Scripting Reference Manual

DE-4000 Series Configurable Safety Shutdown and Control System

Form DE-4000 SRM 06-20

altronic

DE-4000 SCRIPTING REFERENCE MANUAL

One overarching capability that allows a bridge to gap the standard needs of everyday systems and the customer needs of innovation is scripting. A scripting language, cleverly named Lua, is embedded into the DE-4000 system. It operates as a script mainly meaning that it does not need additional tools to convert the "code" into machine language. It also is looked at and corrected for errors every time the script runs. Therefore it is an "interpreted" language and runs all of the time when you ask it.

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/

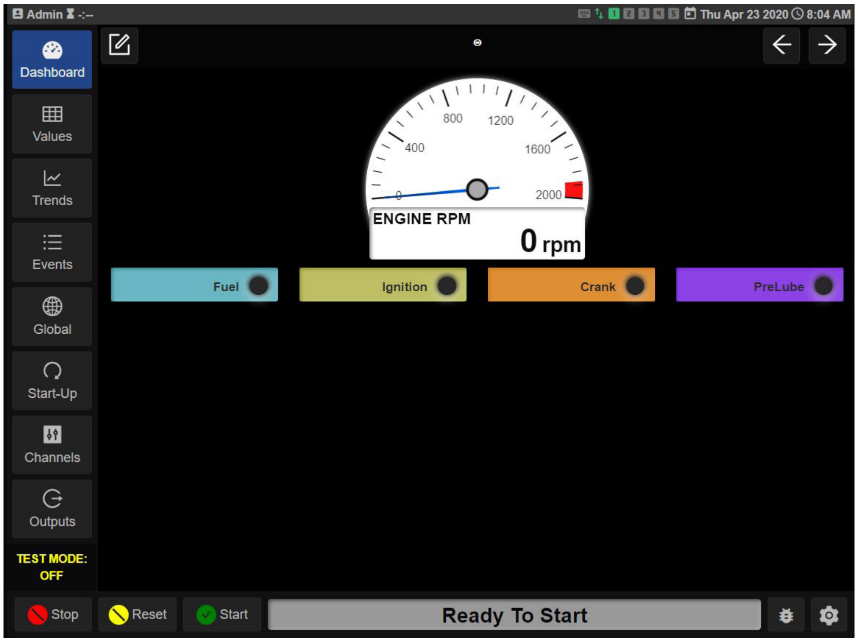
For getting started and using a guided reference, there are several editions of "Programming in Lua" available. Most recent editions are a paid for product that come in paper back or ebook form. While testing out Lua and becoming familiar, a free first edition is available and covers a lot of learning 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.

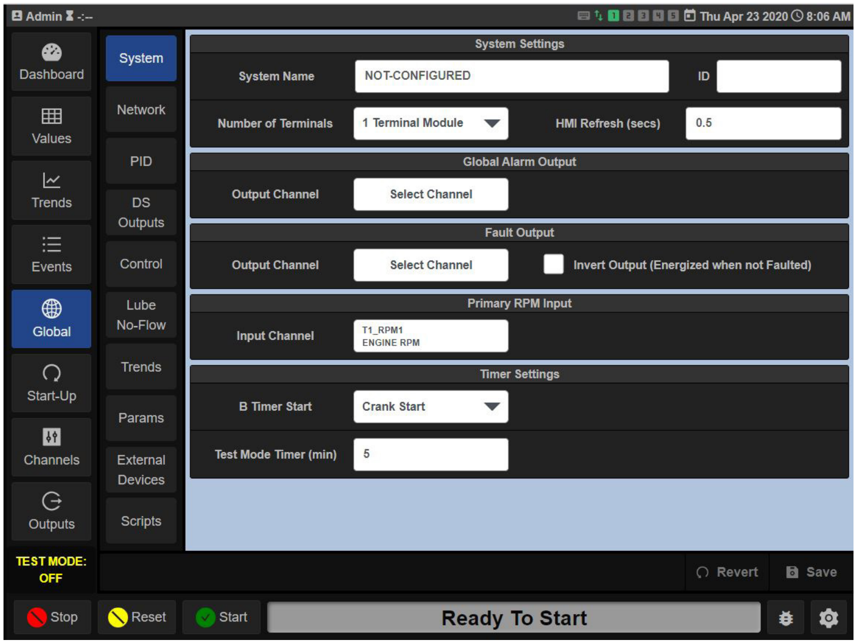
GETTING STARTED WITH DE-4000 SCRIPTS

Basic Scripting on DE-4000

1. Begin on Dashboard on DE-4000 system environment



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



System Settings

System Name: NOT-CONFIGURED ID:

Number of Terminals: 1 Terminal Module HMI Refresh (secs): 0.5

Global Alarm Output

Output Channel: Select Channel

Fault Output

Output Channel: Select Channel □ Invert Output (Energized when not Faulted)

Primary RPM Input

Input Channel: T1_RPM1 ENGINE RPM

Timer Settings

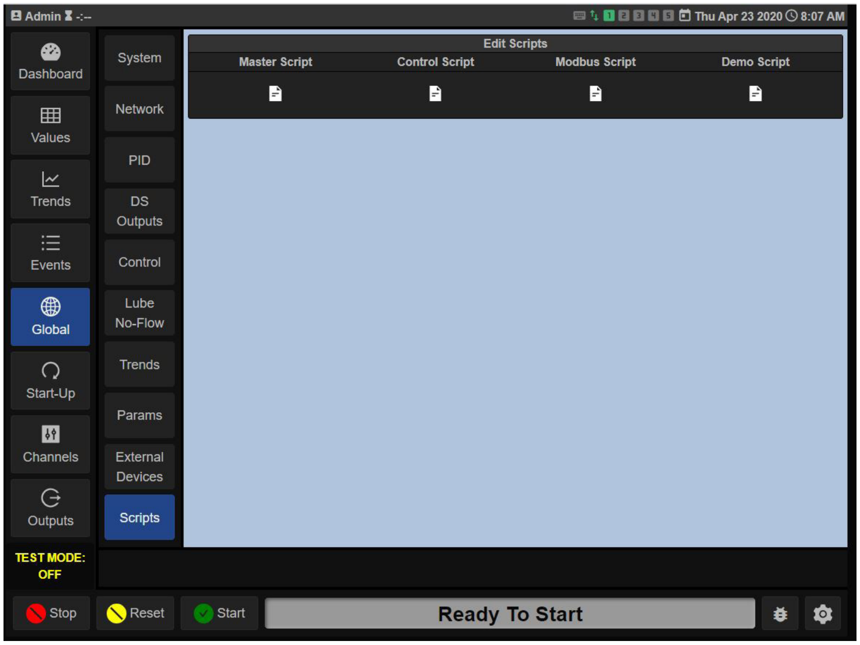
B Timer Start: Crank Start

Test Mode Timer (min): 5

TEST MODE: OFF

Ready To Start

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



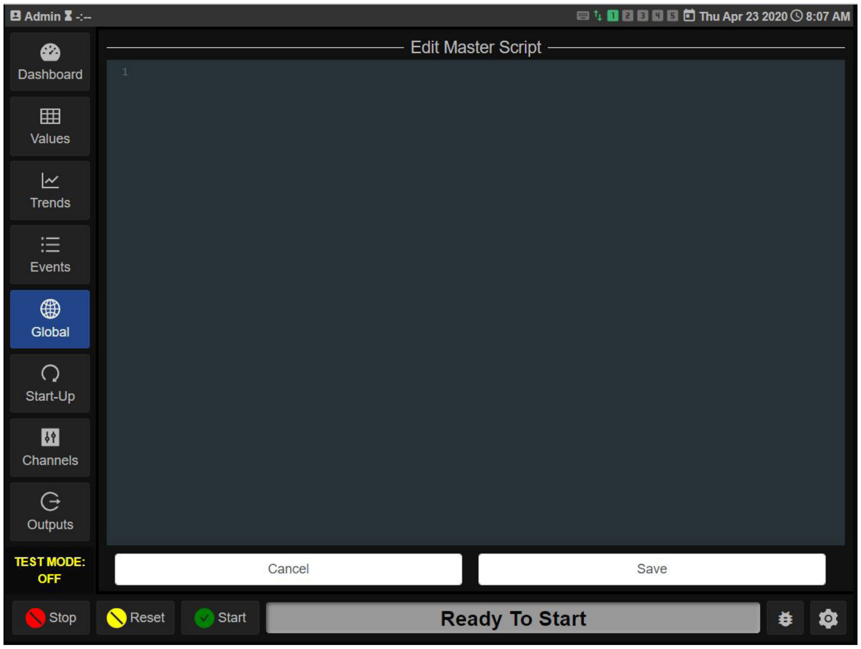
Edit Scripts

Master Script	Control Script	Modbus Script	Demo Script

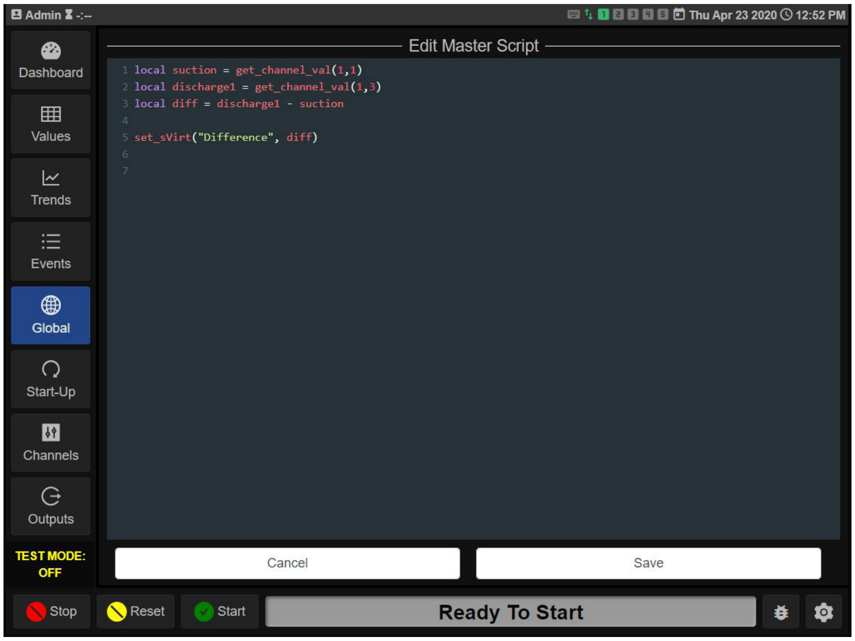
TEST MODE: OFF

Ready To Start

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.



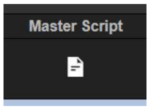
```
local suction = get_channel_val(1,1)
local discharge1 = get_channel_val(1,3)
local diff = discharge1 - suction

set_sVirt("Difference", diff)


```

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)
```

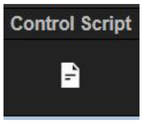
The first line gets the channel value from Terminal board 1 Input 1 and stores it in local variable named **suction.**

The second line gets the channel value from Terminal board 1 Input 3 and stores it in local variable named **discharge1**.

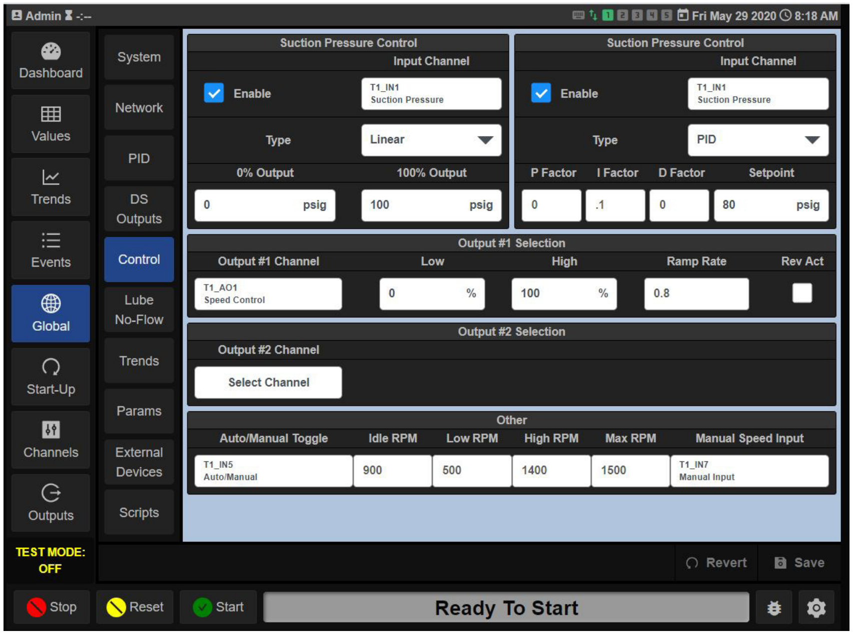
The third line takes the discharge1 pressure and subtracts the suction pressure and stores it in the global variable named **diff** (NOTE: Any value that you want to access from another scripting section must be stored in a global variable. This is used most in calling values into Modbus registers as 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.



Suction Pressure Control

	Input Channel
☑ Enable	T1_IN1 Suction Pressure
Type	Linear
0% Output	100% Output
0 psig	100 psig

Suction Pressure Control

	Input Channel		
☑ Enable	T1_IN1 Suction Pressure		
Type	PID		
P Factor	I Factor	D Factor	Setpoint
0	.1	0	80 psig

Output #1 Selection

Output #1 Channel	Low	High	Ramp Rate	Rev Act
T1_AO1 Speed Control	0 %	100 %	0.8	□

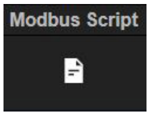
Output #2 Selection

Output #2 Channel
Select Channel

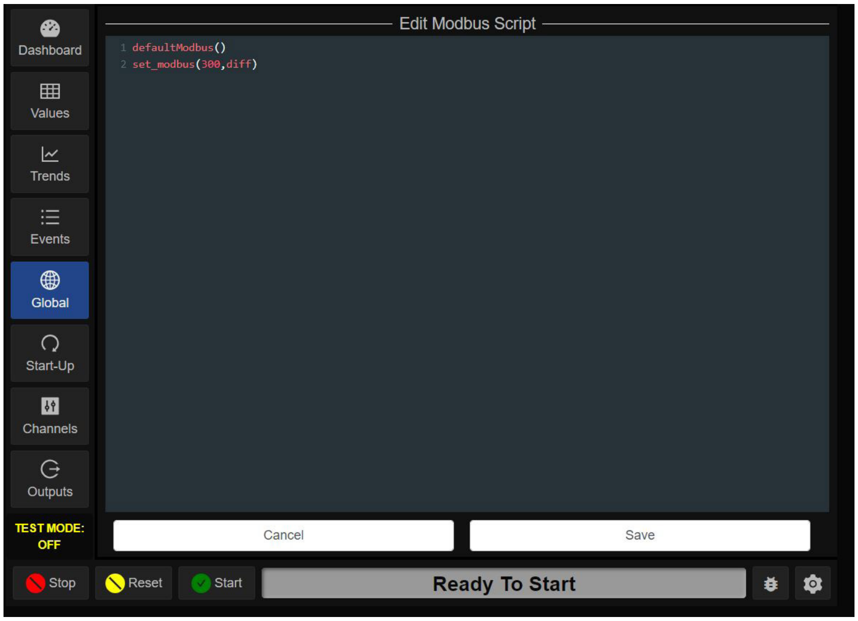
Other

Auto/Manual Toggle	Idle RPM	Low RPM	High RPM	Max RPM	Manual Speed Input
T1_IN5 Auto/Manual	900	500	1400	1500	T1_IN7 Manual Input

- **Modbus Script**



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



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

```
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

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

Create 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")
```

Related function(s): get_param()

get _ channel _ label(terminal,channel)

Return the label for the input channel defined by terminal, channel

Example:

```
-- Read channel label for terminal 1, channel 7
local chanLabel = get _ channel _ label(1,7)
get _ channel _ long _ label(terminal,channel)
```

Return the channel label, but leave off the short label if defined

Note: A channel can be assigned a short label in the DE-4000 channel configuration page. The short label is defined at the end of the channel label and is enclosed in parentheses.

For a suction pressure channel you would define the channel label as: Suction Pressure (SP)

In this case the channel short label is SP

This function will return the long label defined above as Suction Pressure

Example:

```
-- Read channel long label for terminal 1, channel 7
local shortLabel = get _ channel _ short _ label(1,7)
-- Returns "Suction Pressure"
```

get _ channel _ short _ label(terminal,channel)

Return the short label for a channel if defined, otherwise return the channel hash

Note: A channel can be assigned a short label in the DE-4000 channel configuration page.

The short label is defined at the end of the channel label and is enclosed in parentheses.

For a suction pressure channel you would define the channel label as: Suction Pressure (SP)

In this case the channel short label is SP

If the channel short label is not defined the channel hash will be returned. For example, the channel hash for Terminal 1, Input channel 12 is T1:IN12

Example:

```
-- Read channel short label for terminal 1, channel 7
local shortLabel = get _ channel _ short _ label(1,7)
-- Returns "SP"
```

get _ channel _ val(terminal,channel)

Returns current value of analog input channel on terminal module terminal

Return value type is numeric

Example: (reads value of Suction Pressure from terminal module #1, IN5)

```
local sp = get _ channel _ val(1,5)
```

get _ gbl(“index”,default)

Return global config setting stored under **index** or return **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: (get the number of terminal boards installed in the system)

```
local nt = get _ gbl(“NumTerm”,1)
```

get _ modbus(register)

Return the value stored in a 40000 block Modbus register

Note: This function returns values from the 40000 block of registers. In other words, passing the value of 250 into this function will return the value stored at Modbus register 40250

Example:

```
local regVal = get _ modbus(250)
set _ sVirt(“Reg40250”,regVal) --create virtual channel with
                               --value from register 40250
```

Related function(s): set_modbus()

get _ param(“index”)

Return either the default value or the user configured value of the parameter **index**

Example: (get the configured parameter for number of engine cylinders)

```
get _ param(“NumEngCyl”)
```

Related function(s): create_param()

get _ rpm(channel)

Reads the RPM input **channel** in units of revolutions per minute

Note: valid channel numbers are 1 – 10 (2 channels each per 5 possible Terminal Modules)
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: (read RPM1 channel on Terminal Module #1
and RPM2 channel on Terminal Module #3)

```
local engineRPM = get _ rpm(1)
local turbuRPM = get _ rpm(6)
```

get _ sGbl(“index”,{default})

If “index” is defined in the global status table, 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: (get the previously stored value “calculatedPressure”, return 0 if not found)

```
local cp = get _ sGbl(“calculatedPressure”,0)
```

Related function(s): set_sGbl()

get _ state()

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

Example:

```
local engineState = get _ state()
if engineState > 7 then
  set _ timer(“WarmupTimer”,1000)
end
```

get _ sVirt(“index”,default)

Returns the value of virtual channel **index** or returns **default** if the virtual channel does not exist

Example: (get the value of the virtual channel ElapsedTime and set value of status global “timeExceeded” if ElapsedTime is greater then status global “timeLimit”)

```
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
```

Related function(s): set_sVirt()

get _ time()

Return the Unix “epoch” time (defined as number of seconds elapsed since Jan 1, 1970)

Note: you can measure elapsed time by storing a get_time() value at one event and later reading the current get_time() and then subtract the first time from the second time

Example: (store current time if first time through, otherwise calculate elapsed time)

```
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)
```

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 false if time has not been set

Second return value is number of seconds remaining or -1 if timer is not active or -1 if timer has not been set

Example: (if timer is expired, then set global status “timedOut” to true)

```
if not get _ timer(“myTimer”) then
  set _ sGbl(“timedOut”,true)
else
  set _ sGbl(“timedOut”,false)
end
```

Example: (set virtual channel to show number of remaining seconds)

```
local active,remaining = get _ timer(“myTimer”)
if not active then
  set _ sVirt(“timeRemaining”,”Expired”)
else
  set _ sGbl(“timeRemaining”,remaining)
end
```

Related function(s): set_timer()

RandomVariable(length)

Create a string composed of random alpha/numeric text. The length of the returned string is passed in as **length**

Example:

```
-- get a random string 10 characters long
local randomText = RandomVariable(10)
-- returns a random string such as “AIqbFfzQ68”
```

set _ modbus(register,value)

Set a value into a 40000 block Modbus register

Note: This function sets values into the 40000 block of registers. In other words, passing the register parameter of 250 into this function will set the value stored at Modbus register 40250

Example:

```
-- Read channel value at Terminal 1, channel 5 and write
-- to Modbus register 40250
local chanVal = get _ channel _ val(1,5)
set _ modbus(250,chanVal)
```

Related function(s): get_modbus()

set _ sGbl(“index”,value)

Store **value** in the global status table under “index”

Value can be a number or a string but if storing a boolean, use tostring()

Example: (store boolean value “minPressureExceeded”)

```
local mpe = false
local sp = get _ channel _ val(1,5)
if sp > 15 then
  mpe = true
end
set _ sGbl(“minPressureExceeded”,tostring(mpe))
```

Related function(s): get_sGbl()

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: (calculate differential between suction and discharge pressure and assign to virtual channel)

```
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)
```

Related function(s): get_sVirt()

set _ timer(“index”,secs)

Activate timer “index” and set countdown time to **secs**

Example: (create timer “myTimer” and start countdown time at 300 seconds)

```
     set _ timer(“myTimer”,300)
```

Related function(s): get_timer()

APPENDIX:

DEFAULT CONTROL LOOP SCRIPT:

```
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)
local speedRevAct = tonumber_def(get_gbl(“speedRevAct”,0),0)
local recycleRevAct = tonumber_def(get_gbl(“recycleRevAct”,0),0)
local outputLow = tonumber_def(get_gbl(“outputLow”,0),0)
local outputLow2 = tonumber_def(get_gbl(“outputLow2”,0),0)
local outputHigh = tonumber_def(get_gbl(“outputHigh”,0),0)
local outputHigh2 = tonumber_def(get_gbl(“outputHigh2”,0),0)
local spSuctType = get_gbl(“spSuctType”,”linear”)
local spDischType = get_gbl(“spDischType”,”linear”)
local suctPIDPFactor = tonumber_def(get_gbl(“suctPIDPFactor”,0),0)
local suctPIDIFactor = tonumber_def(get_gbl(“suctPIDIFactor”,0),0)
local suctPIDDFactor = tonumber_def(get_gbl(“suctPIDDFactor”,0),0)
local dischPIDPFactor = tonumber_def(get_gbl(“dischPIDPFactor”,0),0)
local dischPIDIFactor = tonumber_def(get_gbl(“dischPIDIFactor”,0),0)
local dischPIDDFactor = tonumber_def(get_gbl(“dischPIDDFactor”,0),0)
local recycleCtrl = false
local recycleSuctionRev = false
local recycleDischargeRev = false
if recycleChan > 0 and recycleTerm > 0 then
  recycleCtrl = true
end
--if recycleCtrl and spSuctType == “linear” and outputLow2 > suctSp then
--  recycleSuctionRev = true
--end
--if recycleCtrl and spDischType == “linear” and  recycleMax < dischSp then
--  recycleDischargeRev = true
--end
--print(“disch: “..tostring(disch)..” suct:”..tostring(suct))
--local suct = 500
local dischPct = 100
local suctPct = 100

local dischOutput = 0
local suctOutput = 0
local rSuctOutput = 0
local rDischOutput = 0
local minLoad = 0
local maxLoad = 100
local minRecycle = 0
local maxRecycle = 100
local speedTarget = get_sGbl(“speedTarget”,0)
local recycleTarget = get_sGbl(“recycleTarget”,0)

function map_range(rangeLow,rangeHigh,input)
  if input <= rangeLow and input <= rangeHigh then
    return 0
  end
  if input >= rangeLow and input >= rangeHigh then
    return 100
  end
  local rangeDiff = math.abs(rangeLow - rangeHigh)
  local min = math.min(rangeLow,rangeHigh)
  local retval = math.abs(input - min) / rangeDiff * 100
  if retval > 100 then retval = 100 end
  if retval < 0 then retval = 0 end
  return retval
end

local suct = false
local suctVal = 0
if tonumber_def(get_gbl(“spSuctEn”,0),0) == 1 then
  if suctTerm > 0 and suctChan > 0 then
    suctVal = get_channel_val(suctTerm,suctChan)
    suct = true
  end
end


if suct then
  if spSuctType == “linear” then
    local suctDiff = suctSp - suctMin
    if suctDiff == 0 then suctDiff = 1 end
    if suctVal < suctSp then
      local suctErr = suctSp - suctVal
      suctPct = suctErr / suctDiff
      if suctPct > 1 then suctPct = 1 end
      if suctPct < 0 then suctPct = 0 end
      suctOutput = (1 - suctPct) * 100
    else
      suctOutput = 100
    end
  else
    set_gbl(“PIDsuctEnable”,1)
    set_gbl(“PIDsuctPFactor”,suctPIDPFactor)
    set_gbl(“PIDsuctIFactor”,suctPIDIFactor)
    set_gbl(“PIDsuctDFactor”,suctPIDDFactor)
    set_gbl(“PIDsuctSp”,suctSp)
    set_gbl(“PIDsuctDeadband”,0.2)
    local suctPidOutput = doPid(“suct”,suctVal)
    suctOutput = suctPidOutput
  end
else
  suctOutput = 100
end

local disch = false
local dischVal = 0
if tonumber_def(get_gbl(“spDischEn”,0),0) == 1 then
  if dischTerm > 0 and dischChan > 0 then
      dischVal = get_channel_val(dischTerm,dischChan)
      disch = true
  end
end
if disch then
  if spDischType == “linear” then
    local dischDiff = dischMax - dischSp
    if dischDiff == 0 then dischDiff = 1 end
    if dischVal > dischSp then
      local dischErr = dischVal - dischSp
      dischPct = dischErr / dischDiff
      if dischPct > 1 then dischPct = 1 end
      if dischPct < 0 then dischPct = 0 end
      dischOutput = (1 - dischPct) * 100
    else
      dischOutput = 100
    end
  else
    set_gbl(“PIDdischEnable”,1)
    set_gbl(“PIDdischPFactor”,dischPIDPFactor)
    set_gbl(“PIDdischIFactor”,dischPIDIFactor)
    set_gbl(“PIDdischDFactor”,dischPIDDFactor)
    set_gbl(“PIDdischSp”,dischSp)
    set_gbl(“PIDdischRevAct”,1)
    set_gbl(“PIDdischDeadband”,0.2)
    local dischPidOutput = doPid(“disch”,dischVal)
    dischOutput = dischPidOutput
  end
else
  dischOutput = 100
end

--print(“suctOutput dischOutput: “..math.floor(suctOutput)..” “..math.floor(dischOutput))

local minOutput = 100
local winning = 0
if suctOutput < minOutput then
  minOutput = suctOutput
  winning = 1
end
if dischOutput < minOutput then
  minOutput = dischOutput
  winning = 2
end

if suctOutput == dischOutput then
  winning = 0
end
```

```
if winning == 0 then
  set_gbl(“PIDsuctMax”,100)
  set_gbl(“PIDdischMax”,100)
end

if winning == 1 then
  set_gbl(“PIDdischMax”,math.min(suctOutput + 2,100))
  set_gbl(“integraldisch”,0)
  set_gbl(“lastErrdisch”,0)
  set_gbl(“outputSumdisch”,0)
  set_gbl(“PIDsuctMax”,100)
end
if winning == 2 then
  set_gbl(“PIDsuctMax”,math.min(dischOutput + 2,100))
  set_gbl(“integralsuct”,0)
  set_gbl(“lastErrsuct”,0)
  set_gbl(“outputSumsuct”,0)
  set_gbl(“PIDdischMax”,100)
end

local recycleMinOutput = minOutput

local manOutput = 0
--*********************************************************************
local manMode = 0
local manTerm = tonumber_def(get_gbl(“manTerm”,0),0)
local manChan = tonumber_def(get_gbl(“manChan”,0),0)
if manTerm > 0 and manChan > 0 then
  local manInput = get_channel_val(manTerm,manChan)
  if manInput > 0.5 then
    manMode = 0
    set_sVirt(“SpeedControl”,”Auto”)
  else
```

```
      manMode = 1
      set_sVirt(“SpeedControl”,”Manual”)
    end
  else
    if get_sVirt(“SpeedControl”,”Auto”) == “Auto” then
      manMode = 0
    else
      manMode = 1
    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(“SpeedTarget”,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
    manSpeed = manSpeed + (si * get_sVirt(“speedBump”,0))
    set_sVirt(“speedBump”,0)
  end

  if get_sVirt(“AutoManBump”,0) > 0 then
    set_sVirt(“SpeedControl”,”Auto”)
    set_sVirt(“AutoManBump”,0)
  end
  if get_sVirt(“AutoManBump”,0) < 0 then
    set_sVirt(“SpeedControl”,”Manual”)
    set_sVirt(“AutoManBump”,0)
  end

  if manMode == 1 then
    local manSpeedTerm = tonumber_def(get_gbl(“manSpeedTerm”,0),0)
    local manSpeedChan = tonumber_def(get_gbl(“manSpeedChan”,0),0)
    if manSpeedTerm > 0 and manSpeedChan > 0 then --*** USE SPEED PCT TO SET
SPEED
      local speedInput = tonumber(get_channel_val(manSpeedTerm,manSpeedChan))
      local speedPct = (speedInput / 5) * 100
      if speedPct > 100kl then speedPct = 100 end
      if speedPct < 0 then speedPct = 0 end
      manOutput = speedPct
      manSpeed = math.floor((speedPct / 100) * diff + lowSpeed + 0.5)
    else -- Use ManualSpeed to set speed
      manOutput = ((manSpeed - lowSpeed) / diff) * 100.0
      if manOutput < 0 then manOutput = 0 end
      if manOutput > 100 then manOutput = 100 end
    end
    minOutput = manOutput
  else
    --speedTarget =
    local stRpm = (speedTarget/100) * maxDiff + idleSpeed
    if stRpm < lowSpeed then stRpm = lowSpeed end
    if stRpm > highSpeed then stRpm = highSpeed end
    manSpeed = math.floor(stRpm)
  end
  --speedTarget = get_sGbl(“speedTarget”,0)
  if manSpeed < lowSpeed then
    manSpeed = lowSpeed
  end
  if manSpeed > highSpeed then
    manSpeed = highSpeed
  end

  set_sVirt(“ManualSpeed”,manSpeed)

  --*******************************************************************

  local output1 = 0
  local output2 = 0
  if spSuctType == “pid” or spDischType == “pid” then
    --Map minOutput to output1
    output1 = map_range(outputLow,outputHigh,minOutput)
    set_sVirt(“out1”,output1)
    --Map minOutput to ourput2
    output2 = map_range(outputLow2,outputHigh2,recycleMinOutput)
    set_sVirt(“out2”,output2)
    local hasRPM = idleSpeed > 0 and lowSpeed > 0 and highSpeed > 0 and max-
Speed > 0
    if outputTerm and outputChan then
      if hasRPM then
        local speedRpm = output1 / 100  * (highSpeed - lowSpeed) + lowSpeed
        --set_ao_val(outputTerm,outputChan,(speedRpm - idleSpeed) / (maxSpeed
- idleSpeed) * 100)
        speedTarget = (speedRpm - idleSpeed) / (maxSpeed - idleSpeed) * 100
      else
        --set_ao_val(outputTerm,outputChan,output1)
        speedTarget = output1
      end
    end
    if recycleTerm and recycleChan then
      set_ao_val(recycleTerm,recycleChan,output2)
    end

    if get_state() == 9 then
      speedTarget = get_sGbl(“speedTarget”,0)
      if speedTarget > 0 then speedTarget = speedTarget - rampRate1 end
      if speedTarget < 0 then speedTarget = 0 end
    end
    if get_state() < 8 then speedTarget = 0 end
    set_sGbl(“speedTarget”,speedTarget)
    --set_sGbl(“a”..outputChan,speedTarget)
    set_ao_val(outputTerm,outputChan,speedTarget)
    --set_ao_val(outputChan,speedTarget)
    --print(suctOutput..” “..dischOutput..” “..speedTarget)
    set_sVirt(“spTarget”,speedTarget)
    --set_speed_val(1,speedTarget)

    if hasRPM then
      local sRpm = (speedTarget/100) * maxDiff + idleSpeed
      set_sVirt(“Speed Target”,math.floor(sRpm + 0.5))
    end



  else

    -- Remember that minOutput is 0 - 100 pct of lowSpeed <-> highSpeed
    -- We need to convert this to 0 - 100 pct of idleSpeed <-> maxSpeed
    local suctPct = map_range(outputLow,outputHigh,minOutput)
    local speedRpm = suctPct / 100  * (highSpeed - lowSpeed) + lowSpeed
    minOutput = (speedRpm - idleSpeed) / (maxSpeed - idleSpeed) * 100
```

```
if minOutput <= speedTarget then
  speedTarget = speedTarget - rampRate1
  if speedTarget < minOutput then speedTarget = minOutput end
else
    speedTarget = speedTarget + rampRate1
    if speedTarget > minOutput then speedTarget = minOutput end
    if speedTarget > maxLoad then speedTarget = maxLoad end
end
if speedTarget > maxLoad then speedTarget = maxLoad end
if speedTarget < minLoad then speedTarget = minLoad end

if recycleCtrl then
  local recyclePct = map_range(outputLow2,outputHigh2,recycleMinOutput)
  --if recycleRevAct == 1 then recyclePct = 100 - recyclePct end
  if recyclePct <= recycleTarget then
    recycleTarget = recycleTarget - rampRate2
    if recycleTarget < recyclePct then recycleTarget = recyclePct end
  else
    recycleTarget = recycleTarget + rampRate2
    if recycleTarget > recyclePct then recycleTarget = recyclePct end
  end
  if recycleTarget > maxRecycle then recycleTarget = maxRecycle end
  if recycleTarget < minRecycle then recycleTarget = minRecycle end
  local recycleOutput = recycleTarget
  if get_state() < 8 then
    recycleTarget = 0
  end
  if recycleRevAct == 1 then
    recycleOutput = 100 - recycleOutput
  end
  --set_sGbl(“a”..recycleChan,recycleOutput)
  set_ao_val(recycleTerm,recycleChan,recycleOutput)
```

```
      set_sGbl(“recycleTarget”,recycleTarget)
      set_sVirt(“recycleTarget”,recycleTarget)
    end

    if get_state() == 9 then
      speedTarget = get_sGbl(“speedTarget”,0)
      if speedTarget > 0 then speedTarget = speedTarget - rampRate1 end
      if speedTarget < 0 then speedTarget = 0 end
    end
    if get_state() < 8 then speedTarget = 0 end
    set_sGbl(“speedTarget”,speedTarget)
    --set_sGbl(“a”..outputChan,speedTarget)
    set_ao_val(outputTerm,outputChan,speedTarget)
    --set_ao_val(outputChan,speedTarget)
    --print(suctOutput..” “..dischOutput..” “..speedTarget)
    set_sVirt(“spTarget”,speedTarget)
    --set_speed_val(1,speedTarget)
    local sRpm = (speedTarget/100) * maxDiff + idleSpeed
    set_sVirt(“Speed Target”,math.floor(sRpm + 0.5))
end
```