Trace Messaging Facility -Version 3

There are two reasons for trace info:
-status reporting during normal operation
-debug tracepoints when troubleshooting unexpected program flow or data values

There are two parts to the generation of traces:

1) Trace commands embedded in the software at strategic places that indicates that a message should be generated if execution gets to this point, and provide the data that goes into the message

2) A trace message generation routine (which the trace command calls) that generates the message based on the information in the trace command, and dynamic configuration information stored in a table. This table can be displayed and overwritten by the MQTT TR command so that tracing can be changed on the fly

Tracepoint information in the trace statement embedded in the code includes the following. Note that various techniques can be used to streamline the trace calling process, and this, and other information may not be explicitly in the trace statement.

- string containing the function name
- trace type:
  H: high level status/info
  M: medium level status/info
  L: low level status/info
  W: warning on unexpected but tolerable condition
  E: serious exception has occurred, preventing normal operation
- optional text string providing data label or situation description
- optional variable name for data to be provided with the trace

Dynamic trace processing controls are all stored in one table, writable via MQTT.

- Table entry 1 is a special case, containing global default trace enabling for the 5 trace types
  - example: enable error and warning traces are enabled everywhere

The 5 types of tracing are represented by binary bits
H = bit 0 = decimal 1 = symbol t$H
M = bit 1 = decimal 2 = symbol t$M
L = bit 2 = decimal 4 = symbol t$L
W = bit 3 = decimal 8 = symbol t$W
E = bit 4 = decimal 16 = symbol t$E

In technical terms:

$traceTab[1] is global enables, with individual bits. If particular trace is to be enabled globally, the corresponding bit is set in this entry. If the bit is zero, the trace is disabled by default, but can be enabled at the routine level. An example value would be

```
$traceTab[t$global] = t$E + t$W      // globally enable error and warning traces
```

- Table entry 2 is also a special case, which controls where trace output goes to:
  - serial monitor (enabled by bit 0 = decimal 1 = symbol t$SM)
  - MQTT topic (enabled by bit 1 = decimal 2 = symbol t$MQ)
  - both (enabled by t$SM + t$MQ)

To configure this entry so that trace output goes to the serial monitor, you would define:
    $traceTab[t$routing] = t$SM   // direct trace output to the serial monitor

- The third and following entries in the table provide trace enabling information for one specific function. Note that the function may have multiple traces and different trace types.) Functions have a unique numeric identifier which corresponds to its position in the trace control table. As a result of this, function ID's start at 3, avoiding the 2 special purpose table entries. The entry for a given function tells what trace types are enabled for it, using the same bit meanings given above for the global enables in table entry 1.

These trace enables are additive: a trace is generated if either the global enables, or the routine's enables allow it, for the type of trace being considered.

The trace table $traceTab is defined in the configDetails.cpp file, and the relevant part looks like this:

```
  // first 2 table entries are special purpose ones

$traceTab[t$global] = t$H + t$M + t$E + t$W ;  //error & warning messages globally enabled
  $traceTab[t$routing] = t$SM ;     // direct traces messages to serial monitor

  // following table entries specify trace enables for individual routines

  $traceTab[3] = t$L + t$W + t$E ;  // routine 3 = setupFlows() in flows.cpp
  $traceTab[4] = t$E + t$W;         // routine 4 = mapDegToPWM in flows.cpp
  $traceTab[5] = t$E + t$W;         // routine 4 = showCfgDetails in configDetails.cpp
  $traceTab[6] = t$M ;              // routine 6 = processCmd in mqttBroker.cpp
```

??????????????????
 correct following section, document the "add a trace" process, define H M L conventions, discuss impact on software execution, review Notes section at end, add overloaded trace forms.
??????????????????

The generated trace message looks like this:

<type> <function-name><trace-sub-ID>(<numeric trace ID>): <data-label-info><data-value>

type = S, W or E
function-name = name of C++ function-name
trace-sub-ID = numeric identifier for traces within same function, starting at 1
numeric trace ID = assigned number for this function name, globally unique. Indexes the config table.
data-label-info = label for acompanying data value, or description of cause of trace message
data-value = debug or status data added to the message

example message: E functionB-1(7): g_X out of range: 18.5

Example code sequence:

```
void loop()
{
  functionA(10,20,30)
  functionB(5.7, 7.9, -10.60)
}


void functionA(x,y,z)
{
    trace(“functionA”, 1, ID6, top, status, “Z value= “, Z)
    float r = sqrt(x*x + y*y +z*z)
    trace(“functionA”, 2, ID6, low, status, “R= “,r)
    g_det = B * B – 4 * a * c    // from quadratic formula
    if (g_det<0) {trace(“functionA”, 3, ID6, top, error, “negative determinant= “, g_det)}
    return
}


void functionB(hipx, hipy, hipz)
{
    trace(“functionB”, 1, ID7, top, status, “starting”, hipx)
    g_X = arcsin( hipy/hipx) * g_hfr_width
    if( g_X > maxW || g_X < maxW)
    {
        trace(“functionB, 1, ID7, error, “g_X out of range: “,g_X)
        // recover in some way...
        g_X = 0
    }
    return
}
```

Example output on next page.

```
with this table setup:                | you would see traces like this for one loop()
                                      |
[1]  0    /no global enables          |  S functionA-1(6): Z value= 30
 ...                                  |  S functionA-2(6): R= Z value= 37.42
[6] Bit1                              |     / never encountered functionA-3 tracepoint
[7]                                   |
```

```
with this table setup:                | you would see traces like this for one loop()
                                      |
[1]  Bit1     /all traces enabled     |  S functionA-1(6): Z value= 30
 ...                                  |  S functionA-2(6): R= Z value= 37.42
[6]  *        /don't care             |  E functionB-1(7): g_X out of range: 18.5
[7]  *        /don't care             |
```

```
with this table setup:                | you would see traces like this for one loop()
                                      |
[1]  0                                |
 ...                                  |
[6]  *                                |  E functionB-1(7): g_X out of range: 18.5
[7] Bit10 + Bit13 +Bit16              |                          |
 /top+err, med+err, low+err
```

```
with this table setup:                | you would see traces like this for one loop()
                                      |
[1]  Bit7    /all traces with errors  |
 ...                                  |  E functionA-3(6): negative determinant= -87.2242
[6]  *        /don't care             |  E functionB-1(7): g_X out of range: 18.5
[7]  *        /don't care             |
```

Notes:
- this facility will distort program execution timing because it takes time to generate and transmit the trace message. It would be nice to extend it to allow setting / clearing of a small number of LEDs via a trace command, which can be done very quickly.
- the requirement to not require any arguments has not been met, and it's not clear to me how this would be possible. The argument count can be reduced by defining specialized macro's that essentially pre-populate some of the fields.
- there's some implied work to write 2 new MQTT commands:
  - one to write numeric values into the configuration table for on-the-fly trace changes
  - one to display the current configuration table, for convenience
- there's some implied work to allow the config table to be set up at compile time with a usable setup.
- this trace facility isn't a replacement for the abbreviated print commands like sp2l(a,b) which have a different purpose, primarily in debugging.
- we could assign bits in the config table to direct individual trace messages to the serial monitor, MQTT, or both, but I'm not sure this is needed.