DE-4000 SCRIPTING REFERENCE MANUAL

Form:DE-4000 SCRIPTING REFERENCE MANUAL:06-26

ALTRONIC

Table of Contents

DE-4000 SCRIPTING REFERENCE MANUAL

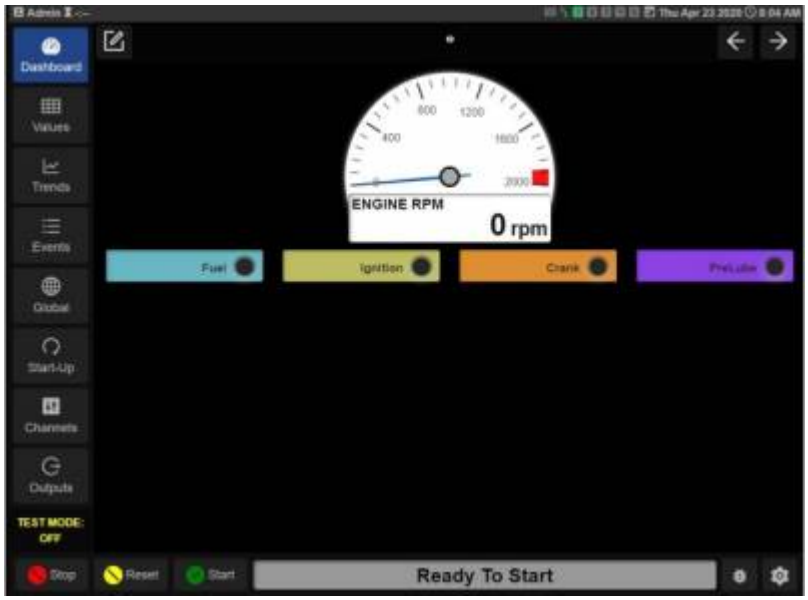
There is a delicate balance between providing a system that has capabilities that can be configured through a fixed set of options, and one that can be extended and expanded with custom programming. In designing the DE-4000 control system, the choice was made to provide a system where most applications 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.

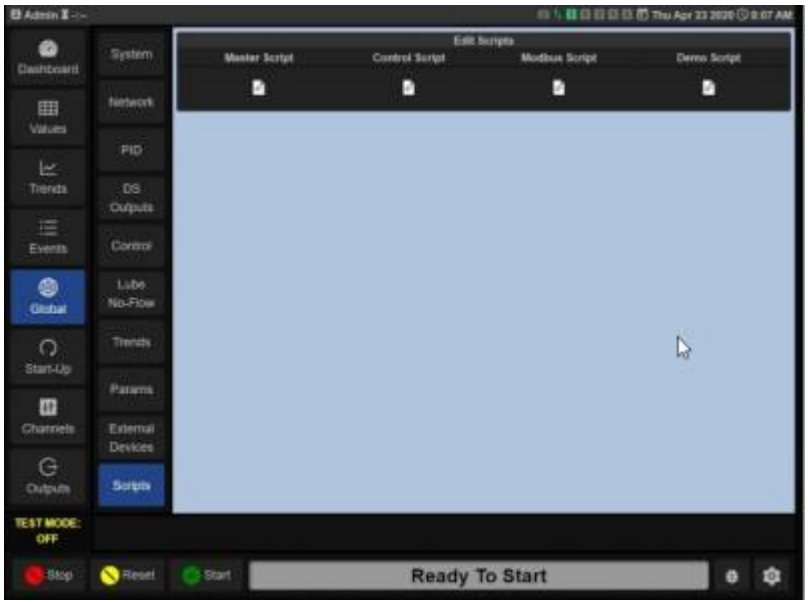
GETTING STARTED WITH DE-4000 SCRIPTS Basic Scripting on DE-4000

1. Begin on Dashboard on DE-4000 system environment



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

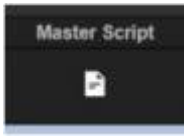
3. In the Sub-Menu on the Left side select “Scripts”



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

Scripting Windows and examples

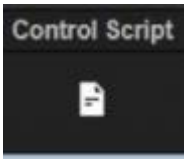
Master Script 

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

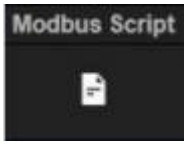
Example:

```
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.

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

DE4000 Lua Script API

CUSTOM FUNCTIONS FOR SCRIPTING

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

---

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.

---

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

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:

```
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
