Initializing uC/FS with NAND

Initialization

µC/FS comes with four different media device types: NAND, NOR, SD Card and RAMDisk. The startup procedure is somewhat similar for using any of the four types in your File System project. We will focus on the NAND setup in the document.

When choosing which of the four devices to use it's important to consider your application. NAND typically makes the most sense in products that require a large amount of storage space such as USB external Hard Drives, digital cameras or MP3 players for instance. If you application is going to utilize a NAND media device this will assist you in getting started.

The typical initialization sequence for using a NAND implementation of uC/FS is as follows:

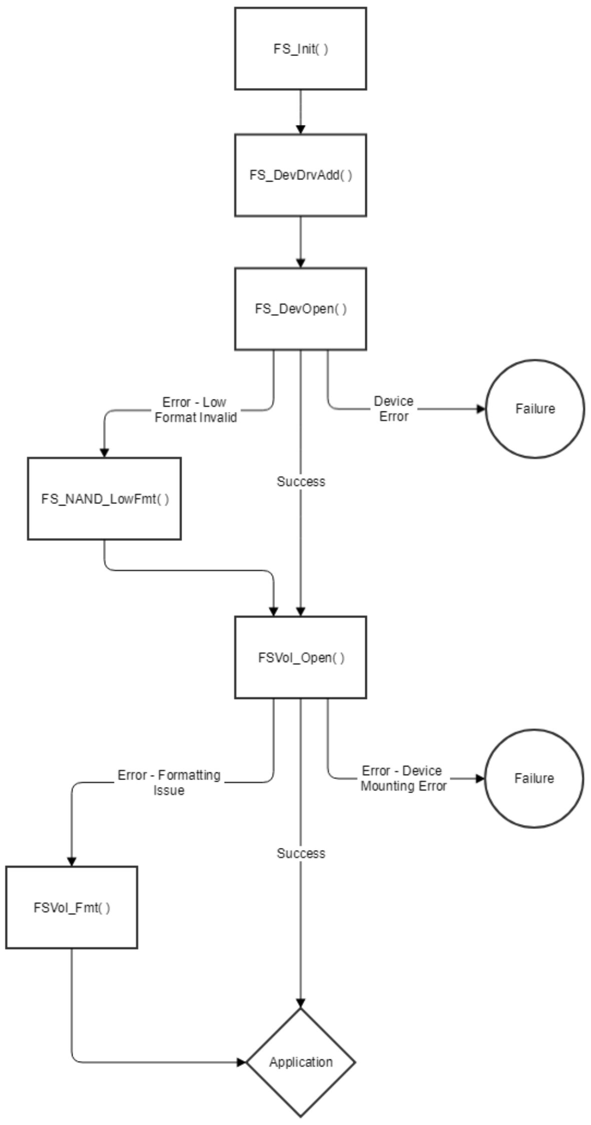

Figure 1 - FS NAND Initialization Sequence

The first step is calling FS_init( ) which will require you to give it a configuration structure. This defines many of the specifications needed to configure the File System. This should only be called ONCE in the application.

After a successful initialization the next step is to add in the desired driver for your specific device. In this situation you would add the NAND device driver to your initialized FS stack. This specifies all the NAND API's the stack will need to perform NAND operations correctly.

After successfully installing the device driver configurations it's time to open the device for use. There is a configuration structure that will contain the part specifications. This needs to be configured prior to opening the device. Depending on if it's a static device, ONFI etc. you will define this configuration as such. Once this has been properly configured you will use this to proceed with opening the device in your application. There are several outcomes you will face next:

- You may get an error return indicating a formatting issue. This is a condition that should be handled in your application. If a formatting error results from trying to open the device, your application should then perform a low-level format with the low-level format API function.
- The device open may be executed without error, in which case your application can process directly on to the next phase of initialization.
- If your application receives a communication device error, such as improperly configured I/O's or a device timeout it will need to abort the setup and try again if the device is still starting up. Or it may be an irrecoverable malfunction.

Once the device has been properly opened, the volume will need to be opened for use. This can be done with a call to the volume open function show in the above diagram. The volume open function will require the identification string of the opened device in order to open the volume. If the open occurs successfully you can move on to utilizing the uC/FS in your application. There are several situations that can arise from this function as well:

- The most straight-forward outcome is if the return value indicates a successful volume open. Then the application can begin using the FS as mentioned previously.
- If there is a device mounting error this could indicate that the device itself is missing. So while the configuration may be correct for the driver and device expected, it may not physically be there.
- The other outcome is a formatting error occurs. The device may be previously formatted differently than your application expects it to be. In this case you would need to call the FS volume format function. Similar to the low-level format but for the volume. After a successful re-formatting of the volume your application can begin to use the uC/FS.