A note on the L network
The L network is built from series switched inductors wound on toroids and parallel switched capacitors. A relay switches the shunt capacitors between the input or output side of the inductors. With the large numbers of inductors, capacitors and relays a fair bit of physical space is needed to accommodate them making it impossible to avoid long leads plus the associated contact resistance, stray inductance and capacitance. The net result of this being that you never see exactly the value of the inductor/s or capacitor/s which have been switched in but instead their values plus strays. Further errors in the reactances occur due to toroid values being limited to one turn boundries and the difficulty of obtaining high voltage capacitors leading to having to accept values which do not have a binary progression. To some extent capacitors can be built from series and parallel connection e.g. nominal 5 pF can be built from 4 x 22pF in series and 10pF from 2 in series and finally 1 x 22pF giving 3 nice binary steps but for the next one the closest practical choice is 47pF or 2 paralled 22pF. There is a practical limit on how many capacitors can be paralled together to build larger values as 704pF require 32 x 22pF capacitors in parallel and 1408pF require 64.

Simple tuning algorithms operate under the constraint of any reactance (capacitor or inductor) when switched into the circuit on its own must be greater than the sum of the reactances preceeding it. e.g. L5 must be greater than L1+L2+L3+L4 etc. This is necessary so that when the software commands an increase or decrease of a reactance it must assume this did occur and the opposite did not in fact happen. However due to strays and tolerences mentioned the chances of this happening are unreliable at best and when it occurs the tune will stop with best match assumed when in fact continuing will give further improvement.

An algorithm has been developed to look ahead and see if a better match exists beyond the current indicated best match and continue if so.

The coarse tune algorithm
In order to quickly get at a ball park figure and establish a starting point for a fine tune, a coarse tune algorithm is applied. This simply switches each C relay in turn starting at 0 relays up to the 8th relay and at each step, stepping each L relay in turn again from 0 to 8 and recording the SWR in a 2 dimensional array. There will be a total of 81 readings (0 to 8 inclusive for C, by 0 to 8 inclusive for L). The array is examined to retrieve the best SWR and the relay combination is recorded.

The process is done with the capacitors connected to the antenna or output side of the inductors (HiZ) and repeated with the capacitors connected to the input side of the L network (LoZ). The best SWR from each of the two sets of readings is returned and the best of the two determine the starting point of the fine tune and whether we are HiZ or LoZ.

```
doRelayCoarseSteps(): Caps are connected to Output (HiZ)
    Cap   L0        L1        L2        L3        L4        L5        L6        L7        L8
    C0    6.62025   7.48000   8.42857   9.80000  25.42857  44.55555  68.69230  71.66666  73.80000
    C1   81.88888   7.36842   8.19444   9.63636  26.33333  44.55555  68.53846  66.76920  73.80000
    C2   73.80000   7.25974   8.19444   9.50746  24.44827  42.15789  68.69230  66.76920  73.80000
    C3   82.11111   6.92592   7.76315   9.35294  23.66666  42.15789  68.53846  66.23076  82.11111
    C4   82.11111   6.38636   7.19512   8.69863  22.25000  40.10000  68.53846  66.23076  74.00000
    C5   73.80000   5.22429   5.88000   7.37209  18.17948  34.82608  63.71428  66.23076  74.00000
    C6   61.50000   2.94535   3.40963   4.40000   9.51940  18.85714  49.33333  71.83333  67.00000
    C7   67.72727   1.50267   1.54166   2.10752   5.41441  13.59259  49.00000  66.38461  67.72727
    C8   82.77777  30.28571  29.23529  36.29411  99.00000  71.54545  74.33333  71.83333  74.60000
bestC = 7; bestL = 1; SWR =   1.50267
```
fig 1

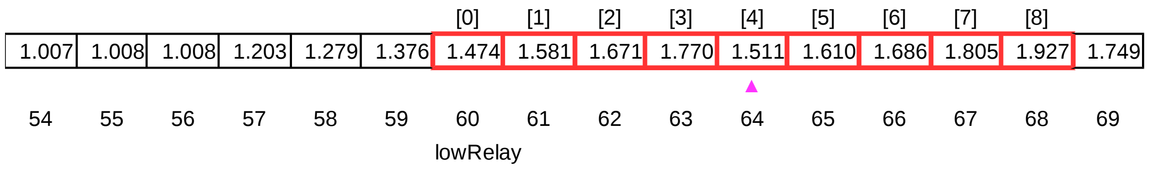
The chart in fig 1 is a printout of the values obtained from sweeping the capacitor and inductor relays. As can be seen there are some pretty wild swings of SWR over the relay ranges but there is always one spot which is better than the others. In this case it occurs when C relay number 7 is operated along with L relay number 1 giving an SWR of 1.50267. This is a close enough point for us to perform a fine tune without too many relay steps to get our match. These values and the following fineTune values are actual vaues returned from the debug output of my tuner while tuning my 40 Metre antenna.

The fine tune algorithm

A snapshot of the SWR's returned by operating the consecutive relays steps around the chosen best relays from the coarse tune subroutine is shown in fig 2. The coarse tune subroutine has returned the best single Inductor and Capacitor relay combination. Holding the Inductor relay fixed the fine tune subroutine is going to traverse the relays around the “best so far” capacitor relay to try to improve the SWR. In the example shown the operated capacitor relay was #7 corresponding to a value of 64 in a binary count (1, 2 , 4, 8, 16, 32, 64, 128) i.e. 7th relay gives a count of 64.

The values array is loaded with the SWR's from the 4 capacitor relay steps each side of the initial relay so it forms a window on the SWR values returned from consecutive relay steps. The value lowRelay is set to hold the relay combination which produced the SWR held in values[0] and is used as a reference to the relay combination needed to get to any array position.

Examining the VSWR data it can be seen that the value returned from the coarse tune subroutine (held in array[4]) is not the best SWR which is actually at relay combination 54 in fig 2. The program examines the array to get the SWR trend and returns a count of where the best SWR is which occurs at position, values[0] in this case. If count is less than the reference position i.e. less than (values[4]), the array window is shifted down and if more it is shifted up. In each case the shift is tested to ensure that the array won't extend beyond 0 relay count or 255 relay count.



						[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	
1.007	1.008	1.008	1.203	1.279	1.376	1.474	1.581	1.671	1.770	1.511	1.610	1.686	1.805	1.927	1.749
										▲					
54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
						lowRelay									

fig 2

The reason for the complexity in this algorithm is that due to tolerance spreads in inductor and capacitor values the total reactance does not smoothly progress as the relays increment or decrement. This particularly shows up when stepping from values like 63 to 64 where several relays change as 64 = relay 7 operated but 63 = relays 1+2+3+4+5+6 operated. If the SWR values in fig 2 were simply examined each side of relay 64 looking to see if they were greater or less, it would suggest that the value at 64 was the dip. However this is not the case and the true low SWR is at relays 49 (off the scale shown in fig 2) so the best SWR would not have been obtained by simply examining the relay each side of the initial best value. By looking further on each side, it is possible to accurately pick up an SWR trend in spite of the capacitor or inductor tolerances.

The window of SWR values held in the values[9] array is simply moved in the direction of best swr until array[4] holds the lowest SWR and the relay combination to produce this becomes the tune setting. The effect of traversing the relays is shown in fig 3 and it can be seen that as the lowRelay is stepped down the chain the SWR's in values[4] is falling. The value “cnt” refers to the array position holding the smallest SWR and if less than 4 we traverse down and if greater than 4 we traverse up. The limit is reached when the lowest SWR occurs at values[4] and we are then tuned.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	Low	cnt
1.3762	1.4737	1.5810	1.6706	1.7702	1.5106	1.6102	1.6864	1.8050	59	0
1.2791	1.3762	1.4737	1.5810	1.6706	1.7702	1.5106	1.6102	1.6864	58	0
1.2026	1.2791	1.3762	1.4737	1.5810	1.6706	1.7702	1.5106	1.6102	57	0
1.0078	1.2026	1.2791	1.3762	1.4737	1.5810	1.6706	1.7702	1.5106	56	0
1.0076	1.0078	1.2026	1.2791	1.3762	1.4737	1.5810	1.6706	1.7702	55	0
1.0074	1.0076	1.0078	1.2026	1.2791	1.3762	1.4737	1.5810	1.6706	54	0
1.0073	1.0074	1.0076	1.0078	1.2026	1.2791	1.3762	1.4737	1.5810	53	0
1.0071	1.0073	1.0074	1.0076	1.0078	1.2026	1.2791	1.3762	1.4737	52	0
1.0069	1.0071	1.0073	1.0074	1.0076	1.0078	1.2026	1.2791	1.3762	51	0
1.0067	1.0069	1.0071	1.0073	1.0074	1.0076	1.0078	1.2026	1.2791	50	0
1.0066	1.0067	1.0069	1.0071	1.0073	1.0074	1.0076	1.0078	1.2026	49	0
1.2028	1.0066	1.0067	1.0069	1.0071	1.0073	1.0074	1.0076	1.0078	48	1
1.4203	1.2028	1.0066	1.0067	1.0069	1.0071	1.0073	1.0074	1.0076	47	2
1.5128	1.4203	1.2028	1.0066	1.0067	1.0069	1.0071	1.0073	1.0074	46	3
1.5778	1.5128	1.4203	1.2028	1.0066	1.0067	1.0069	1.0071	1.0073	45	4

fig 3

After tuning the capacitors, the same process is repeated for the inductors and then back to the capacitors again etc. until no further improvement can be obtained. It should be noted that sometimes changing say the inductors, will cause the capacitors to back up a little over changes just made to them and vice versa but eventually equilibrium is reached.

The Code

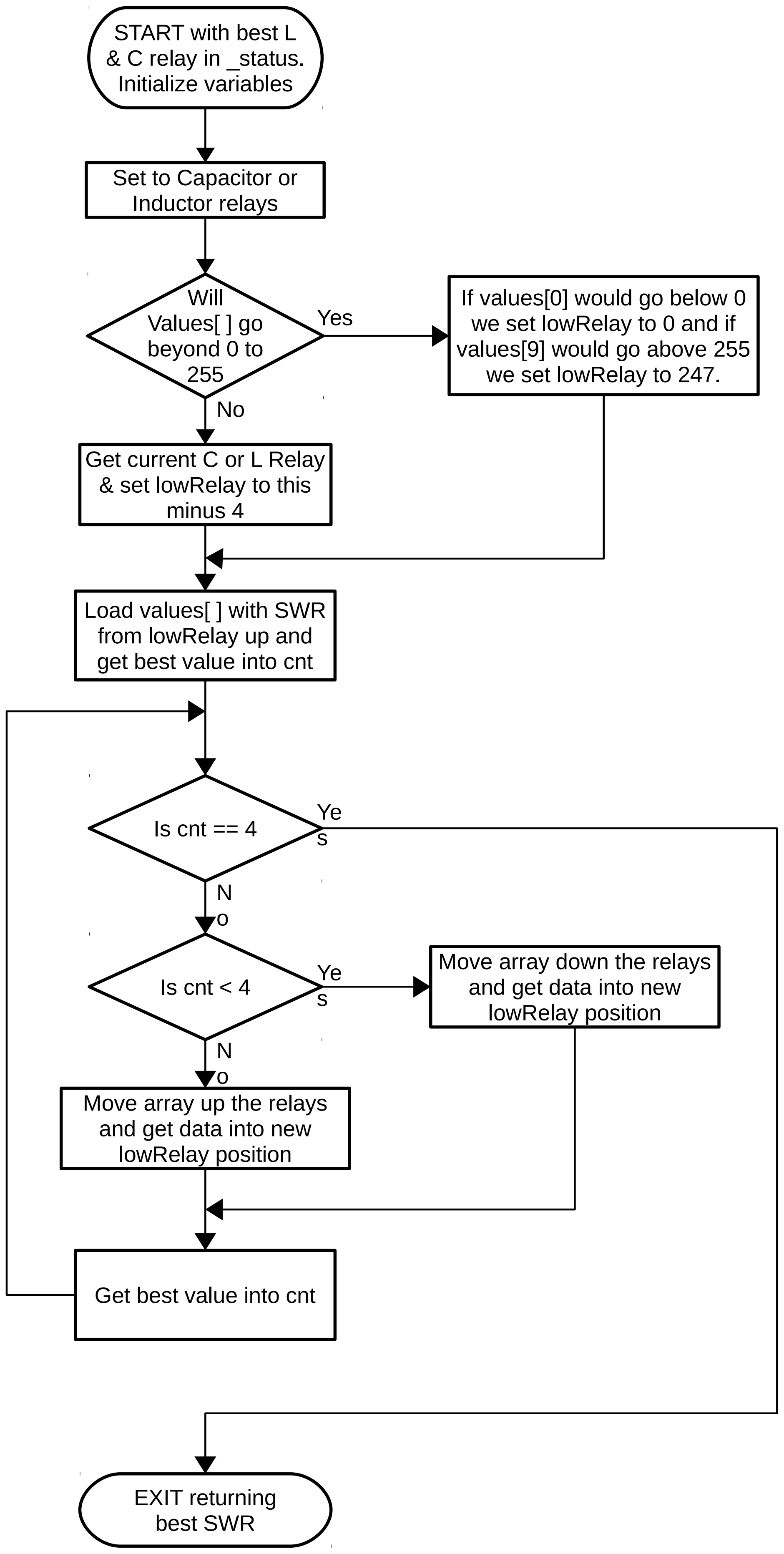
I have included the Coarse and fine tune subroutines plus the function to find the best value in the array. A flow chart of the fine tune routine is also included.

```
/
*********************************************************************************
**************************/

void doRelayCoarseSteps()
{
  // This subroutine steps through the capacitor and inductor relays looking for the combination
which gives
  // the lowest SWR. Only individual relays are stepped with no multiple C or L combinations so a
fine tune
  // subroutine is later called to set exact values for L and C.

  // This procedure is carried out by initially setting capacitor relays to zero C and stepping the
inductor
  // relays one by one, incrementing the capacitor and repeating the procedure.
  // The SWR is read at each step into an 2 dimensional array which is later parsed for the lowest
SWR and
  // the C and L combination to give this is set along with rawSWR in the _status array.

  // Entry: The caller sets the C/O relay to HiZ or LoZ as required
  // Exit with relay settings which give best SWR for the C/O relay setting on entry.

  unsigned long values[9][9];
  unsigned long bestSWR = 99900000;
  byte bestC = 0;
  byte bestL = 0;
  char buffer[16];
  // Initialise with no L or C relays operated and C/O relay set by the caller.
  _status.C_relays = 0;
  _status.L_relays = 0;
  setRelays();
//  delay(20); // Extra settling time for an ultra stable initial reading
  getSWR();  //Get SWR with relays at initial state.
  bestSWR = _status.rawSWR;

  for(byte c =0; c < 9; c++) {
    if(c != 0) {
      _status.C_relays = 0;
      bitSet(_status.C_relays, c - 1);
    }
    for(byte x = 0; x < 9; x++) {
      if(x != 0) {
        _status.L_relays = 0;
        bitSet(_status.L_relays, x - 1);
      }
      setRelays();
      getSWR();
      values[c][x] = _status.rawSWR;
    }
  } //endfor

   // We parse the array looking for the combination of L and C relays which give lowest SWR
  for(byte c =0; c < 9; c++) {
    for(byte x = 0; x < 9; x++) {
      if(values[c][x] < bestSWR) {
        bestSWR = values[c][x];
        bestC = c;
        bestL = x;
      }
    }
  }

  // Now set the relays to give best coarse tune based on bestSWR
  _status.C_relays = 0; // No bits set for no relays operated
  _status.L_relays = 0;
  if(bestC > 0) bitSet(_status.C_relays, bestC - 1); // Set bits 0 .. 7 here (Relays 1 to 8)
  if(bestL > 0) bitSet(_status.L_relays, bestL - 1);
  setRelays();
  delay(20); // Extra settling time for an ultra stable final reading
  getSWR();  //Get SWR with relays at final state.

#ifdef DEBUG_COARSE_TUNE_STATUS // Print the DEBUG header
  Serial.print(F("doRelayCoarseSteps(): Caps are connected to "));
  if (_status.outputZ == hiZ) Serial.print(F("Output (HiZ)"));
  else Serial.print(F("Input (LoZ"));
  Serial.println();
  Serial.print("Cap\t");
  for(int x = 0; x < 9; x++) {
    Serial.print(F("  L"));
    Serial.print(x);
    Serial.print(F("       "));
  }
  Serial.println();

  // Now print the DEBUG body which is the debug data captured in the array
  for(byte c =0; c < 9; c++) {
    Serial.print(F("C"));
    Serial.print(c);
    Serial.print("\t");
    for(byte x = 0; x < 9; x++) {
      sprintf(buffer, "%3lu",values[c][x] / 100000);
      Serial.print(buffer);
      Serial.print(F("."));
      sprintf(buffer, "%-05lu ",values[c][x] % 100000);
      Serial.print(buffer);
      Serial.print(F(" "));
    }
    Serial.println("");
  }
  Serial.print(F("bestC = "));
  Serial.print(bestC);
  Serial.print(F("; bestL = "));
  Serial.print(bestL);
  Serial.print(F("; SWR = "));
  sprintf(buffer, "%3lu",_status.rawSWR / 100000);
  Serial.print(buffer);
  Serial.print(F("."));
  sprintf(buffer, "%-05lu ",_status.rawSWR % 100000);
  Serial.println(buffer);
#endif //DEBUG_COARSE_TUNE_STATUS

} //end subroutine

/
*******************************************************************************
*************************/

uint32_t fineStep(bool reactance) // Enter with swr and relay status up to date
{
  // On entry, we look at the current SWR and the SWR from the 4 relay steps each side to see if
there is a trend and
  // in which direction. The SWR from the 9 relays is read into an array which is examined to see if
we need to step up
  // or step down. A check is made to ensure that stepping the relays won't overflow 255 or
underflow 0 then the array
  // is traversed up or down consecutive relays until the best SWR is centred at values[4]. The
associated relays are
  // set in _status array. Parameter reactance determines whether we are stepping _status.C_relays or
_status.L_relays.
```

```
  uint32_t values[9]; // An array of SWR values centred around the relays set in "_status" at entry
  uint8_t lowRelay;   // The relay combination which gives the SWR held in Values[0] (0 to 255)
  uint8_t cnt;  // Mostly used to point to a position in the array
  uint8_t *pReactance; // A pointer to either _status.C_relays or _status.L_relays
  boolean header = true; // Used to decide whether to print the data table header on debug

  if(reactance == INDUCTANCE) {  // set to operate on either _status.C_relays or _status.L_relays
    pReactance = &_status.L_relays;
  } else {
    pReactance = &_status.C_relays;
  }

  // Load the array with the SWR values obtained from the current "_status_X_Relays" and the
relays 4 above & below.
  // Check to see that "uint8_t lowRelay" stay in bounds, i.e does not become less than 0 or greater
than 255.
  if((*pReactance  >= 4) && (*pReactance <= 251)) {  // Do only if lowRelay won't over or
underflow
    lowRelay = *pReactance - 4;
  } else {
    if(*pReactance < 4) { // lowRelay could go out of bounds here so limit its value
      lowRelay = 0;
    } else lowRelay = 247;
  }
  // Loading the array with SWR's from 9 relays centred around current relay
  cnt = 0;
  for(int x = lowRelay; x < (lowRelay + 9); x++) {
    *pReactance = x; // Select the relay starting from lowRelay and stepping up over a total of 9
relays.
    setRelays();     // and operate it
    getSWR();
    values[cnt] = _status.rawSWR;
    cnt++;
  }       // On exit, _status.X_relays = lowRelays + 8; cnt = 9

  displayAnalog(0, 0, _status.fwd);
  displayAnalog(0, 1, _status.rev);

  cnt = findBestValue(values, 9); // Get the array position holding the lowest SWR

// Print the contents of the initialised array and if it is Inductor or Capacitor relays being stepped.
#ifdef DEBUG_FINE_STEP
    Serial.print(F("fineStep: Values on entry using "));
    if(reactance == INDUCTANCE) {
      Serial.println(F("INDUCTORS"));
    } else {
      Serial.println(F("CAPACITORS"));
    }
    printFineValues(printHeader, values, cnt, lowRelay);
    printFineValues(printBody, values, cnt, lowRelay);
#endif
```

```
  // Assume if cnt < 4, we need to search down but not if lowRelay at 0 or we will underflow
  // If cnt = 4 we have found the SWR dip
  // If cnt > 4, we need to search up but not if lowRelay at 247 or more else we will overflow

  while(cnt != 4) {
    if(((lowRelay == 0) && (cnt < 5)) || ((lowRelay == 247) && (cnt > 3))) {
#ifdef DEBUG_FINE_STEP
      Serial.println(F("We will over/underflow so choosing best value"));
#endif
      break;
    } // ---------------------------------------------------
    else if(cnt < 4) { // We won't over/underflow and need to search down
#ifdef DEBUG_FINE_STEP
  if(header) Serial.println(F("cnt < 4 so searching down"));
#endif
      lowRelay--;
      for(uint8_t i = 8; i > 0; --i){ // Shift the array to the right 8 steps
        values[i]=values[i-1];
      }
      *pReactance = lowRelay;
      setRelays();
      getSWR();
      delay(5);
      getSWR();
      values[0] = _status.rawSWR;
    } // ---------------------------------------------------
    else { // We won't over/underflow and need to search up
#ifdef DEBUG_FINE_STEP
  if(header) Serial.println(F("cnt > 4 so searching up"));
#endif
      lowRelay++;
      for(uint8_t i=0; i < 8; i++){ // Shift the array to the left 8 steps
        values[i]=values[i+1];
      }
      *pReactance = lowRelay + 8;
      setRelays();
      getSWR();
      delay(5);
      getSWR();
      values[8] = _status.rawSWR;
    } // ---------------------------------------------------
    cnt = findBestValue(values, 9);
#ifdef DEBUG_FINE_STEP // Print a table of SWR values in the array at this point
  if(header) {         // Do a header if first time through the "while" loop
    printFineValues(printHeader, values, cnt, lowRelay);
    header = false;
  }
  printFineValues(printBody, values, cnt, lowRelay);
#endif
    displayAnalog(0, 0, _status.fwd);
    displayAnalog(0, 1, _status.rev);
  } // End while ============================================
  // Set up the _status struct with the relay pointed to by lowRelay plus the cnt offset which
  // will correspond to the lowest SWR for this pass of fine tune.
  *pReactance = lowRelay + cnt;
  setRelays();
  setRelays();     // Extra relay switching for an accurate final reading
  delay(20);   // Extra relay settling time for an accurate final reading
  getSWR();

  displayAnalog(0, 0, _status.fwd);
  displayAnalog(0, 1, _status.rev);

#ifdef DEBUG_FINE_STEP
    Serial.print(F("fineStep: Values on exit using "));
    if(reactance == INDUCTANCE) {
      Serial.println(F("INDUCTORS"));
    } else {
      Serial.println(F("CAPACITORS"));
    }
    printStatus(printHeader);
    printStatus(printBody);
#endif
  return _status.rawSWR;
}

/
******************************************************************************
*************************/

void printFineValues(boolean doHeader, uint32_t values[], uint8_t cnt, uint8_t lowRelay)
{
  int x;
  char buffer[16];

  if(doHeader) {
    for(x = 0; x < 9; x++) {
      Serial.print(F("Values["));
      Serial.print(x);
      Serial.print(F("]  "));
    }
    Serial.print(F("lowRelay "));
    Serial.println(F("cnt"));
  } else {
    for(x = 0; x < 9; x++) {
      Serial.print(float(values[x]) / 100000, 4);
      Serial.print(F("     "));
    }
    Serial.print(lowRelay);
    Serial.print("\t    ");
    Serial.println(cnt);
  }
}
```

```
/
*****************************************************************************************
**************************/

uint8_t findBestValue(uint32_t values[], uint8_t cnt)
// Parses an array of uint32_t values, whose length is equal to cnt. The position in the array (0 to
cnt-1)
// containing the smallest value is returned. If all positions are of equal value then position 0 is
returned.
// If more than one position contains the lowest value then the first position containing it is returned.

{
  uint32_t bestValue = 0;
  uint8_t bestPosition = 0;

  bestValue--;  // Initialize by rolling back to set to maximum value

  for (uint8_t x = 0; x < cnt; x++) {
    if(values[x] <= bestValue) {
      bestValue = values[x];
      bestPosition = x;
    }
  }
  return bestPosition;
}

/
*****************************************************************************************
**************************/
```


START with best L & C relay in _status. Initialize variables
Set to Capacitor or Inductor relays
Will Values[ ] go beyond 0 to 255
Yes
If values[0] would go below 0 we set lowRelay to 0 and if values[9] would go above 255 we set lowRelay to 247.
No
Get current C or L Relay & set lowRelay to this minus 4
Load values[ ] with SWR from lowRelay up and get best value into cnt
Is cnt == 4
Yes
No
Is cnt < 4
Yes
Move array down the relays and get data into new lowRelay position
No
Move array up the relays and get data into new lowRelay position
Get best value into cnt
EXIT returning best SWR