Preface

This document lists many examples of VDF-files and accompanying explanations which can 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.

- 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:

1. group name like `profile`
2. element within the group like `min_bg`
3. assigned new value like 95
4. optionally comment field like ### my text

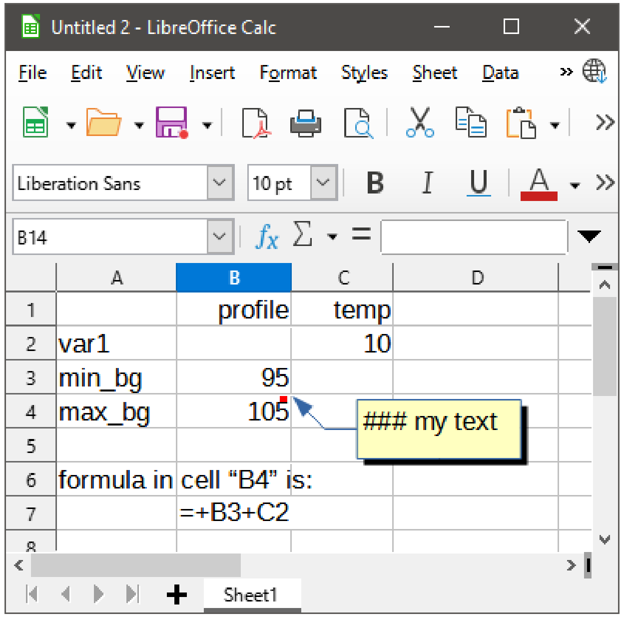
You can think of it as a spreadsheet:

a column heading

a row heading

a cell content like value or formula

a comment attached to the cell



The examples 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']`. 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 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 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. 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 results:

```
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 I first developed for autoISF 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 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.

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.

```
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   ###
```

The equivalent code word for the higher targets is STAIR_HTG.

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

STAIR

INTERPOL

POLYGON

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 algo and replace it by your own algo 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    delta_ISF    my_new_delta_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 algo it is helpful to understand the basic results contained in glucose_status:

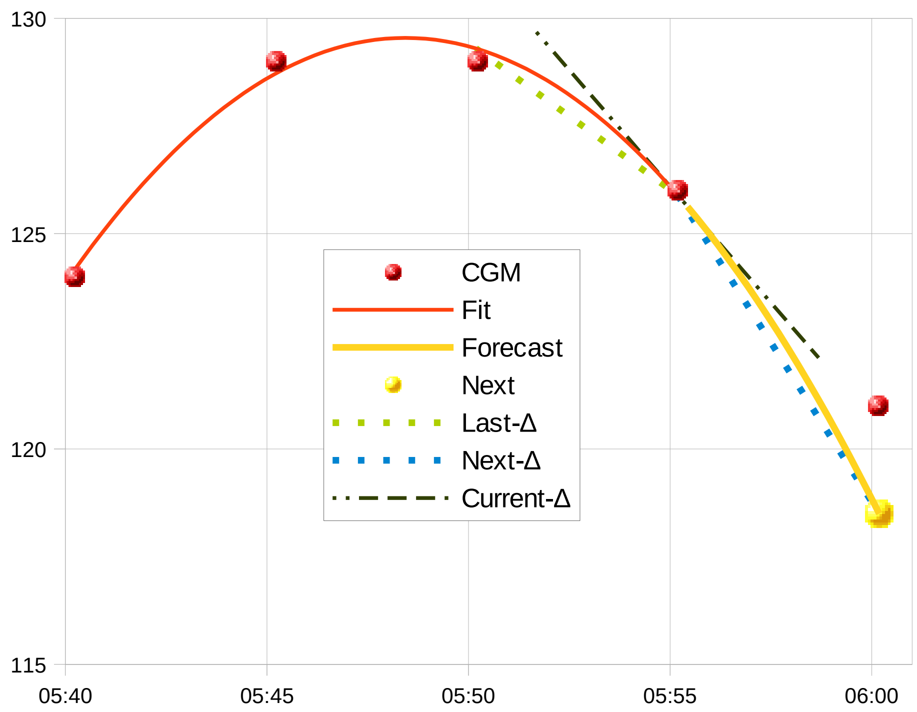
- glucose — standard APS glucose valid at the time the loop ran
- delta — standard APS delta
- short_avgdelta — standard APS short_avgdelta
- long_avgdelta — standard APS long_avgdelta
- date — time at which the above mentioned glucose was measured
- dura_ISF_minutes — duration of the glucose plateau
- dura_ISF_average — average level of glucose during that plateau window
- slope05 — average delta during the last 7.5 minutes calculated from best linear fit
- slope15 — average delta during the last 17.5[1] minutes calculated from best linear fit
- slope40 — average delta during the last 42.5[2] minutes calculated from best linear fit
- 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

1 Uses interval 0-17.5 as opposed to short_avgdelta which uses 7.5-17.5 minutes
2 Uses interval 0-42.5 as opposed to long_avgdelta which uses 17.5-42.5 minutes

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 we can use the first derivative:

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

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
- 5 minutes ahead means t_5=+1 and $delta(1) = a_1+2*a_2$, i.e. the next_delta

For the acceleration we can 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 in BG history and tries increasing the length of the fit window until this correlation coefficient deteriorates.

This gives you the background and tools to try your own extensions by coding it in VDF. Ideas could be combined effects of delta and acceleration or long lasting rises like seen in slope40. I am currently playing with long lasting accelerations, i.e. with parabola_fit_minutes >15.