Code Documention

In case this helps.... I have some rationale behind what I wrote..... which works for me ;)

a) the WiFi interface has to read from the ESthing all the crap and headers etcetera that describe an incoming message on a particular client number. The checkForOK function is written the way it is so that the constant strings remain in progmem and dont consume RAM. Thats why you cant remove the F().

b) once it has an incoming string WiFiInterface sends it in one of 3 directions (DCCEXparser, WiThrottle or HTMPParser)

c) These parsers afre sent the command string and a MemStream for the reply. The freason the reply must be collected in a memstream is that we can't start sending the client a reply until we know how long the reply is!

BEWARE - The memstream is using the SAME buffer as the command to build up the reply... So the parsers must read the command fully before writing to the memstream or else the command will be corrupted.

d) when the parser returns, it takes the accumulated reply from the memstream and sends it back to the originating client.

(this part is actually blocking and should really be implemented as part of the Finite State Automation in the wifi loop().)

e) because the response is sent as soon as the parser returns, this doesnt really allow for callbacks because the stream has gone by the time the callback arrives. To solve this problem one must EITHER ban all multiple connections to the Wifi or implement a memstream that can be flushed and continued and hold RAM buffers for each connected client simultaneously.

The stringFormatter::send routine does essentially what a printf would do but allows us to extend the capability by including %S for example so that we dont have to do all the crap of copying constant strings into RAM.

I also use a new %E feature which means I can print things like a Wifi bufer with visible \r\n characters which makes debugging so much easier.

@grbba Why do you need to create strings like <R 1 1 1> and pass them to the DCCEXparser... ? Why not just call the DCC:: api directly, that's what it's for.

(Unless of course you're creating the strings from the far side of your queue. )

when I wrote the code that split the <...> parameters into p[0]...p[n] every command only had integer parameters. To implement <1 PROG> I adjusted the parrameter splitter to generate hashes of non-integer parameters... I added the <#> command into the CVReader bespoke filter (since renamed <$>) so I could discover the numbers created for any keyword e.g. try this <$ 1 2 3 PROG MAIN>

If you imagine that CVReader.ino is a bespoke sketch that uses all the rest of the code in a library, you can see that you can create a bespoke filter in your own sketch and ask the library to invoke it.

The filter is in your sketch because its bespoke. The library calls it because you asked it to. Yes the libraray has that code... but it has NO reference to the filter so it will compile just fine without one. If you create a filter you must call setFilter to tell the libraray you want a callback. This is where the parser calls the custom filter:

if (filterCallback) filterCallback(stream,opcode,params,p);