DE-4000 SCRIPTING REFERENCE MANUAL

Form:DE-4000 SCRIPTING REFERENCE MANUAL:11-26

ALTRONIC



All rights reserved © ALTRONIC, LLC 2026

All rights reserved © ALTRONIC, LLC 2026

Table of Contents

DE-4000 SCRIPTING REFERENCE MANUAL 1
1. Begin on Dashboard on DE-4000 system environment 1
2. Choose “Global” from menu on left side of screen 2
3. In the Sub-Menu on the Left side select “Scripts” 2
4. Select one of the page icons under one of the 4 script options to open editor 3
5. Scripting can be entered into the editor 3
5.1 DE4000 Lua Script API 5
5.2 Master Control Script 10

All rights reserved © ALTRONIC, LLC 2026

All rights reserved © ALTRONIC, LLC 2026

DE-4000 SCRIPTING REFERENCE MANUAL

There is a delicate balance between providing a system that has capabilities that can be configured through a fixed set of options, and one that can be extended and expanded with custom programming. In designing the DE-4000 control system, the choice was made to provide a system where most applications can be met with simple configuration, but advanced functionality can be provided through custom programming using the “Lua” language.

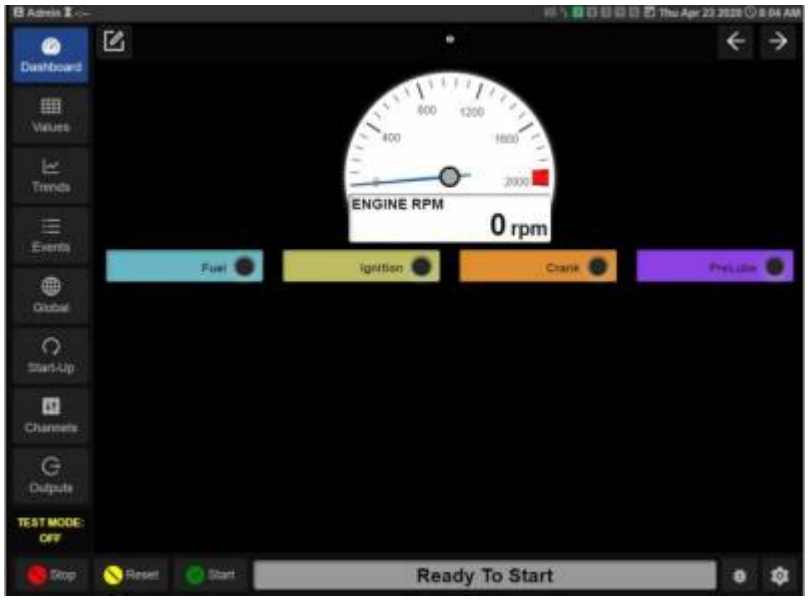
Lua is often referred to as a scripting language. Scripting languages differ from compiled languages as they eliminate extra step of compiling the written program into machine code.

Lua comes with a background of being robust, fast, and geared towards embedded applications, with a proven track record in the gaming industry. For the DE-4000 system it is small and fits in the memory we have available, holds a lot of power, and keeps it simple for writing in the language. All information regarding the Lua scripting language is located at https://Lua.org Using the Lua engine as an embedded tool allows for taking advantage of a full architecture and standard at your fingertips. Within the language there are all of the normal attributes to programming such as functions, variables, statements, expressions etc. All of this reference material can be found at https://lua.org/manual/5.3/ For getting started and using a guided reference, there are several editions of “Programming in Lua” available. Most recent editions are a paid for product that come in paper back or ebook form. While testing out Lua and becoming familiar, a free first edition is available and covers a lot of learning needs to get comfortable with the language. It can be located at https://www.lua.org/pil/contents.html. A major advantage to using Lua is its inherent ability to allow custom functions. While all normal functions and calls are published, there is the ability to add new functions in the DE-4000 firmware. Once new functions are defined and have calls to their internal properties, they then can be published for the user. This includes functions such as our flexible Modbus table and talking with various terminal boards linked in the system. Below is the start to the list of Altronic based functions. As functionality and features come to life through new ideas, this document will continually get updated with the latest scripts that we make available.

GETTING STARTED WITH DE-4000 SCRIPTS Basic Scripting on DE-4000

1. Begin on Dashboard on DE-4000 system environment

All rights reserved © ALTRONIC, LLC 2026

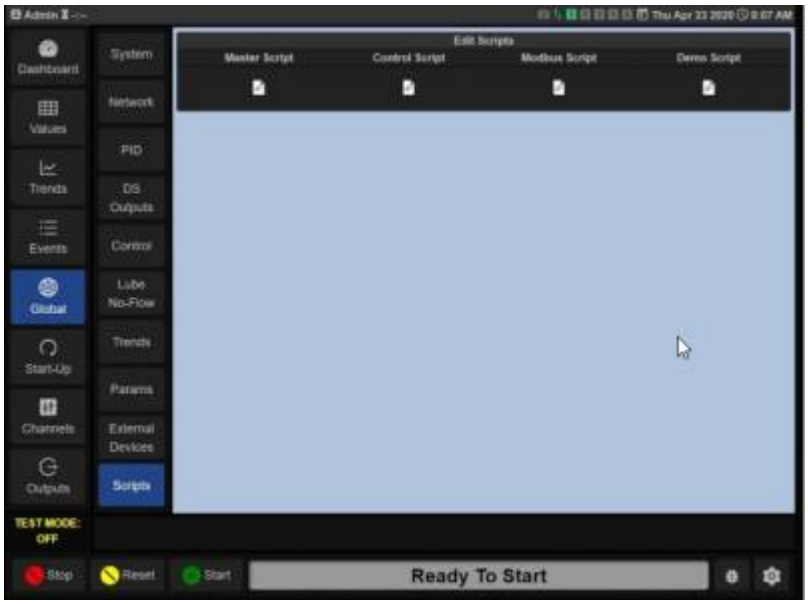


2. Choose “Global” from menu on left side of screen



3. In the Sub-Menu on the Left side select “Scripts”

All rights reserved © ALTRONIC, LLC 2026



4. Select one of the page icons under one of the 4 script options to open editor

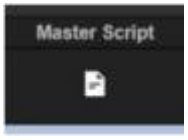


5. Scripting can be entered into the editor

All rights reserved © ALTRONIC, LLC 2026



Scripting Windows and examples

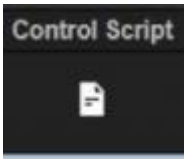
Master Script 

The Master Script section is the Primary scripting environment. Primary scripting functions can be written in this section.

Example:

```
local suction = get_channel_val(1,1)
local discharge1 = get_channel_val(1,3)
diff = discharge1 - suction
set_sVirt(“Difference”, diff)
```

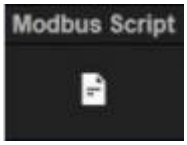
The first line gets the channel value from Terminal board 1 Input 1 and stores it in local variable named suction. The second line gets the channel value from Terminal board 1 Input 3 and stores it in local variable named discharge1. The third line takes the discharge1 pressure and subtracts the suction pressure and stores it in the global variable named diff (NOTE: Any value that you want to access from another scripting section must be stored in a global variable. This is used most in calling values into Modbus registers as explained below). The fourth line copies the value from diff and stores it into the Virtual status channel named “Difference” This channel can be displayed on the Dashboard.

Control Script 

The Control Script section is used to override the default control strategy found on the Global/Control page. A copy of the default control script (found in attached appendix) can be copied into this section and then modified to change the control functionality as well as add additional control loops beyond the default 2.

All rights reserved © ALTRONIC, LLC 2026



Modbus Script 

The Modbus Script section is used to move data into and out of Modbus registers




```
defaultModbus()
set_modbus(300,diff)
```

The first line pulls in the factory set Modbus mapping The second line moves the value from the global variable named diff into the 40300 Modbus Register

5.1 DE4000 Lua Script API

CUSTOM FUNCTIONS FOR SCRIPTING

All rights reserved © ALTRONIC, LLC 2026

create_param("index",default,"catergory","description")

- creates a user configurable parameter
- parameter is stored as `index`,
- default value(If not changed by user) is `default`
- parameters will be grouped on the Global/Params page by `category`
- `description` is text to describe the parameter to the user

Example:

```
create_param("NumEngCyl",8,"Engine Params","Num. of Engine Cylinders")
```

---

get_channel_val(terminal,channel)

- returns current value of analog input `channel` on terminal module `terminal`
- return value type is numeric

Example:

```
local sp = get_channel_val(1,5)
```

> reads value of Suction Pressure from Terminal Module #1 , Input #5

---

get_gbl("index",default)

- returns global config setting stored under `index` or returns `default` if not defined

> **note:** get_gbl is used to retrieve global CONFIGURATION settings that are typically set when the system is configured and do not change as the system is running. If you want to set and retrieve global STATUS variables use the get_sGbl() and set_sGbl() functions >If you want to create and read virtual channels use the set_sVirt() and get_sVirt() functions.

Example:

```
local nt = get_gbl("NumTerm",1)
```

> gets the number of terminal boards installed in the system

All rights reserved © ALTRONIC, LLC 2026

get_param("index")

- return either the default value or the user configured value of the parameter `index`

Example:

```
get_param("NumEngCyl")
```

>gets the configured parameter for number of engine cylinders

---

get_rpm(channel)

- reads the RPM input `channel` in units of revolutions per minute

> **note:** valid channel numbers are 1 - 10(2 channels per board, up to 5 terminal boards)

Each Terminal Module has 2 RPM inputs (RPM1 and RPM2)

- Terminal Module **#1** RPM channels are **1,2**
- Terminal Module **#2** RPM channels are **3,4**
- Terminal Module **#3** RPM channels are **5,6**
- Terminal Module **#4** RPM channels are **7,8**
- Terminal Module **#5** RPM channels are **9,10**

Example:

```
local engineRPM = get_rpm(1)
local turboRPM = get_rpm(6)
```

> Read RPM1 channel from terminal module #1 and read RPM2 channel from Terminal module #3

---

get_sGbl("index", default)

- If `index` is defined in the global status table then it returns the value associated with `index`
- If `index` is not defined and optional `default` is provided then returns `default`

>**note:** It is recommended to always provide a default value when using this function

Example:

```
local cp = get_sGbl("calculatedPressure",0)
```

> get the previously stored value "calculatedPressure", Returns `0` if not found.

All rights reserved © ALTRONIC, LLC 2026

get_state()

- returns the current engine state(possible values currently 0 - 10)

Example:

```
local engineState = get_state()
if engineState > 7 then
    set_timer("WarmupTimer",1000)
end
```

---

get_sVirt("index")

- returns the value of virtual channel `index` or returns `default` if the virtual channel does not exist.

Example:

```
local tl = get_sGbl("timeLimit")
local et = get_sVirt("ElapsedTime",0)
if et > tl then
    set_sGbl("timeExceeded",true)
else
    set_sGbl("timeExceeded",false)
end
```

>Gets the value of virtual channel ElapsedTime and set value of status global "timeExceeded" if ElapsedTime is greater than status global "timeLimit"

---

get_time()

- returns the UNIX "epoch" time (Defined as the number of seconds elapsed since Jan 1, 1970)

Example:

```
local startTime = get_sGbl("startTime",0)
if startTime == 0 then
    local currentTime = get_time()
    startTime = currentTime
    set_sGbl("startTime",currentTime)
end
local et = get_time() - startTime
set_sVirt("ElapsedTime",et)
```

>Stores current time if first time through, otherwise calculate elapsed time

All rights reserved © ALTRONIC, LLC 2026

get_timer("index")

- returns 1 or 2 values
- First return value(Boolean) is **true** if timer is active(counting down) or **false** if timer is expired or has not been set yet
- Second return value is the number of seconds remaining or **-1** if timer is not active or has not been set yet

Example:

```
if not get_timer("myTimer") then
    set_sGbl("timedOut",true)
else
    set_sGbl("timedOut",false)
end
```

> if timer is expired, then set global status "timedOut" to **true**

```
local active,remaining = get_timer("myTimer")
if not active then
    set_sVirt("timeRemaining","Expired")
else
    set_sVirt("timeRemaining",remaining)
end
```

---

getStateLabel(state)

- return the label for the engine state corresponding to the parameter `state`

Example:

```
local stateLabel = getStateLabel(get_state())
local active, remaining = get_timer("myTimer")
if remaining > 0 then
    stateLabel == StateLabel.." "..remaining
end
set_sVirt("Countdown",stateLabel)
```

---

set_sGbl("index",value)

- store `value` in the global status table under `index`
- value can be a number or string but if storing a boolean use the **tostring()** function

Example:

All rights reserved © ALTRONIC, LLC 2026

```
local mpe = false
local sp = get_channel_val(1,5)
if sp > 15 then
    mpe = true
end
set_sGbl("minPressureExceeded",tostring(mpe))
```

> store boolean value **minPressureExceeded**

---

set_sVirt("index",value)

- sets a virtual status channel with channel name `index`

> **Note:** Once you create a virtual channel, you can add that channel to the dashboard using the channel name `index`

Example:

```
local sp = get_channel_val(1,5) --suction pressure
local dp = get_channel_val(1,6) --discharge pressure
local diffPress = dp - sp
set_sVirt("SuctDischDiff",diffPress)
```

> calculate the differential between suction and discharge pressure and assign to virtual channel

---

set_timer("index",secs)

- activate timer `index` and set countdown time to `secs`

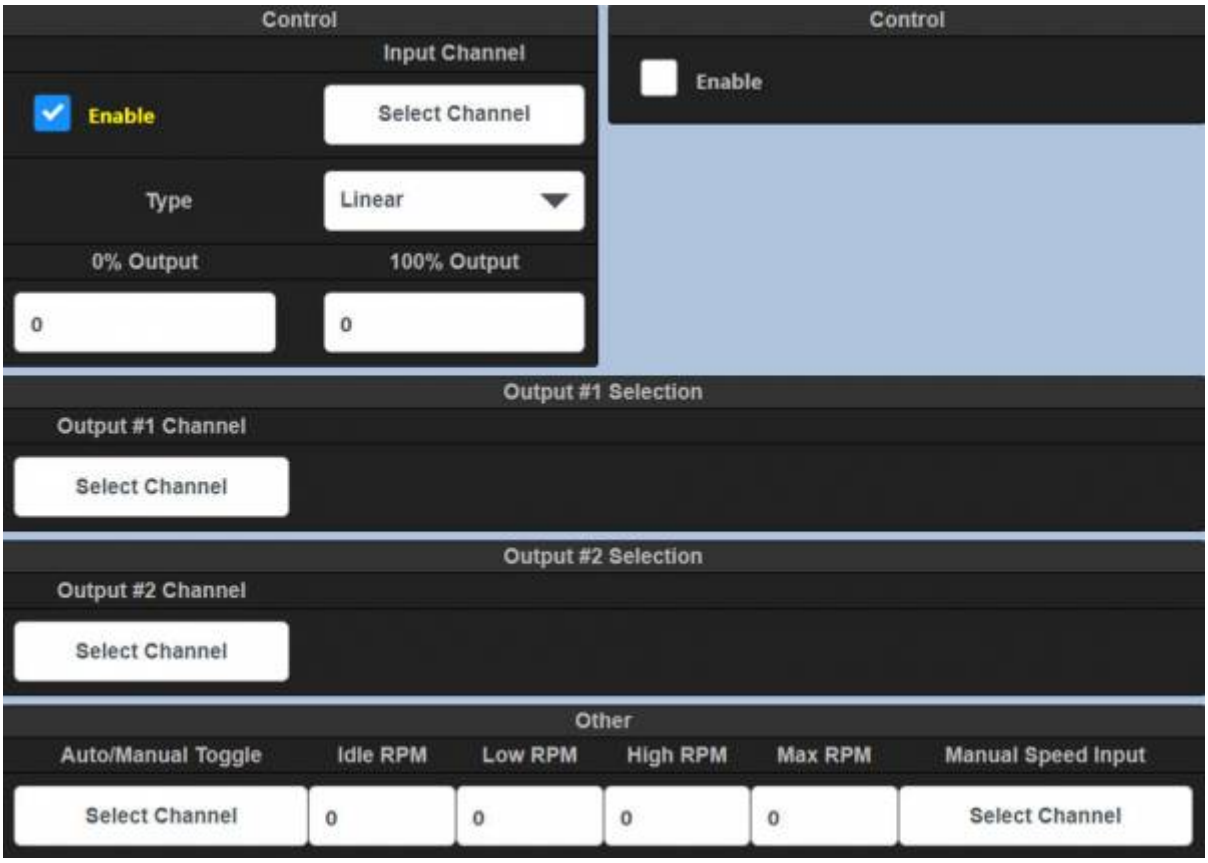
Example:

```
set_timer("myTimer",300)
```

> create timer `myTimer` and start countdown time to 300 seconds

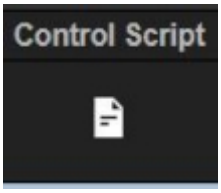
5.2 Master Control Script

All rights reserved © ALTRONIC, LLC 2026



When you enter a control setup under the Global Control page the code that runs is called MasterControl.

If you wish to modify this functionality you can copy this code into the Control Script editor and make your changes to the standard configuration.



mastercontrol.lua

```
local rampRate1 = get_gbl("rampRate1",0.8)
local rampRate2 = get_gbl("rampRate2",0.8)
local dischTerm = tonumber_def(get_gbl("spDischTerm",0),0)
local dischChan = tonumber_def(get_gbl("spDischChan",0),0)
local suctTerm = tonumber_def(get_gbl("spSuctTerm",0),0)
local suctChan = tonumber_def(get_gbl("spSuctChan",0),0)
local suctMin = tonumber_def(get_gbl("suctMin",0),0)
local recycleMin = tonumber_def(get_gbl("recycleMin",0),0)
local recycleMax = tonumber_def(get_gbl("recycleMax",0),0)
local suctSp = tonumber_def(get_gbl("suctSp",0),0)
local dischMax = tonumber_def(get_gbl("dischMax",0),0)
local dischSp = tonumber_def(get_gbl("dischSp",0),0)
local outputTerm = tonumber_def(get_gbl("outputTerm",0),0)
local outputChan = tonumber_def(get_gbl("outputChan",0),0)
local recycleTerm = tonumber_def(get_gbl("outputTerm2",0),0)
local recycleChan = tonumber_def(get_gbl("outputChan2",0),0)
```


All rights reserved © ALTRONIC, LLC 2026

```
17.   local speedRevAct = tonumber_def(get_gbl("speedRevAct",0),0)
18.   local recycleRevAct = tonumber_def(get_gbl("recycleRevAct",0),0)
19.   local outputLow = tonumber_def(get_gbl("outputLow",0),0)
20.   local outputLow2 = tonumber_def(get_gbl("outputLow2",0),0)
21.   local outputHigh = tonumber_def(get_gbl("outputHigh",0),0)
22.   local outputHigh2 = tonumber_def(get_gbl("outputHigh2",0),0)
23.   local spSuctType = get_gbl("spSuctType","linear")
24.   local spDischType = get_gbl("spDischType","linear")
25.   local suctPIDPFactor =
    tonumber_def(get_gbl("suctPIDPFactor",0),0)
26.   local suctPIDIFactor =
    tonumber_def(get_gbl("suctPIDIFactor",0),0)
27.   local suctPIDDFactor =
    tonumber_def(get_gbl("suctPIDDFactor",0),0)
28.   local dischPIDPFactor =
    tonumber_def(get_gbl("dischPIDPFactor",0),0)
29.   local dischPIDIFactor =
    tonumber_def(get_gbl("dischPIDIFactor",0),0)
30.   local dischPIDDFactor =
    tonumber_def(get_gbl("dischPIDDFactor",0),0)
31.   local recycleCtrl = false
32.   local recycleSuctionRev = false
33.   local recycleDischargeRev = false
34.   if recycleChan > 0 and recycleTerm > 0 then
35.     recycleCtrl = true
36.   end
37.   --if recycleCtrl and spSuctType == "linear" and outputLow2 >
    suctSp then
38.   --  recycleSuctionRev = true
39.   --end
40.   --if recycleCtrl and spDischType == "linear" and  recycleMax <
    dischSp then
41.   --  recycleDischargeRev = true
42.   --end
43.   --print("disch: "..tostring(disch).." suct:"..tostring(suct))
44.   --local suct = 500
45.   local dischPct = 100
46.   local suctPct = 100
47.
48.
49.   local dischOutput = 0
50.   local suctOutput = 0
51.   local rSuctOutput = 0
52.   local rDischOutput = 0
53.   local minLoad = 0
54.   local maxLoad = 100
55.   local minRecycle = 0
56.   local maxRecycle = 100
57.   local speedTarget = get_sGbl("speedTarget",0)
58.   local recycleTarget = get_sGbl("recycleTarget",0)
59.
```


All rights reserved © ALTRONIC, LLC 2026

```
60.   function map_range(rangeLow,rangeHigh,input)
61.     if input <= rangeLow and input <= rangeHigh then
62.       return 0
63.     end
64.     if input >= rangeLow and input >= rangeHigh then
65.       return 100
66.     end
67.     local rangeDiff = math.abs(rangeLow - rangeHigh)
68.     local min = math.min(rangeLow,rangeHigh)
69.     local retval = math.abs(input - min) / rangeDiff * 100
70.     if retval > 100 then retval = 100 end
71.     if retval < 0 then retval = 0 end
72.     return retval
73.   end
74.
75.   local suct = false
76.   local suctVal = 0
77.   if tonumber_def(get_gbl("spSuctEn",0),0) == 1 then
78.     if suctTerm > 0 and suctChan > 0 then
79.       suctVal = get_channel_val(suctTerm,suctChan)
80.       suct = true
81.     end
82.   end
83.
84.
85.   if suct then
86.     if spSuctType == "linear" then
87.       local suctDiff = suctSp - suctMin
88.       if suctDiff == 0 then suctDiff = 1 end
89.       if suctVal < suctSp then
90.         local suctErr = suctSp - suctVal
91.         suctPct = suctErr / suctDiff
92.         if suctPct > 1 then suctPct = 1 end
93.         if suctPct < 0 then suctPct = 0 end
94.         suctOutput = (1 - suctPct) * 100
95.       else
96.         suctOutput = 100
97.       end
98.     else
99.       set_gbl("PIDsuctEnable",1)
100.      set_gbl("PIDsuctPFactor",suctPIDPFactor)
101.      set_gbl("PIDsuctIFactor",suctPIDIFactor)
102.      set_gbl("PIDsuctDFactor",suctPIDDFactor)
103.      set_gbl("PIDsuctSp",suctSp)
104.      set_gbl("PIDsuctDeadband",0.2)
105.      local suctPidOutput = doPid("suct",suctVal)
106.      suctOutput = suctPidOutput
107.    end
108.  else
109.    suctOutput = 100
110.  end
```


All rights reserved © ALTRONIC, LLC 2026

```
111.
112.
113.   local disch = false
114.   local dischVal = 0
115.   if tonumber_def(get_gbl("spDischEn",0),0) == 1 then
116.     if dischTerm > 0 and dischChan > 0 then
117.         dischVal = get_channel_val(dischTerm,dischChan)
118.         disch = true
119.     end
120.   end
121.   if disch then
122.     if spDischType == "linear" then
123.       local dischDiff = dischMax - dischSp
124.       if dischDiff == 0 then dischDiff = 1 end
125.       if dischVal > dischSp then
126.         local dischErr = dischVal - dischSp
127.         dischPct = dischErr / dischDiff
128.         if dischPct > 1 then dischPct = 1 end
129.         if dischPct < 0 then dischPct = 0 end
130.         dischOutput = (1 - dischPct) * 100
131.       else
132.         dischOutput = 100
133.       end
134.     else
135.       set_gbl("PIDdischEnable",1)
136.       set_gbl("PIDdischPFactor",dischPIDPFactor)
137.       set_gbl("PIDdischIFactor",dischPIDIFactor)
138.       set_gbl("PIDdischDFactor",dischPIDDFactor)
139.       set_gbl("PIDdischSp",dischSp)
140.       set_gbl("PIDdischRevAct",1)
141.       set_gbl("PIDdischDeadband",0.2)
142.       local dischPidOutput = doPid("disch",dischVal)
143.       dischOutput = dischPidOutput
144.     end
145.   else
146.     dischOutput = 100
147.   end
148.
149.   --print("suctOutput dischOutput: "..math.floor(suctOutput).."
    "..math.floor(dischOutput))
150.
151.
152.
153.
154.   local minOutput = 100
155.   local winning = 0
156.   if suctOutput < minOutput then
157.     minOutput = suctOutput
158.     winning = 1
159.   end
160.   if dischOutput < minOutput then
```


All rights reserved © ALTRONIC, LLC 2026

```
161.     minOutput = dischOutput
162.     winning = 2
163.   end
164.
165.   if suctOutput == dischOutput then
166.     winning = 0
167.   end
168.
169.   if winning == 0 then
170.     set_gbl("PIDsuctMax",100)
171.     set_gbl("PIDdischMax",100)
172.   end
173.
174.   if winning == 1 then
175.     set_gbl("PIDdischMax",math.min(suctOutput + 2,100))
176.     set_gbl("integraldisch",0)
177.     set_gbl("lastErrdisch",0)
178.     set_gbl("outputSumdisch",0)
179.     set_gbl("PIDsuctMax",100)
180.   end
181.   if winning == 2 then
182.     set_gbl("PIDsuctMax",math.min(dischOutput + 2,100))
183.     set_gbl("integralsuct",0)
184.     set_gbl("lastErrsuct",0)
185.     set_gbl("outputSumsuct",0)
186.     set_gbl("PIDdischMax",100)
187.   end
188.
189.   local recycleMinOutput = minOutput
190.
191.   local manOutput = 0
192.   --
     ********************************************************************
     **
193.   local manMode = 0
194.   local manTerm = tonumber_def(get_gbl("manTerm",0),0)
195.   local manChan = tonumber_def(get_gbl("manChan",0),0)
196.   if manTerm > 0 and manChan > 0 then
197.     local manInput = get_channel_val(manTerm,manChan)
198.     if manInput > 0.5 then
199.       manMode = 0
200.       set_sVirt("SpeedControl","Auto")
201.     else
202.       manMode = 1
203.       set_sVirt("SpeedControl","Manual")
204.     end
205.   else
206.     if get_sVirt("SpeedControl","Auto") == "Auto" then
207.       manMode = 0
208.     else
209.       manMode = 1
```


All rights reserved © ALTRONIC, LLC 2026

```
    end
  end
--[[
      local idleSpeed = get_gbl("idleSpeed",0)
      local lowSpeed = get_gbl("lowSpeed",0)
      local highSpeed = get_gbl("highSpeed",0)
      local speedPct = 0

      if st > highSpeed then st = highSpeed end
      if st < lowSpeed then st = lowSpeed end
      if get_state() ~= 8 then
        --st = idleSpeed
      end
      set_sVirt("Speed Target",st)
      speedPct = (st - lowSpeed) /(highSpeed - lowSpeed) * 100
      if speedPct < 0 then speedPct = 0 end
      if speedPct > 100 then speedPct = 100 end
      st = speedPct

      if idleSpeed < lowSpeed then
        local speedRpm = speedPct / 100  * (highSpeed - lowSpeed)
    + lowSpeed
        st = (speedRpm - idleSpeed) / (highSpeed - idleSpeed) *
    100
        --st = (st - idleSpeed) / (highSpeed - idleSpeed) * 100
      end
]]--


  --if manMode == 1 and get_state() == 8 then
  local manSpeed = get_sVirt("ManualSpeed",0)
  local idleSpeed = get_gbl("idleSpeed",0)
  local lowSpeed = get_gbl("lowSpeed",0)
  local highSpeed = get_gbl("highSpeed",0)
  local maxSpeed = get_gbl("maxSpeed",0)
  local diff = highSpeed - lowSpeed
  if diff < 0 then diff = 0 end
  local maxDiff = maxSpeed - idleSpeed
  if maxDiff < 0 then maxDiff = 0 end

  if get_sVirt("speedBump",0) ~= 0 then
    local si = get_gbl("SpeedIncrement",0)
    local sip = get_param("SpeedIncrement",0)
    if sip ~= 0 then si = sip end
```


All rights reserved © ALTRONIC, LLC 2026

```
259.     manSpeed = manSpeed + (si * get_sVirt("speedBump",0))
260.     set_sVirt("speedBump",0)
261.   end
262.
263.   if get_sVirt("AutoManBump",0) > 0 then
264.     set_sVirt("SpeedControl","Auto")
265.     set_sVirt("AutoManBump",0)
266.   end
267.
268.   if get_sVirt("AutoManBump",0) < 0 then
269.     set_sVirt("SpeedControl","Manual")
270.     set_sVirt("AutoManBump",0)
271.   end
272.
273.   if manMode == 1 then
274.     local manSpeedTerm = tonumber_def(get_gbl("manSpeedTerm",0),0)
275.     local manSpeedChan = tonumber_def(get_gbl("manSpeedChan",0),0)
276.     if manSpeedTerm > 0 and manSpeedChan > 0 then --*** USE SPEED
    POT TO SET SPEED
277.       local speedInput =
    tonumber(get_channel_val(manSpeedTerm,manSpeedChan))
278.       local speedPct = (speedInput / 5) * 100
279.       if speedPct > 100 then speedPct = 100 end
280.       if speedPct < 0 then speedPct = 0 end
281.       manOutput = speedPct
282.       manSpeed = math.floor((speedPct / 100) * diff + lowSpeed +
    0.5)
283.     else -- Use ManualSpeed to set speed
284.       manOutput = ((manSpeed - lowSpeed) / diff) * 100.0
285.       if manOutput < 0 then manOutput = 0 end
286.       if manOutput > 100 then manOutput = 100 end
287.     end
288.     minOutput = manOutput
289.   else
290.     --speedTarget =
291.     local stRpm = (speedTarget/100) * maxDiff + idleSpeed
292.     if stRpm < lowSpeed then stRpm = lowSpeed end
293.     if stRpm > highSpeed then stRpm = highSpeed end
294.     manSpeed = math.floor(stRpm)
295.   end
296.
297.
298.
299.
300.
301.
302.   --speedTarget = get_sGbl("speedTarget",0)
303.   if manSpeed < lowSpeed then
304.     manSpeed = lowSpeed
305.   end
306.   if manSpeed > highSpeed then
```


All rights reserved © ALTRONIC, LLC 2026

```
307.     manSpeed = highSpeed
308.   end
309.
310.   set_sVirt("ManualSpeed",manSpeed)
311.
312.
313.
314.   --
     **********************************************************************
     **
315.
316.
317.   local output1 = 0
318.   local output2 = 0
319.   if spSuctType == "pid" or spDischType == "pid" then
320.     --Map minOutput to output1
321.     output1 = map_range(outputLow,outputHigh,minOutput)
322.     set_sVirt("out1",output1)
323.     --Map minOutput to ourput2
324.     output2 = map_range(outputLow2,outputHigh2,recycleMinOutput)
325.     set_sVirt("out2",output2)
326.     local hasRPM = idleSpeed > 0 and lowSpeed > 0 and highSpeed >
     0 and maxSpeed > 0
327.     if outputTerm and outputChan then
328.       if hasRPM then
329.         local speedRpm = output1 / 100  * (highSpeed - lowSpeed) +
     lowSpeed
330.         --set_ao_val(outputTerm,outputChan,(speedRpm - idleSpeed)
     / (maxSpeed - idleSpeed) * 100)
331.         speedTarget = (speedRpm - idleSpeed) / (maxSpeed -
     idleSpeed) * 100
332.       else
333.         --set_ao_val(outputTerm,outputChan,output1)
334.         speedTarget = output1
335.       end
336.     end
337.     if recycleTerm and recycleChan then
338.       set_ao_val(recycleTerm,recycleChan,output2)
339.     end
340.
341.     if get_state() == 9 then
342.       speedTarget = get_sGbl("speedTarget",0)
343.       if speedTarget > 0 then speedTarget = speedTarget -
     rampRate1 end
344.       if speedTarget < 0 then speedTarget = 0 end
345.     end
346.     if get_state() < 8 then speedTarget = 0 end
347.     set_sGbl("speedTarget",speedTarget)
348.     --set_sGbl("a"..outputChan,speedTarget)
349.     set_ao_val(outputTerm,outputChan,speedTarget)
350.     --set_ao_val(outputChan,speedTarget)
```


All rights reserved © ALTRONIC, LLC 2026

```
351.     --print(suctOutput.." "..dischOutput.." "..speedTarget)
352.     set_sVirt("spTarget",speedTarget)
353.     --set_speed_val(1,speedTarget)
354.
355.     if hasRPM then
356.       local sRpm = (speedTarget/100) * maxDiff + idleSpeed
357.       set_sVirt("Speed Target",math.floor(sRpm + 0.5))
358.     end
359.
360.
361.
362.   else
363.
364.     -- Remember that minOutput is 0 - 100 pct of lowSpeed <->
     highSpeed
365.     -- We need to convert this to 0 - 100 pct of idleSpeed <->
     maxSpeed
366.     local suctPct = map_range(outputLow,outputHigh,minOutput)
367.     local speedRpm = suctPct / 100  * (highSpeed - lowSpeed) +
     lowSpeed
368.     minOutput = (speedRpm - idleSpeed) / (maxSpeed - idleSpeed) *
     100
369.
370.
371.
372.     if minOutput <= speedTarget then
373.       speedTarget = speedTarget - rampRate1
374.       if speedTarget < minOutput then speedTarget = minOutput end
375.     else
376.         speedTarget = speedTarget + rampRate1
377.         if speedTarget > minOutput then speedTarget = minOutput
     end
378.         if speedTarget > maxLoad then speedTarget = maxLoad end
379.     end
380.     if speedTarget > maxLoad then speedTarget = maxLoad end
381.     if speedTarget < minLoad then speedTarget = minLoad end
382.
383.     if recycleCtrl then
384.       local recyclePct =
     map_range(outputLow2,outputHigh2,recycleMinOutput)
385.       --if recycleRevAct == 1 then recyclePct = 100 - recyclePct
     end
386.       if recyclePct <= recycleTarget then
387.         recycleTarget = recycleTarget - rampRate2
388.         if recycleTarget < recyclePct then recycleTarget =
     recyclePct end
389.       else
390.         recycleTarget = recycleTarget + rampRate2
391.         if recycleTarget > recyclePct then recycleTarget =
     recyclePct end
392.       end
```


All rights reserved © ALTRONIC, LLC 2026

```
393.       if recycleTarget > maxRecycle then recycleTarget =
     maxRecycle end
394.       if recycleTarget < minRecycle then recycleTarget =
     minRecycle end
395.       local recycleOutput = recycleTarget
396.       if get_state() < 8 then
397.         recycleTarget = 0
398.       end
399.       if recycleRevAct == 1 then
400.         recycleOutput = 100 - recycleOutput
401.       end
402.       --set_sGbl("a"..recycleChan,recycleOutput)
403.       set_ao_val(recycleTerm,recycleChan,recycleOutput)
404.       set_sGbl("recycleTarget",recycleTarget)
405.       set_sVirt("recycleTarget",recycleTarget)
406.     end
407.
408.     if get_state() == 9 then
409.       speedTarget = get_sGbl("speedTarget",0)
410.       if speedTarget > 0 then speedTarget = speedTarget -
     rampRate1 end
411.       if speedTarget < 0 then speedTarget = 0 end
412.     end
413.     if get_state() < 8 then speedTarget = 0 end
414.     set_sGbl("speedTarget",speedTarget)
415.     --set_sGbl("a"..outputChan,speedTarget)
416.     set_ao_val(outputTerm,outputChan,speedTarget)
417.     --set_ao_val(outputChan,speedTarget)
418.     --print(suctOutput.." "..dischOutput.." "..speedTarget)
419.     set_sVirt("spTarget",speedTarget)
420.     --set_speed_val(1,speedTarget)
421.     local sRpm = (speedTarget/100) * maxDiff + idleSpeed
422.     set_sVirt("Speed Target",math.floor(sRpm + 0.5))
423.
424.
425.   end
```

From:
https://staging.altronic.a2hosted.com/ - **wiki STAGING !!!!!**

Permanent link:
https://staging.altronic.a2hosted.com/doku.php?id=documents:de4000:de4000script&rev=1643144021

Last update: **2022/01/28 21:17**

All rights reserved © ALTRONIC, LLC 2026