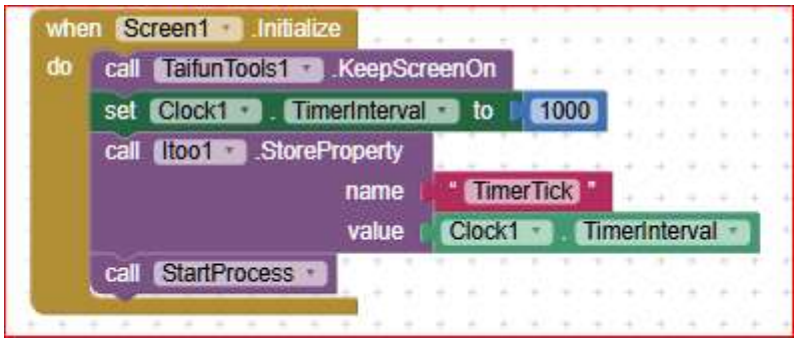
A “TRUE” DELAY FUNCTION FOR AI2 BY USING THE ITOO EXTENSION

The following notes are aimed at showing how to implement a non blocking delay function for AI2.



```
when Screen1 .Initialize
do  call TaifunTools1 .KeepScreenOn
    set Clock1 . TimerInterval to 1000
    call Itoo1 .StoreProperty
                     name  " TimerTick "
                     value  Clock1 . TimerInterval
    call StartProcess
```

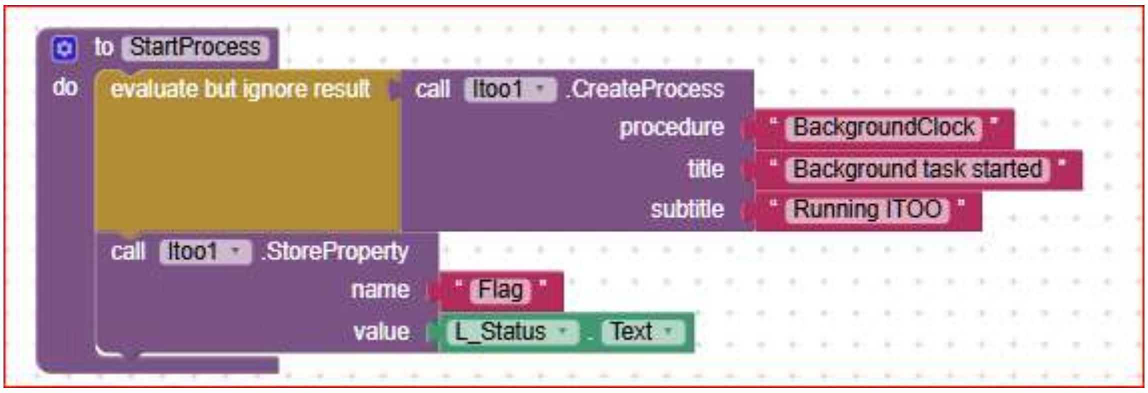
To keep the screen ON, the Taifun’s extension TaifunTools is used.

The timer tick is set to 1 (one) second in this example. Therefore the delay can be only multiple of 1 second.

The StoreProperty block allows to pass this data to the task that will run in background thanks to “itoo”.

The StartProcess procedure executes that steps necessary to start the hidden task running behind the scenes.

Here below its contents:



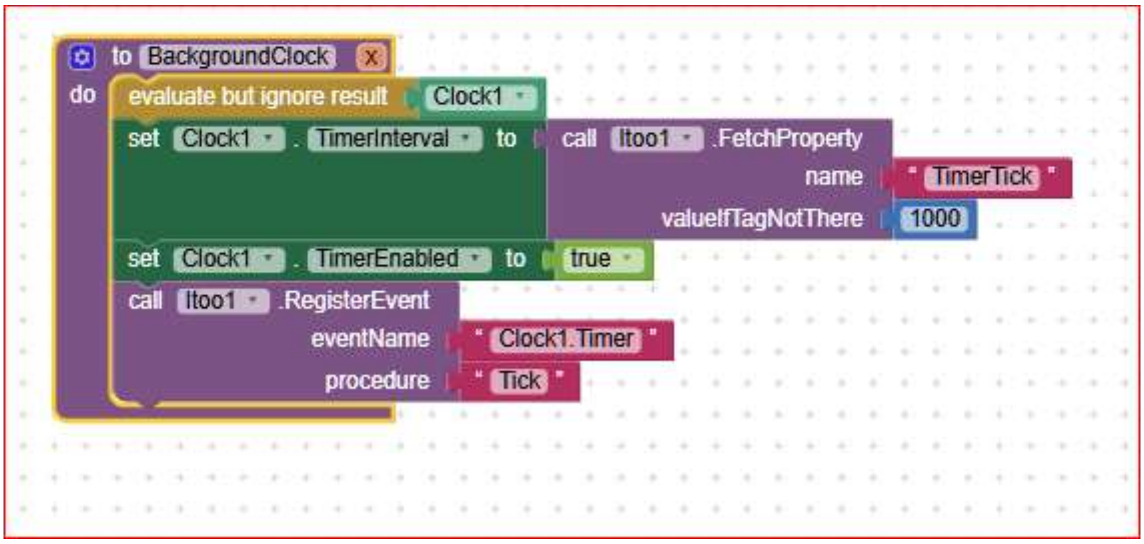
```
to StartProcess
do  evaluate but ignore result  call Itoo1 .CreateProcess
                                        procedure  " BackgroundClock "
                                        title  " Background task started "
                                        subtitle  " Running ITOO "
    call Itoo1 .StoreProperty
                     name  " Flag "
                     value  L_Status . Text
```

The Flag is the semaphore that is used to stop/release the app flow when the delay is running.

The hidden process is started once, and runs forever, till the app is closed, to avoid the popup title box to appear at any time it is invoked.

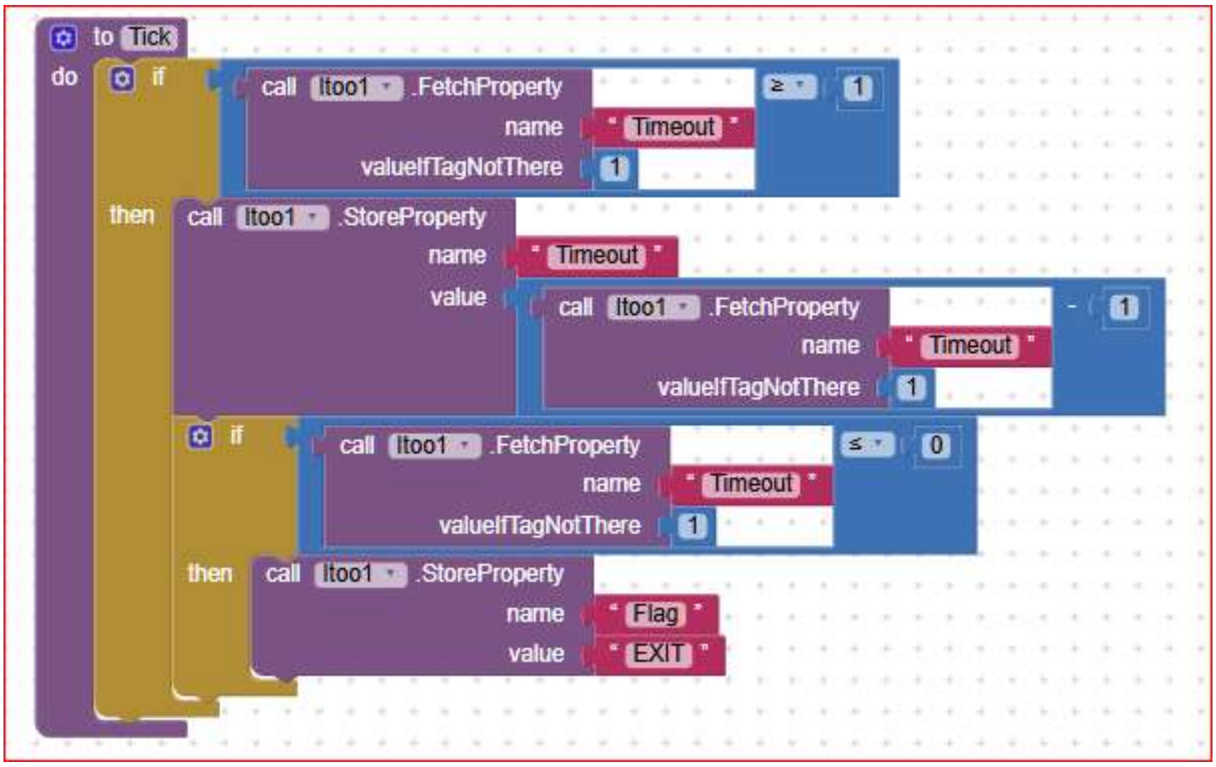
The BackgroundClock is the process that runs hidden. It receives the timer tick by using the FetchProperty block which is then set to the Clock1 TimerInterval, then the Clock1 is started.

Since in the background the events have no effect, the RegisterEvent block of itoo does the job on event's behalf. The procedure Tick will do the job of the Cock1.Timer standard event. Please note that the parameter "x", though not used, MUST be present for itoo to work propoerly.



```
to BackgroundClock x
do  evaluate but ignore result  Clock1
    set Clock1 . TimerInterval to  call Itoo1 .FetchProperty
                                        name  " TimerTick "
                                        valueIfTagNotThere  1000
    set Clock1 . TimerEnabled to  true
    call Itoo1 .RegisterEvent
         eventName  " Clock1.Timer "
         procedure  " Tick "
```

The following procedure works in place of the Clock1.Timer event:

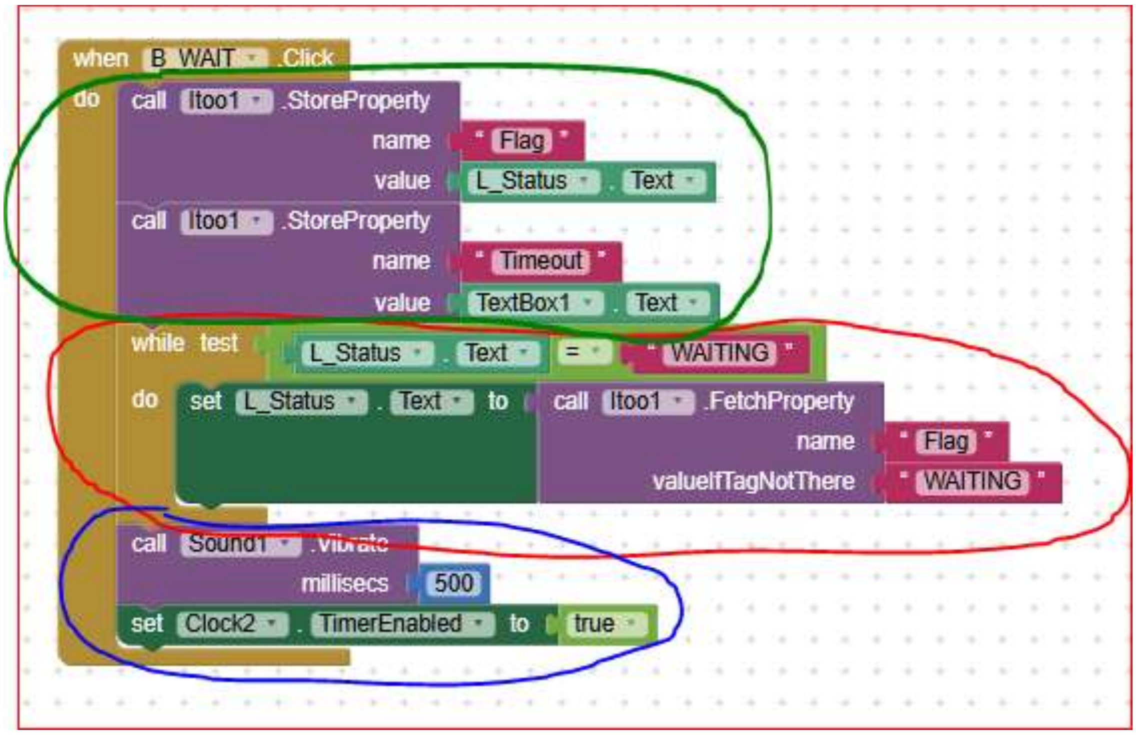


```
to Tick
do  if    call Itoo1 .FetchProperty              ≥  1
               name  " Timeout "
               valueIfTagNotThere  1
    then  call Itoo1 .StoreProperty
               name  " Timeout "
               value   call Itoo1 .FetchProperty          -  1
                            name  " Timeout "
                            valueIfTagNotThere  1
          if    call Itoo1 .FetchProperty        ≤  0
                     name  " Timeout "
                     valueIfTagNotThere  1
          then  call Itoo1 .StoreProperty
                     name  " Flag "
                     value  " EXIT "
```

The "event" above runs every second, , and checks if a timeout has been set and if it is still greater or equal than 0. If already 0, the timeout has elapsed and the procedure exits without nothing to do.

When the Timeout reaches 0 in its countdown, the flag "Flag" is set to EXIT and stored by using the block SToreProperty, therefore the app can be released to continue.

The following, is the “core” procedure.



```
when B_WAIT .Click
do  call Itoo1 .StoreProperty
                      name  "Flag"
                     value  L_Status . Text
    call Itoo1 .StoreProperty
                      name  "Timeout"
                     value  TextBox1 . Text
    while test  L_Status . Text  =  "WAITING"
       do  set L_Status . Text to  call Itoo1 .FetchProperty
                                                    name  "Flag"
                                      valueIfTagNotThere  "WAITING"
    call Sound1 .Vibrate
                 millisecs  500
    set Clock2 . TimerEnabled to  true
```

The green circled blocks make available to itoo the Flag data (the ptoperty Text of a label), that by default contains “WAITING” string, and the quantity of seconds to wait, extracted by the TextBox1, set by the user.

The red circled blocks are a while…do loop that waits for the change of the label L_Status.Text.

This while…do in the standard behaviour of AI2 will never exit, since there is no way to change the label text.

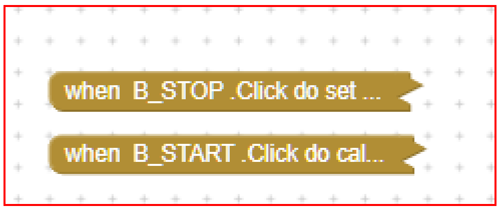
On the contrary, itoo is running on its own, and when the timeout has elapsed, it is capable to set the Flag value to “EXIT” (which is no longer = “WAITING”) and to make available this value to the app by means of the StoreProperty block.

In this way the while test will become “false” because the L_Status.Text has been changed by itoo, therefore it exits, leaving the app to continue from the instruction immediately after the while.

The blue circled blocks are just for aestethical purposes: when the timeout has elapsed a “buzz” is emitted, to wake up the user (😊) and the Clock2 is enabled so to allow the L_Status.Text to show in sequence “EXIT” and “WAITING” (again).

This is just an example, it is up to you to go further and to discover other powerful functionalities of AI2 empowered by itoo.

PS:



```
when B_STOP .Click do set ...
when B_START .Click do cal...
```

Not used in this version, provision for future expansions