Table of Contents

Preface

This document lists many examples of VDF-files and accompanying explanations which can be helpful for your own application. It starts with simple examples of changing a single setting, shows advanced use with if-else like assignments and ends with how to redefine your pump profile.

The easy way to check what can be adapted is to look into the SMB tab and go through the loop input sections:

- Constraints
  They are checked in other parts of AAPS and are out of reach. However, when you change some other setting these hard limits are not checked by the emulator. That means if you feel you need to lift those hard limits in your private version of AAPS you can run the emulator with larger values to find your personal hard limit.

- Glucose status (called **`glucose_status`**)
  These data come from the CGM, be it directly, after smoothing or further processing like bg-deltas or bg-acceleration. They should not be adapted.

- Current temp

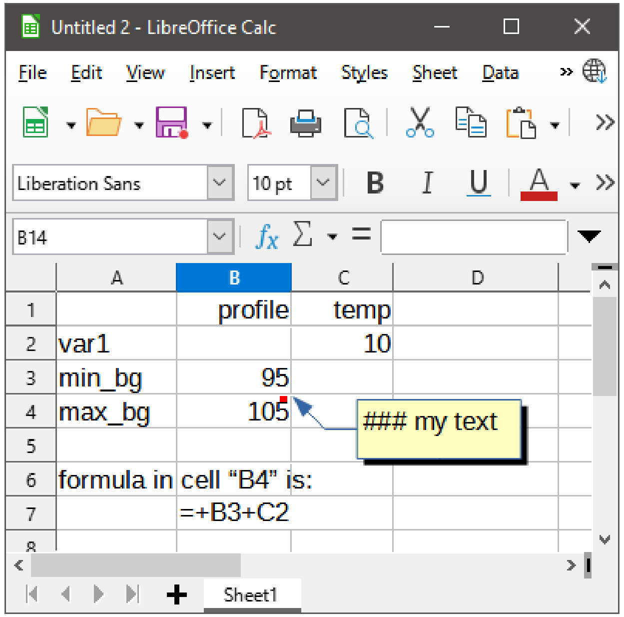
- IOB data (called **`iob_data`**)
  The calculation of IOB is not part of the code which was migrated to the emulator and is therefore out of scope.

- Profile (called **`profile`**)
  Most of the AAPS settings are part of this AAPS profile. Do not get confused with the pump profile, that is just an unlucky wording. You can see all the possible items listed here. Again, some of the names inside the profile section differ from the well known terms. Therefore here the unusual but relevant elements:
  - min_bg		lower BG target
  - max_bg		higher BG target
  - target_bg	average of the 2 above; this gets overwritten inside AAPS and is therefore not needed or even misleading
  - carb_ratio	normally known as CR
  - sens		normally known as ISF

- Meal data (called **`meal_data`**)
  The calculation of IOB is not part of the code which was migrated to the emulator and is therefore out of scope.

- Autosense data (called **`autosense_data`**)
  The calculation of IOB is not part of the code which was migrated to the emulator and is therefore out of scope.

Layout and general hints



The VDF-file entries have 4 logical groups or columns:	You can think of it as a spreadsheet:
1. group name like `profile`	a column heading
2. element within the group like `min_bg`	a row heading
3. assigned new value like 95	a cell content like value or formula
4. optionally comment field like ### my text	a comment attached to the cell

The example given above will look like this:

```
profile             min_bg              95                                      ### my text
```

The examples may be used via copy/paste. In your own typing be careful with the single quote character like below in `profile['min_bg']`. In word processing tools there are other characters that look similar like a backwards single quote. All those lead to errors. If in doubt copy it from here.

The group temp

This is a special group besides profile etc. and holds your own interim variables with your own names. Especially when you create long and complex expressions it can hold interim results and simplify reading and debugging. In the example to the right the variable “var1” was added with a value of 10 and later referred to in cell “B4”. In VDF formulation it looks like this:

```
temp                var1                10                                      ### interim value
profile             min_bg              95                                      ### new lower target, fixed
profile             max_bg              profile['min_bg'] + temp['var1']        ### new upper target, expression
```

Profile – adapting settings in regular AAPS

Adapting targets

This is a very frequent situation. Let us start with alternative methods how to set a lower target:

```
profile           min_bg         95                              ### a simple, fixed assignment
```

This will set the lower target to 95 no matter what it was in the master run. It is your responsibility to ensure this is not higher than the upper target value.

If we want to be more advanced we may use an expression rather than a fixed value like this:

```
profile           min_bg         profile['min_bg'] + 10          ### a simple, relative assignment
```

This will set the lower target to10mg/dl higher than what it was in the master run. The expression `profile['min_bg']` refers to the current value and this formalism with square brackets and single quotes can be used in analogy for all other current settings.

If we want to be careful not to exceed 99mg/dl as our lower target we can expand it further:

```
profile           min_bg         min(99,profile['min_bg']+10)     ### use the python function ”min”
```

Varying the upper target works in analogy. There is one special case, namely mimicking a TempTarget. To achieve that we need to define lower and upper targets, ensure they have the same value and set the flag for the target being a TempTarget. It goes like this:

```
profile           min_bg         min(99,profile['min_bg']+10)     ### use the python function ”min”
profile           max_bg         profile['min_bg']                ### copy from lower target
profile           temptargetSet  True                             ### enable the flag
```

You may have wondered why we do not declare the intended `target_bg` likewise and directly. In fact that is always calculated as average of `min_bg` and `max_bg` inside AAPS and the emulator anyway. If we have a TempTarget active in the master run we may simply disable it with a 1-liner:

```
profile           temptargetSet  False                            ### disable the flag
```

Adapting ISF, CR and basal

In the previous section we learned that there was no command to directly set a TempTarget. Likewise there is no command to directly set a percentage change of the profile but we can set them individually. To make the case more challenging and building on what we did for the targets let us assume we want to mimic a 10% increase in resistance. Which we normally achieve by setting the pump profile to 100%. Here is the VDF that does it:

```
profile          sens             profile['sens']      / 1.1         ### numerically reduce ISF by 10%
profile          carb_ratio       profile['carb_ratio'] / 1.1        ### numerically reduce CR by 10%
profile          current_basal    profile['current_basal']*1.1       ### numerically increase basal by 10%
```

Of cause we can just set one of them on its own. Let us assume we want to modify ISF based on `delta`:

```
temp             slope            glucose_status['delta']            ### holds BG delta
profile          sens             profile['sens'] - temp['slope']    ### stronger if BG climbs, weaker if BG falls
```

Now let us assume we only want such an adaptation if BG changed its trend recently. As change in BG trend we use the difference between delta now and short average delta and check that this difference is more than ± 3mg/dl. In the VDF such logical tests result in “1” if True and “0” if False which makes it possible to get different results for different conditions. Have a look:

```
temp             acce            glucose_status['delta']-glucose_status['short_avgdelta']   ### estimated acceleration
temp             OK              abs(temp['acce']) > 3                       ### “1” if acce > ± 3mg/dl, “0” else
temp             slope           glucose_status['delta'] * temp['OK']        ### holds BG delta if True, “0” else
profile          sens            profile['sens'] - temp['slope']             ### stronger if BG climbs, weaker if BG falls
```

In cases of such complex definitions it is a must to check that these expressions evaluate as intended. The LOG-file echoes every result of the VDF-file line by line. Here are extracts from the LOG-file showing the 3 possible outcomes:

```
loop execution in row=7635 of logfile l:\not_me\Norbert\AndroidAPS._2022-04-20_10-31-49_.2.zip at= 2022-04-20T09:05:52.560Z
appended new entry to temp with acce=-2.3899999999999997
appended new entry to temp with OK=False
appended new entry to temp with slope=0
edited old value of 20.159999999999997 in profile with sens=20.159999999999997

loop execution in row=10770 of logfile l:\not_me\Norbert\AndroidAPS._2022-04-20_10-31-49_.2.zip at= 2022-04-20T09:10:33.309Z
appended new entry to temp with acce=3.33
appended new entry to temp with OK=True
appended new entry to temp with slope=2
edited old value of 20.159999999999997 in profile with sens=18.159999999999997
...
loop execution in row=4979 of logfile l:\not_me\Norbert\AndroidAPS._2022-04-20_10-31-49_.3.zip at= 2022-04-20T09:35:17.920Z
appended new entry to temp with acce=3.5600000000000005
appended new entry to temp with OK=True
```

```
appended new entry to temp with slope=-5
edited old value of 20.159999999999997 in profile with sens=25.159999999999997
```

At time 09:05UTC the acceleration is between -3 and +3, so not applicable. The flag variable OK is False and sensitivity is unchanged.
At time 09:10UTC the acceleration is 3.33, so applicable. The flag variable OK is True and sensitivity is reduced because slope is positive.
At time 09:35UTC the acceleration is 3.56, so applicable. The flag variable OK is True and sensitivity is increased because slope is negative.

This is just a theoretical example of the capabilities. You can try your own approaches. Believe it or not, many features of autoISF I first developed by testing them in VDF before being programmed in the apk itself.

Adapting a pump profile

In the VDF you can also define your circadian tables for ISF, CR and basal. You need to pay special attention to the difference between UTC time and your local time zone for the pump. Here is an example for CET (Central European Time):

```
STAIR_ISF           00:00:00Z   45          ### 01h_C(entral)E(uropean)T(ime) or 02h_CEST
STAIR_ISF           01:00:00Z   44          ###
STAIR_ISF           02:00:00Z   42          ###
...
STAIR_ISF           17:00:00Z   36          ### 18h_CET
STAIR_ISF           18:00:00Z   38          ###
...
STAIR_ISF           22:00:00Z   43          ### 23h_CET
STAIR_ISF           23:00:00Z   44          ### 00h_CET
profile     sens                STAIR_ISF   ###

STAIR_CR            00:00:00Z   8.0         ### 01h_C(entral)E(uropean)T(ime) or 02h_CEST
STAIR_CR            01:00:00Z   7.5         ###
...
STAIR_CR            20:00:00Z   7.5         ###
STAIR_CR            21:00:00Z   8.0         ###
STAIR_CR            22:00:00Z   9.0         ### 23h_CET
STAIR_CR            23:00:00Z   9.0         ### 00h_CET
profile     carb_ratio          STAIR_CR    ###

STAIR_BAS           00:00:00Z   0.41        ### 01h_C(entral)E(uropean)T(ime) or 02h_CEST
STAIR_BAS           01:00:00Z   0.43        ###
STAIR_BAS           02:00:00Z   0.44        ###
STAIR_BAS           03:00:00Z   0.50        ###
...
STAIR_BAS           19:00:00Z   0.75        ### 20h_CET
STAIR_BAS           20:00:00Z   0.75        ###
STAIR_BAS           21:00:00Z   0.60        ###
STAIR_BAS           22:00:00Z   0.45        ### 23h_CET
STAIR_BAS           23:00:00Z   0.43        ### 00h_CET
profile     current_basal       STAIR_BAS   ###
```

Some lines were omitted in that example for better readability. The lines must be sorted by UTC time. Therefore the first line (winter time) or first two lines (summer time) from your CET/CEST based pump definition must be cut off and appended at the end. If your profile is not fully populated for 24 hours this may also mean you first need to create a pump entry at 01 or 02 hours, respectively.

Before adding anything else to the VDF you should already do a test run and check the LOG file to ensure that the assignments were correct. So here is the above example for the first hours of the day. After 2:30UTC Autosense was active which modifies the settings and therefore blurs the picture.

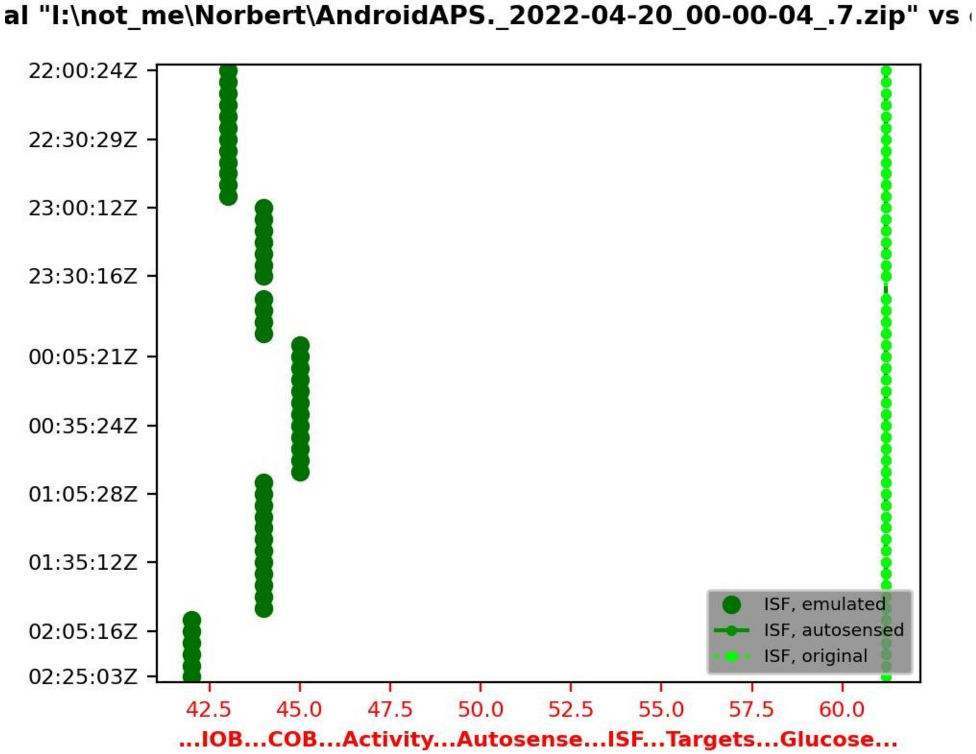
On the right you see the echo of the VDF assignments for the first loop in that time window. The loops time is just after 22:00 UTC. You can check which value was defined in each of the 3 STAIR-types and that the correct values were picked for ISF, CR and basal rate.

Below you see the graphic result for ISF. To make the graph easy the only output options were “ISF/PDF”. This suppresses all other items you would get with the standard option “All/-pred/-flowchart”. The option “PDF” allows to grab the graph in PNG or JPEG format before it gets saved in the PDF. The original ISF is shown in light green, the emulated ISF in darker green.

```
========== Echo of what-if definitions actioned for variant pumpProfile
========== created on Thu, 20 Apr 2023 18:38:33 +0200
========== for loop events found in logfile
l:\not_me\Norbert\AndroidAPS._2022-04-20_00-00-04_.0.zip

loop execution in row=970 of logfile
l:\not_me\Norbert\AndroidAPS._2022-04-20_00-00-04_.0.zip at=
2022-04-19T22:00:24.327Z
appended new entry to STAIR_ISF with 00:00:00Z=45
appended new entry to STAIR_ISF with 01:00:00Z=44
appended new entry to STAIR_ISF with 02:00:00Z=42
not actioned: [...], [], []
appended new entry to STAIR_ISF with 17:00:00Z=36
appended new entry to STAIR_ISF with 18:00:00Z=38
not actioned: [...], [], []
appended new entry to STAIR_ISF with 22:00:00Z=43
appended new entry to STAIR_ISF with 23:00:00Z=44
edited old value of 61.199999999999996 in profile with sens=43
not actioned: [...], [], []
appended new entry to STAIR_CR with 00:00:00Z=8.0
appended new entry to STAIR_CR with 01:00:00Z=7.5
not actioned: [...], [], []
appended new entry to STAIR_CR with 20:00:00Z=7.5
appended new entry to STAIR_CR with 21:00:00Z=8.0
appended new entry to STAIR_CR with 22:00:00Z=9.0
appended new entry to STAIR_CR with 23:00:00Z=9.0
edited old value of 10 in profile with carb_ratio=9.0
not actioned: [...], [], []
appended new entry to STAIR_BAS with 00:00:00Z=0.41
appended new entry to STAIR_BAS with 01:00:00Z=0.43
appended new entry to STAIR_BAS with 02:00:00Z=0.44
appended new entry to STAIR_BAS with 03:00:00Z=0.50
not actioned: [...], [], []
appended new entry to STAIR_BAS with 19:00:00Z=0.75
appended new entry to STAIR_BAS with 20:00:00Z=0.75
appended new entry to STAIR_BAS with 21:00:00Z=0.60
appended new entry to STAIR_BAS with 22:00:00Z=0.45
appended new entry to STAIR_BAS with 23:00:00Z=0.43
edited old value of 0.74 in profile with current_basal=0.45
```

Normal text file



Handling automations

Regarding automations there are two aspects:

How to get rid of automation actions that were present in the master run but may disturb the alternative approach by autoISF

If you used automations to adapt the profile then check the previous page how you would revert the actions and reset the profile to your base defined in the pump.

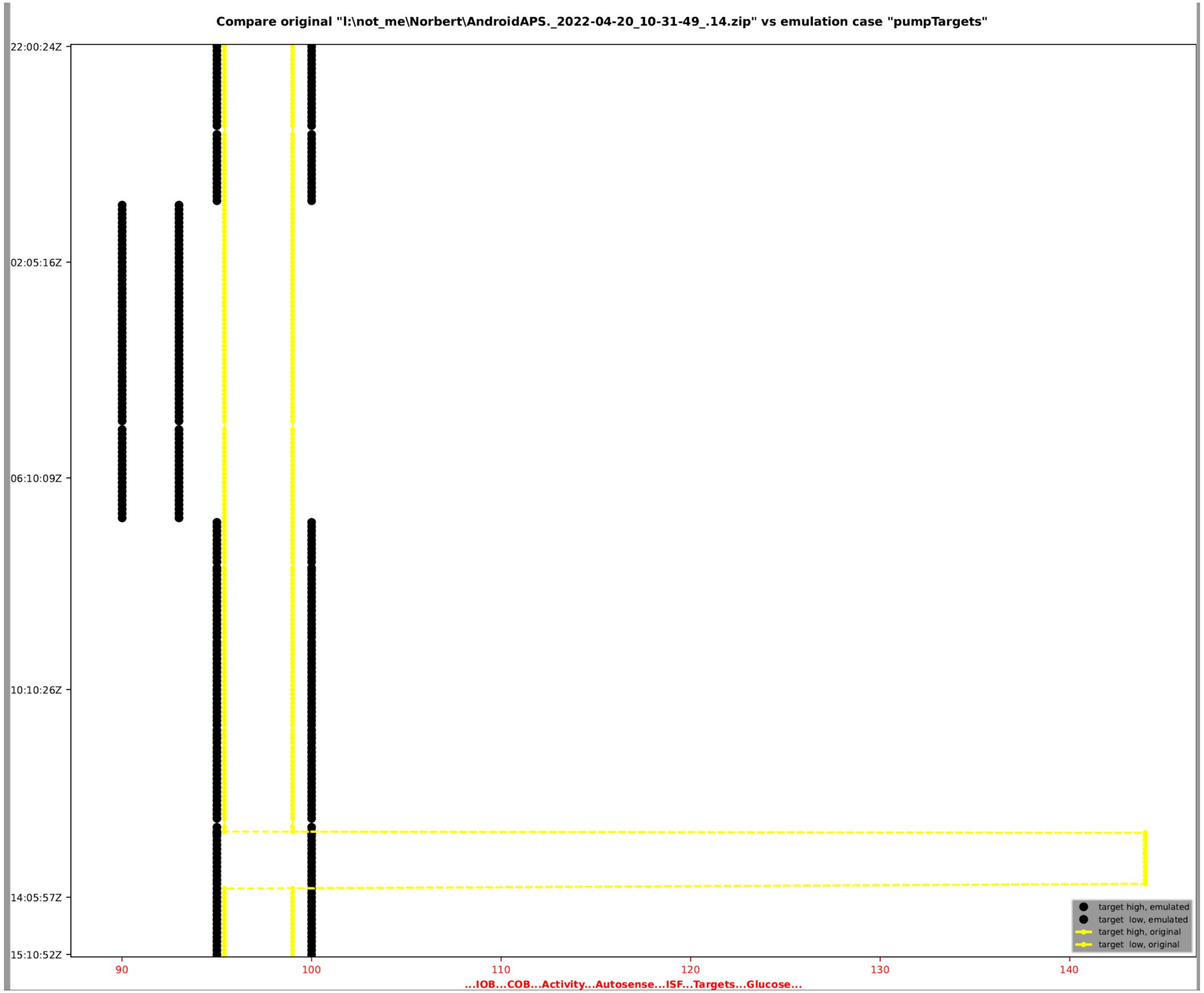
If you used automations to change targets then consult page 3 of this document which has several examples of how to set targets. When you emulate a longer time window during which the pumps profile has a target change then you can use a method analogous to the previous page: Here is an example of setting the lower target to 90mg/dl between 02 :00 and 08:00 CET and 95 otherwise. The higher target is more than the lower just for demonstration purposes and to show the possibilities even if I always use a bandwidth of zero:

```
STAIR_LTG        00:00:00Z   95          ### 01h_C(entral)E(uropean)T(ime) or 02h_CEST
STAIR_LTG        01:00:00Z   90          ###
STAIR_LTG        07:00:00Z   95          ### 08h_CET
STAIR_LTG        23:00:00Z   95          ### 00h_CET
profile          min_bg      STAIR_LTG   ###

STAIR_HTG        00:00:00Z   100         ### 01h_C(entral)E(uropean)T(ime) or 02h_CEST
STAIR_HTG        01:00:00Z   93          ###
STAIR_HTG        07:00:00Z   100         ### 08h_CET
STAIR_HTG        23:00:00Z   100         ### 00h_CET
profile          max_bg      STAIR_HTG   ###
```

Here is what the resulting PDF looks like when restricted to just output ”target”. You can clearly see the interval during the night with lowered targets. In the afternoon the user had set a temporary exercise target which gets overwritten.

With these methods you can cancel all profile and target changes introduced by automations. You then have a neutral playing field to define methods of autoISF to address whatever effect you had in mind with the automations. You can scale the various weights such that the insulin delivered or withheld is roughly the same amount as in the automation approach. After activating those features in live autoISF you can then fine-tune the settings in a follow-up emulator study.


Compare original "I:\not_me\Norbert\AndroidAPS._2022-04-20_10-31-49_.14.zip" vs emulation case "pumpTargets"
22:00:24Z
02:05:16Z
06:10:09Z
10:10:26Z
14:05:57Z
15:10:52Z
90
100
110
120
130
140
...IOB...COB...Activity...Autosense...ISF...Targets...Glucose...
target high, emulated
target low, emulated
target high, original
target low, original

How to emulate automations inside the emulator

The condition can be checked like we saw in the section about adapting ISF by assigning the result of a logical check to a `Temp` variable. There are conditions which are only available in the emulator like all the autoISF settings and others only in AAPS. Examples are:

```
temp        iobTH5     iob_data['iob'] > 5                  ### ”1” if IOB>5,  ”0” otherwise for checking an IOB  threshold
temp        Ttset      profile['temptargetSet']             ### ”1” if set,    ”0” otherwise
temp        tGT100     profile['min_bg'] > 100              ### ”1” if lower target>100, ”0” otherwise
temp        bg140      glucose_status['glucose'] > 140      ### ”1” if BG>140, ”0” otherwise
```

For declaring an AND condition you just multiply the related flags, e.g. to check whether TT is set and is above 100:

```
temp        TTgt100    temp['TTset'] * temp['tGT100']       ### ”1” if TT>100, ”0” otherwise
```

For declaring an OR condition you add the related flags and check whether it is above 0:

```
temp        anyone     temp['iobTH5']+temp['bg140'] > 0     ### ”1” if IOB>5 or BG>140, ”0” otherwise
```

The action of the emulated automation can be any setting the emulator can handle like seen in the examples in the preceding sections.

Other special commands

There are three special cases of quasi arrays that were introduced to handle interim or time varying assignments:

STAIR

This is the original and more general form of the commands seen earlier for defining the pump profile. The format example is

```
STAIR     2020-04-14T03:00:00Z  120                 ### value starting at 3am UTC, equals 05:00 CEST to prepare for exercise
STAIR     2020-04-14T05:00:00Z  150                 ### value starting at 07:00 CEST, the start of early morning exercise
STAIR     2020-04-14T06:00:00Z  profile['max_bg']   ### back to regular value starting at 08:00, end of morning exercise
profile   min_bg                STAIR               ### time varying value
profile   max_bg                STAIR               ### time varying value, same as min_bg
```

This is useful for any time dependant values. The drawback is that it works only on that given day. May be more useful for air travel passing time zones than in the given example.

INTERPOL

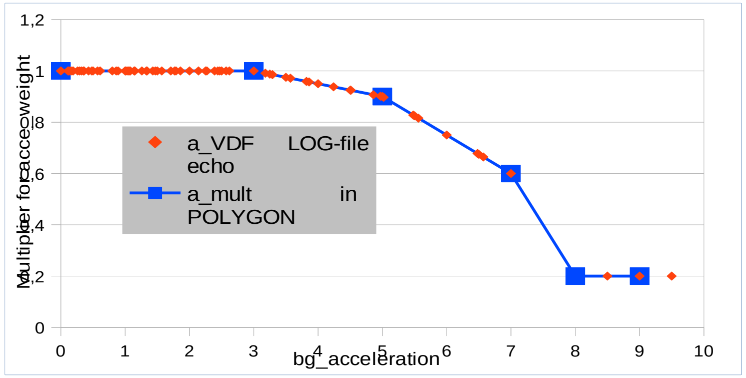
This uses linear interpolation and extrapolation to define a time dependent function. The format is

```
INTERPOL    2020-04-14T20           100
INTERPOL    2020-04-14T22           140
profile     min_bg                  round(INTERPOL)-20        ### time varying value
```

POLYGON

This command is the most general as it provides a list of value pairs. The first element of the pair is any existing value like bg, the second element of the pair is a result at that input value. Like in the case of the command INTERPOL values outside the input range are extrapolated, values inside are interpolated.. This is far too theoretical to be comprehended easily, so lets us look at an example which I plan to investigate anyway in the near future. The background of the example has to do with glycaemic index of snacks. A hypothesis could be that a small but fast snack leads to

a_in	a_mult in POLYGON
0	1
3	1
5	0,9
7	0,6
8	0,2
9	0,2



higher accelerations than a regular, larger meal with fat and fibre. This snack would trigger a larger bolus than appropriate for the low total carbs taken. So to dampen the acce_ISF impact in such cases the weight could be reduced for higher accelerations. A starting point could be this table of acceleration versus damping factor *a_mult*. In VDF format the example table is defined in the first six lines:

```
POLYGON     0                        1.0              ### no change, constant for extrapolation at low end
POLYGON     3                        1.0              ### no change, for extrapolation at low end
POLYGON     5                        0.9              ### slight linear decrease between 3 and 5
POLYGON     7                        0.6              ### stronger linear decrease between 5 and 7
POLYGON     8                        0.2              ### even stronger linear decrease between 7 and 8
POLYGON     9                        0.2              ### constant, for extrapolation at high end

temp        a_in                     glucose_status['bg_acceleration']              ### acce input
temp        a_VDF                    POLYGON(temp['a_in'])                          ### a_mult multiplier for acce weight
profile     bgAccel_ISF_weight       profile['bgAccel_ISF_weight'] * temp['a_VDF']   ### modulated acce weight
```

Line 8 defines a temporary variable *a_in* as a short hand name. Line 9 then looks up the resulting multiplier *a_VDF* based on the current acceleration *a_in* as input. The last line then multiplies the current *bgAccel_ISF_weight* by the multiplier.

I ran a demo for all the loops of April 26. From the LOG file I extracted the values for *a_in* and the assigned *a_VDF*. The chart above includes those results overlaying the original polygon table graph. Results below 0 and above 10 were truncated from the plot window. For such an involved study it was worth the effort to produce that overlay before going ahead and looking at the resulting SMB impact.

Changing autoISF settings

All the settings for autoISF are accessible as elements in the group profile. So on that side there is nothing really new. If however you want to experiment with a new logic or add more effects you can test them by appropriate VDF commands.

The group new_parameter

This group is used for specific autoISF settings and was introduced to by-pass the internal algorithm and replace it by your own algorithm expressed by VDF formulae. So you can redefine any one of the 4 BG effects like this:

```
new_parameter    acce_ISF    my_new_acce_ISF        ### insert VDF commands upfront to calculate your own factor
new_parameter    bg_ISF      my_new_bg_ISF          ### insert VDF commands upfront to calculate your own factor
new_parameter    dura_ISF    my_new_dura_ISF        ### insert VDF commands upfront to calculate your own factor
```

One special trick in this context is to assign a value of “1” which effectively shuts off that individual contribution. That has the same effect as setting the related weight to “0”. For developing your own algorithm it is helpful to understand the basic results contained in glucose_status:

- glucose — standard AAPS glucose valid at the time the loop ran
- delta — standard AAPS delta
- short_avgdelta — standard AAPS short_avgdelta
- long_avgdelta — standard AAPS long_avgdelta
- date — time at which the above mentioned glucose was measured
- dura_ISF_minutes — duration of the glucose plateau
- dura_ISF_average — average level of glucose during that plateau window
- useFSL1minuteRaw — flag is true when Libre data in 1 minute mode are used
- parabola_fit_correlation — measure of fit quality; “1” is perfect; anything less than “0.9” is disregarded in autoISF
- parabola_fit_minutes — duration of the fit window; minimum of 10m for Libre 1-minute data; minimum of 15m otherwise
- parabola_fit_last_delta — delta between now and 5m ago, both derived from the fit formula
- parabola_fit_next_delta — delta between 5m ahead and now, both derived from the fit formula
- parabola_fit_a0 — fit polynomial coefficient, approximates the current glucose
- parabola_fit_a1 — fit polynomial coefficient, current delta, i.e. the tangent to the parabola at this time
- parabola_fit_a2 — fit polynomial coefficient, half the acceleration
- bg_acceleration — acceleration derived from fit formula

Dealing with the even/odd target switches

You may want to check whether the current target is even or not. This is also the very basis for further manipulations of making the current target even or odd. The start is a simple check whether the target is odd where simple means we are in the mg/dl world:

```
Temp         odd         profile['min_bg'] % 2             ### python modulo function: ”1” if odd, ”0” if even
```

If now we want to ensure an even target we just subtract this result from the current target. In total we have this VDF:

```
Temp         odd         profile['min_bg'] % 2             ### python modulo function: ”1” if odd, ”0” if even
profile      min_bg      profile['min_bg'] - temp['odd']   ### even MINUS 0 stays even, odd MINUS 1 becomes even
profile      max_bg      profile['min_bg']                 ### same as min_bg
```

Here there is no check whether min_bg and max_bg were identical beforehand. Since version 3.0.1 of autoISF there is no longer a need to check for TT set or not. Based on the preceding method we can ensure odd target this way:

```
Temp         odd         profile['min_bg'] % 2             ### python modulo function: ”1” if odd, ”0” if even
profile      min_bg      profile['min_bg'] -temp['odd']+1  ### adds 1 to even target from above
profile      max_bg      profile['min_bg']                 ### same as min_bg
```

For those using mmol/l units the first line for assessing even/odd becomes slightly more complicated and I had to extract and translate the maths from the master code into the first 2 lines. Also, at the end the mmol/l values need to be converted back to mg/dl because that is what is stored in profile[‘min_bg’] etc. at this stage. The complete case of ensuring odd target looks like this:

```
Temp         t_mmol      round(profile['min_bg']/18, 1)    ### convert mg/dl target back to mmol/l target
temp         odd         ((temp['t_mmol']*10) %2) /10      ### python modulo function, weighted: ”0.1” if odd, ”0” if even
temp         odd_mmol    temp['t_mmol'] -temp['odd'] + 0.1 ### odd target in mmol/l
profile      min_bg      round(temp['even_mmol']*18,0)     ### convert back to mg/dl
profile      max_bg      profile['min_bg']                 ### same as min_bg
```

QA check for an even/odd target example

Such a complex set of VDF commands should always be tested before real usage. In fact due to these tests I saw that my first version was deficient. Here we do not need a full blown emulation run. It suffices to run through those assignments and check the results by inspecting the logfile for the first loop event. So we put together a scenario of defining a series of targets and applying all the commands one after the other. Each such iteration will overwrite what we defined a moment ago before even going to the loop execution. To save us space and time the assignment of the upper target is skipped in each case. That rather long script looks like this:

```
profile     min_bg      74                              ### even in mg/dl, odd 4.1 in mmol/l
temp        t_mg        profile['min_bg']               ### interim storage
temp        t_mmol      round(temp['t_mg']/18, 1)       ### convert 74 mg/dl back to 4.1 mmol/l
temp        odd         profile['min_bg'] % 2           ###
profile     min_bg      profile['min_bg'] -temp['odd']  ### should be even, i.e. 74
profile     min_bg      temp['t_mg'] -temp['odd'] +1    ### should be odd,  i.e. 75
temp        odd         ((temp['t_mmol']*10) %2) /10    ###
temp        even_mmol   temp['t_mmol'] -temp['odd']     ### should be even, i.e. 4.0
profile     min_bg      round(temp['even_mmol']*18,0)   ### convert back to 72 mg/dl
temp        odd_mmol    temp['even_mmol'] + 0.1         ### should be odd,  i.e. 4.1
profile     min_bg      round(temp['odd_mmol']*18,0)    ### convert back to 74 mg/dl
                                                        ### separator for readability; can be completely BLANK
profile     min_bg      79                              ### odd in mg/dl, even 4.4 in mmol/l
temp        t_mg        profile['min_bg']               ### interim storage
temp        t_mmol      round(temp['t_mg']/18, 1)       ### convert 79 mg/dl back to 4.4 mmol/l
temp        odd         profile['min_bg'] % 2           ###
profile     min_bg      profile['min_bg'] -temp['odd']  ### should be even, i.e. 78
profile     min_bg      temp['t_mg'] -temp['odd'] +1    ### should be odd,  i.e. 79
temp        odd         ((temp['t_mmol']*10) %2) /10    ###
temp        even_mmol   temp['t_mmol'] -temp['odd']     ### should be even, i.e. 4.4
profile     min_bg      round(temp['even_mmol']*18,0)   ### convert back to 79 mg/dl
temp        odd_mmol    temp['even_mmol'] + 0.1         ### should be odd,  i.e. 4.5
profile     min_bg      round(temp['odd_mmol']*18,0)    ### convert back to 81 mg/dl
```

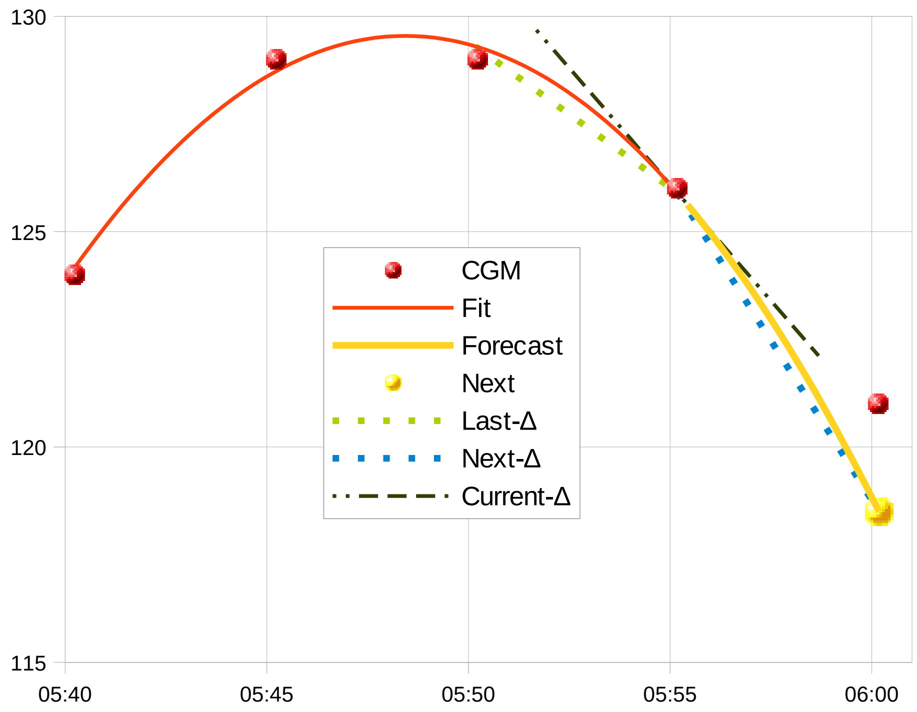
Camparing this with the echo in the logfile side by side shows agreement:

```
edited old value of 100 in profile with min_bg=74                              ### even in mg/dl, odd 4.1 in mmol/l
appended new entry to temp with t_mg=74                                        ### interim storage
appended new entry to temp with t_mmol=4.1                       18, 1)        ### convert 74 mg/dl back to 4.1 mmol/l
appended new entry to temp with odd=0                             2            ###
edited old value of 74 in profile with min_bg=74                 temp['odd']   ### should be even, i.e. 74
edited old value of 74 in profile with min_bg=75                 'odd'] +1     ### should be odd,  i.e. 75
edited old value of 0 in temp with odd=0.1                       ) %2) /10     ###
appended new entry to temp with even_mmol=3.9999999999999996     p['odd']      ### should be even, i.e. 4.0
edited old value of 75 in profile with min_bg=72.0               ol']*18,0)    ### convert back to 72 mg/dl
appended new entry to temp with odd_mmol=4.1                      0.1          ### should be odd,  i.e. 4.1
edited old value of 72.0 in profile with min_bg=74.0             l']*18,0)     ### convert back to 74 mg/dl
not actioned: [], [###], [separator for readability; can be completely BLANK  ]  ### separator for readability; can be completely BLANK
edited old value of 74.0 in profile with min_bg=79                             ### odd in mg/dl, even 4.4 in mmol/l
edited old value of 74 in temp with t_mg=79                                    ### interim storage
edited old value of 4.1 in temp with t_mmol=4.4                  18, 1)        ### convert 79 mg/dl back to 4.4 mmol/l
edited old value of 0.1 in temp with odd=1                        2            ###
edited old value of 79 in profile with min_bg=78                 temp['odd']   ### should be even, i.e. 78
edited old value of 78 in profile with min_bg=79                 'odd'] +1     ### should be odd,  i.e. 79
edited old value of 1 in temp with odd=0.0                       ) %2) /10     ###
edited old value of 3.9999999999999996 in temp with even_mmol=4.4  p['odd']    ### should be even, i.e. 4.4
edited old value of 79 in profile with min_bg=79.0               ol']*18,0)    ### convert back to 79 mg/dl
edited old value of 4.1 in temp with odd_mmol=4.5                 0.1          ### should be odd,  i.e. 4.5
edited old value of 79.0 in profile with min_bg=81.0             l']*18,0)     ### convert back to 81 mg/dl
```

Some theory regarding the best fit

For some it may help to go back to school and explain the business of polynomials, parabola, coefficients of fit, etc.
Let us start with the basic formula for a parabola, i.e. a polynomial of order 2:

$$bg(t_5) = a_2*t_5^2 + a_1*t_5 + a_0$$



where t_5 is the time measured from now and expressed in units of 5 minutes. This definition of t_5 makes it very easy to do the sums in the top of the head.

- So, the current t_5 is 0 which means $bg(0)=a_0$.
- 5 minutes ago means t_5=-1 and $bg(-1) = a_2-a_1+a_0$; etc.

For the deltas, i.e. the tangents of the parabola, we use the first derivative to get:

$$delta(t_5) = 2*a_2*t_5 + a_1$$

resulting in these examples

- currently t_5=0 and $delta(0) = a_1$, i.e. the current delta
- 5 minutes ago means t_5=-1 and $delta(-1) = a_1-2*a_2$, i.e. the last_delta

Please note that these tangents differ from the deltas calculated inside AAPS which basically are slopes between 2 data points like last-Δ is between now and 5 minutes ago.

For the acceleration we use the derivative of delta, i.e. the second derivative of bg:

$$acce\ (t_5) = 2*a_2$$

which is constant.

Best fit after Gauss is a mathematical formula for a parabola which has minimal square sum of deviations between fitted and real data. With the fit we smooth out noise in the data and need at least 4 measurements to be included. The quality of the fit is measured by the correlation coefficient. AutoISF goes back up to 45 minutes in BG history and thus increases the length of the fit window step by step. The final fit is the one with the best correlation coefficient.

Alphabetical Index