated with the actual number of bytes read. If 0, either EOF or an error occurred. a0 is also updated to point after the new data in the buffer.

```
move.l    #BUFFER,a0
move.l    #128,d0
jsr       skunkFILEREAD
```

BUFFER:

```
.ds.b     128
```

### **skunkFILECLOSE()**

Instructs the currently open file to close. No arguments.

```
jsr skunkFILECLOSE
```

## SkunkBoard Rev 2 programming

Although not strictly recommended, the following instructions may be useful to programmers wishing to write software specifically for SkunkBoard revision 2. We reserve the right to change these interfaces without warning.

### **Set Bank 0**

```
move.w #$4ba0,$c00000 ; set bank 0
```

### **Set Bank 1**

```
move.w #$4ba1,$c00000 ; set bank 1
```

### **Set 6MB Mode**

```
move.w #$4ba0,$c00000 ; set bank 0  
move.w #$4003,$c00000 ; set 6MB mode
```

Note that the latter sequence for 6MB mode is strongly recommended, in case users set the mode incorrectly when starting from the joystick. If you publish on ROM, and still wish to permit Skunkboards to run the code, it should be safe to leave the instructions in place.

Also note that the bank switch is instantaneous. Thus, Set Bank 0 or Set Bank 1 should only be executed from RAM, unless you ABSOLUTELY KNOW what is going to happen when it changes.

## ***jcpauto.exe***

*jcpauto.exe* is a new experimental version of JCP that performs more actions automatically. Specifically, two actions are automated:

- If a flash-targetted file is passed without the -f switch, JCP will automatically switch to flash mode. *jcpauto* will only target bank 0.
- If a flash-targetted file is larger than 4MB, then 6MB mode is assumed.
- If the Jaguar is busy when JCP is invoked, a reset is automatically performed.

These actions were added to support automatic drag-and-drop support of files in the Windows GUI, though they may be useful elsewhere. If these functions prove useful and reliable, they will be turned on by default. If you have problems with *jcpauto.exe*, continue to use *jcp.exe* as before.

*(Under Linux or OSX, you may build this version by defining JCP\_AUTO in the header.)*

## ***GDB Support***

A PC-side server application, and a stub for the Jaguar side is available in \Source\GDB. The idea is, you launch jdb.cof on the Jaguar side (using JCP). Then launch JSERVE.EXE on the PC. Finally attach your existing 68k GDB to the local server using port 4567. The following GDB commands apply to starting it:

```
set target localhost:4567
load
run
```

This package is not yet ready to be supported and is provided only to those who want a starting point. Everything in this folder is subject to change. At this time, only the Windows package for Jserve.exe is included.

For more information and support, visit  
<http://harmlesslion.com/phpBB2> and locate the Skunkboard forum