# USB Drive Controller

**Written By: Steven Stuber**

**What is it?**

The USBDriveController program is an application usable in Windows, Linux or Mac that allows the downloading of the camera movies and pictures from the Sony Handycams. It has several arguments available to pass by command line in order to achieve the desired result. One significant aspect of the application is the “safe” mode, which simply implies that it uses a method of detection for cameras that is failsafe. This means it should detect properly on any computer capable of using either the Unix file-searching functions or the Windows API. The “non-safe” modes are not dangerous to use, but may result in the program terminating early if the user has insufficient privileges on the given machine. Further notes on the use of the program are in the Appendix.

**How to use it**

The following list represents a generalized use of the program in the order it would occur:

Plugging in the USB hub with cameras connected to it will bring up a screen on each camera that requires you to press the Computer button. This allows the drives to be detected and mounted.

Run “USBDriveController.exe report”; this displays a list of connected devices and also a total number of devices detected — this is very useful in making sure all of the drives are responsive at the time of issuing commands

a) Run “USBDriveController.exe –v –p –e” to erase all camera files, for example

b) Run “USBDriveController.exe –v –p C:\Temp\” to copy all of the camera files to C:\Temp\

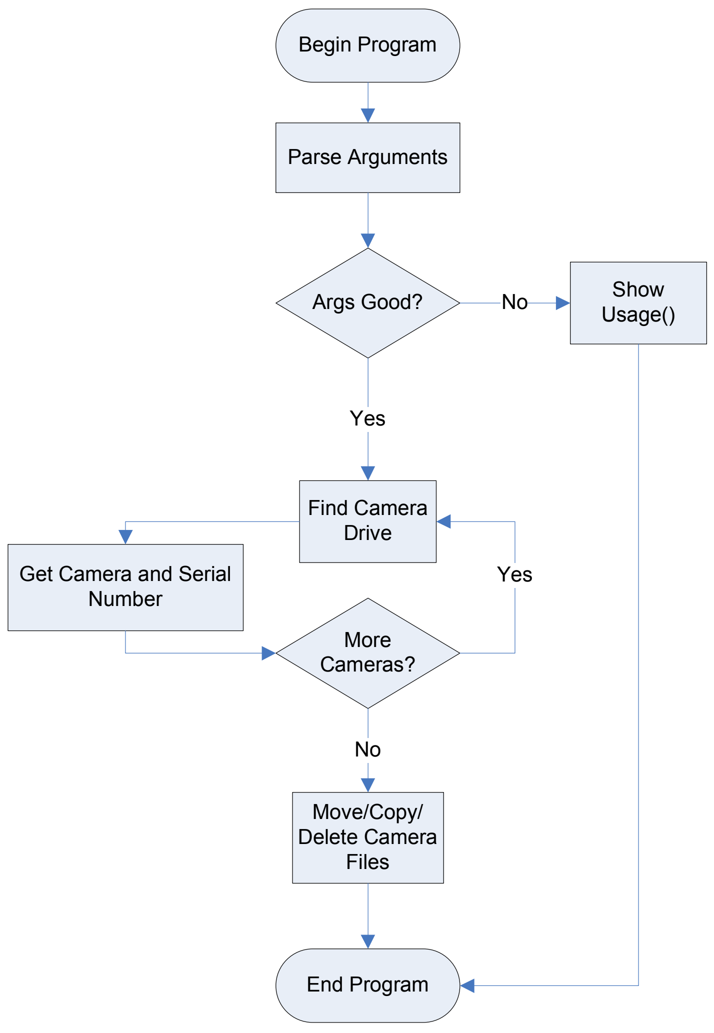
c) Run “USBDriveController.exe –v C:\Temp\ -m” to move all of the video files to C:\Temp\

d) Run “./DriveController –i” to initialize all cameras with CameraXX files in Linux or Mac

e) Run “./DriveController –v –l –s /tmp/capture/” to download the last video file to the folder “/tmp/capture/” and do it using the safe method (looking for CameraXX files)

## Appendix

### Flow Chart



### Code Notes

The program starts off by parsing the arguments supplied by the user. If bad or insufficient arguments are found, the usage function is displayed and the program ends. Upon receipt of proper arguments, the program will attempt to find all the cameras. In Windows, the program looks through all available drives; if they appear to be removable, it will double-check if they are in fact a Sony Handycam. In Linux, Mac or Windows, it finds the cameras in the same way outlined in the sections found near the end of this document. Refer to these for an idea of how the rest of the detection works in Linux, Mac or Windows. The camera would use either safe or regular mode of searching for the cameras and, by doing this, would get the drive letter or mount location and camera serial number, and then, by cross-referencing with the camSerial array, the camera number. This would take place until all mounted drives or drive letters had been checked. In the special case where “report” was supplied, the cameras would then be displayed along with their mount points and the program would end, but for any other set of valid arguments we would continue. These arguments would set values and flags, all stored into the controller class named USBAccess. It would then be decided to call some of the functions of USBAccess depending on the flags. All of the delete, copy or move operations take place by looping through to either operate on one particular camera or go through all the list of cameras that were defined in the find-camera stage. The facilitation of moving or deleting files is done through the use of the system() function that lets the operating system complete these tasks. This is a safer method than doing the file management ourselves. Additional messages are supplied depending on the value of the verbose flag set in the argument parsing stage.

**Slash Commands**

| | |
|---|---|
| Usage in Windows: | USBDriveController.exe [<options>...] <path> |
| Usage in Linux/Mac: | ./DriveController [<options>...] <path> |

Valid options:

| | |
|---|---|
| -h \| –help | Outputs a help message similar to this |
| -i \| --init | Adds CameraXX file on Camera HD for all available cameras<br>Useful so that people could later use cameras using a safe mode.<br>Basically only facilitates the use of safe mode – no other benefits.<br>Cannot be specified with --safe command as well, as this would leave no way to find out which cameras corresponded to which numbers |
| -s \| --safe | Detects cameras using CameraXX file on Camera HD<br>Theoretically on the order of maybe 50 microseconds faster than non safe mode<br>Cannot be used with --init. See comment for --init. See details in Safe Mode Notes for more information on the different modes |
| -v \| --video | Processes the videos from the cameras<br><br>By default without –e or –m this downloads the videos (must specify a path). If –e is specified, this will mean to erase video files. Additionally, if –m is specified, files will be moved off the cameras |

| | |
|---|---|
| -p \| --picture | Processes the pictures from the cameras By default without –e or –m, pictures are downloaded (path must be specified). If –e is specified, this will mean to erase picture files. Also, if –m is specified, files will be moved off the cameras |
| -m \| --move | Move the files to the computer (no copy on the camera) |
| -c \| --camera <camera-numbers...> | Specifies which cameras will be downloaded (default = all) e.g., "–c 5,4,3,6" will do operations on cameras 5, 4, 3 and 6 |
| -e \| --erase | Erases the pictures/videos instead of downloading them |
| -ri \| --reset-index | Deletes video index info (causes camera to ask to reconstruct) Resets new camera videos back to 00000.mts No equivalent for picture files Requires a physical button press on the viewfinder Camera reconstruction takes only a few seconds |
| -l \| --last | Specifies to download/move/erase the last camera video file (does nothing for pictures). Names the files camXX.mts where XX is the camera number. |
| -vb \| --verbose | Displays more status information when files are moved or deleted |

**Safe Mode on All**

The safe-mode option basically uses the most standard form of file searching to facilitate detection of cameras. In Linux/Mac, this means using opendir() calls, and on Windows it uses FindFirstFile(). This mode should be used when your computer has limited privileges and you cannot use the regular mode. The beauty of this is that, if all else fails, a camera can be initialized manually by adding a CameraXX file to the hard drive; then detection will work.

**Non-Safe Mode on Linux/Mac**

This mode involves looking at the /media/.hal-mtab file in order to make a connection between the drive's mounted location (e.g., /media/disk-1) and its operating system code (e.g., sdc1). This operating system code is then used to look through the /dev/.udev/db folder to find a file that contains serial number information, which is then cross-referenced with the list of serial numbers supplied to determine the camera number.

A guide for adding a new camera is shown below. The exact same steps are taken in finding the cameras automatically with a non-safe mode.

**Non-Safe Mode on Windows**

All of the drives on the machine are looked through and it is determined whether they are removable or not. If they are removable, the drive letter is used in the following fashion. In HKEY_LOCAL_MACHINE\System\MountedDevices, a value named \DosDevices\E: (or whatever the drive letter happens to be) is looked at, and a string containing the operating system code is parsed out (looks like `7&4353453&0`). This is then searched for in the keys under the main key HKEY_LOCAL_MACHINE\System\CurrentControlSet\Enum\USBSTOR\Disk&Ven_Sony&Prod_Camcorder&Rev_1.00\. Once we find that the value ParentIdPrefix contains the operating system code, we know that whatever folder ParentIdPrefix was in represents the serial number for that camera (folders look like SERIALNUMBER&0).

A guide for adding a new camera is shown below, and the exact same steps are taken in finding the cameras automatically with a non-safe mode.

**Replacing an Old Camera**

Replacing an old camera simply requires you to find the serial number of the new camera, find the location in the camSerials array where the old serial number is located, and replace the old with the new.

**Adding a New Camera**

If a new camera was added, the counter-values and maximum camera number #define would need to be increased. The camera serial number would also need to be discovered and added to the camSerials array. To find this serial number, you need to do one of the following:

*For Windows:*

Plug the camera into the computer and click the computer button on the camera

Determine the drive letter assigned to the new camera (e.g., E:)

In the start menu, select Run

Type regedit and click ok

Navigate to the registry key HKEY_LOCAL_MACHINE\System\MountedDevices

Depending on the drive letter, we need to look into the value named:
\DosDevices\E: (in our case)

The value of this key is in hex. Along the right side, there is translation to ASCII. Reading this, we see there is a section that resembles this form: `7&4353453&0,` or something similar (form is X&XXXXXXX&X). This is the Windows operating system code for this device.

Now navigate to the registry key
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Enum\USBSTOR\Disk&Ven_Sony&Prod_Camcorder&Rev_1.00\

Under this key, we have subkeys of the form XXXXXXXXXXXX&0, where XXXXXXXXXXXX is the serial number of the camera

Look through each subkey and see which one contains a value named ParentIdPrefix, which contains the operating system code we found earlier (e.g., 7&4353564&0)

This subkey’s name is then the serial number + &0

An optional method would be simply to search the registry in the folder containing subkeys with serial number names and search for the operating system code; however, scrolling through them seems fast enough

Now, simply add this serial number to the camSerials array under the array index camSerials[k], where k is the camera number you want assigned to this camera

*ForLinux/Mac:*

**Some distributions have a properties option under the drives right click menu where serial numbers are sometimes available under a tab named “Details” or similar. I highly suggest using this method as it is immensely easier than the following one – which is the method used by the application.**

Open the /media/ folder and make note of the number of disks currently there and their names (e.g., disk-1 or disk-2)

Plug the camera into the computer and click the computer button on the camera

Find out what folder was created for that camera in the /media/ folder

Navigate to the folder /media/ in a terminal and open the file .hal-mtab. It will contain a few lines; one will say the path, such as /media/disk-3. If this is the path you found earlier, look at the part of that line that has a value similar to sdc1 or another similar four-letter value

Now navigate to the /dev/.udev/db folder and, using grep, find a file that contains that four-letter value you found in the last step

Open the file you found

It should contain a line starting with ID_SERIAL_SHORT. This should be followed immediately by the camera’s serial number

Now, simply add this serial number to the camSerials array under the array index camSerials[k], where k is the camera number you want assigned to this camera

## Troubleshooting

The most obvious problem that will occur will be due to the location that the operating system mounts its drives. I have allowed it to search through /Volumes/ or /media/ on Mac and Linux respectively. A new location would need to be programmed in. This change would effect the member safeGetSerialNumber() and the constructor for the Camera class. It would also effect the FindCameras() member of the USBAccess class.