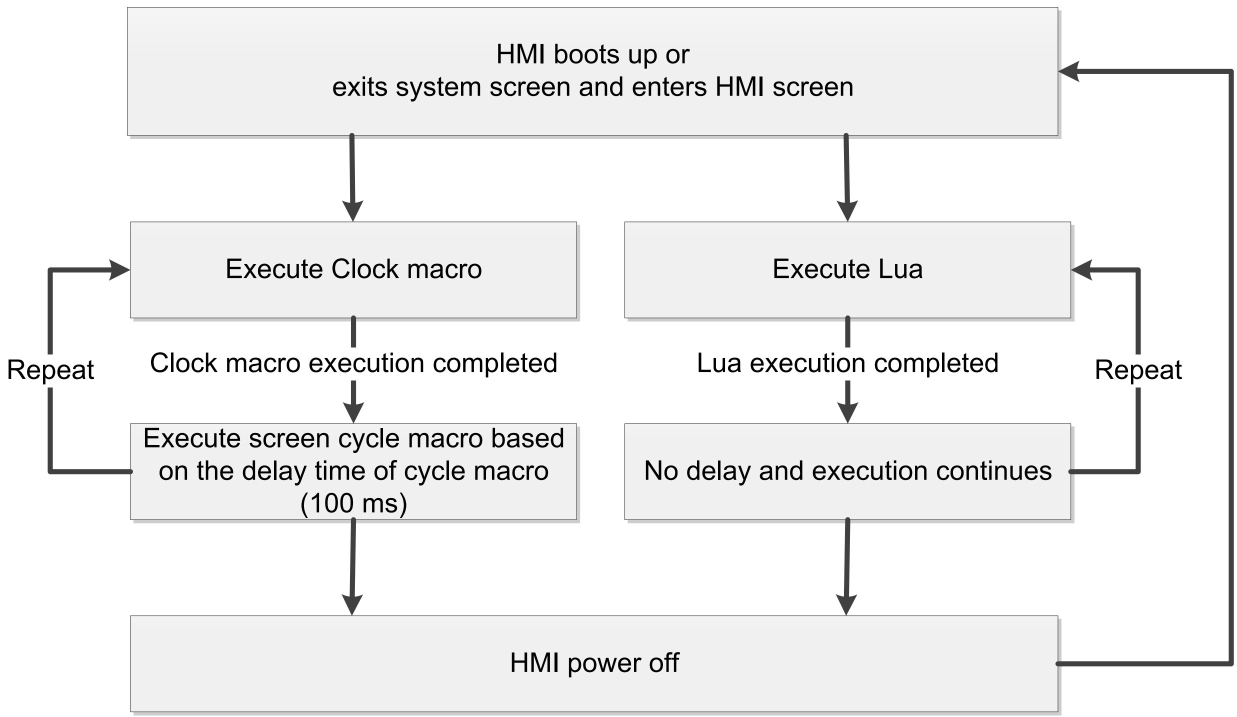
1. Lua program execution function

Lua is a programming language similar to Python and JavaScript. Its syntax is easier and more user-friendly than the macros provided by the HMI. You can use simple instructions to complete complicated computing and develop variable functions, as well as programming the functions on your own, which makes the programming more flexible and easier to meet the application requirements.

The Lua program runs repeatedly during the HMI operation, which is similar to the HMI Clock macro and can run with other HMI macros at the same time without affecting the execution efficiency of each.

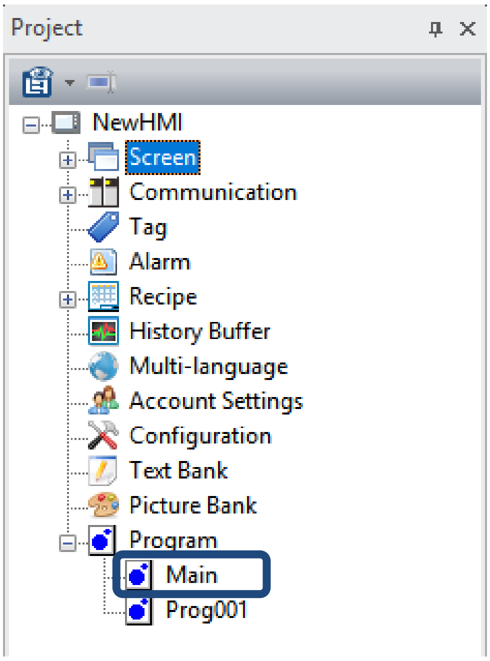


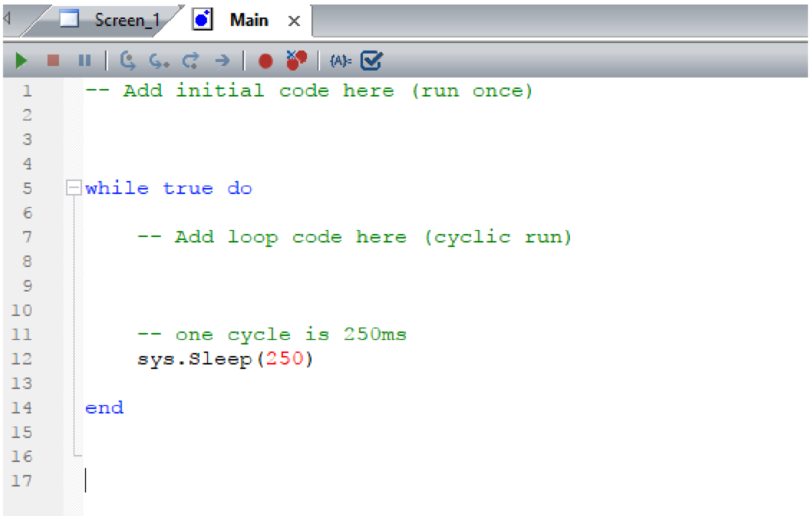
The Lua program editing interface and the IDE (Integrated Development Environment) are similar. In addition to the program editing function, it has a debug function that allows you to run the program on the HMI or simulate online (with a simulator) to check if the Lua program is suitable for the circumstance. The debug function can set the breakpoint, run the code line by line, as well as monitor the variables for quick modification of the program.

Please refer to Table 1.1 Lua programming example.

Table 1.1 Lua programming example

<table>
<tr><th colspan="2">1. Lua programming</th></tr>
<tr><td>Edit [Main] program</td><td>Step 1: double-click the [Main] tab in the DOPSoft project tree to enter the editing interface.





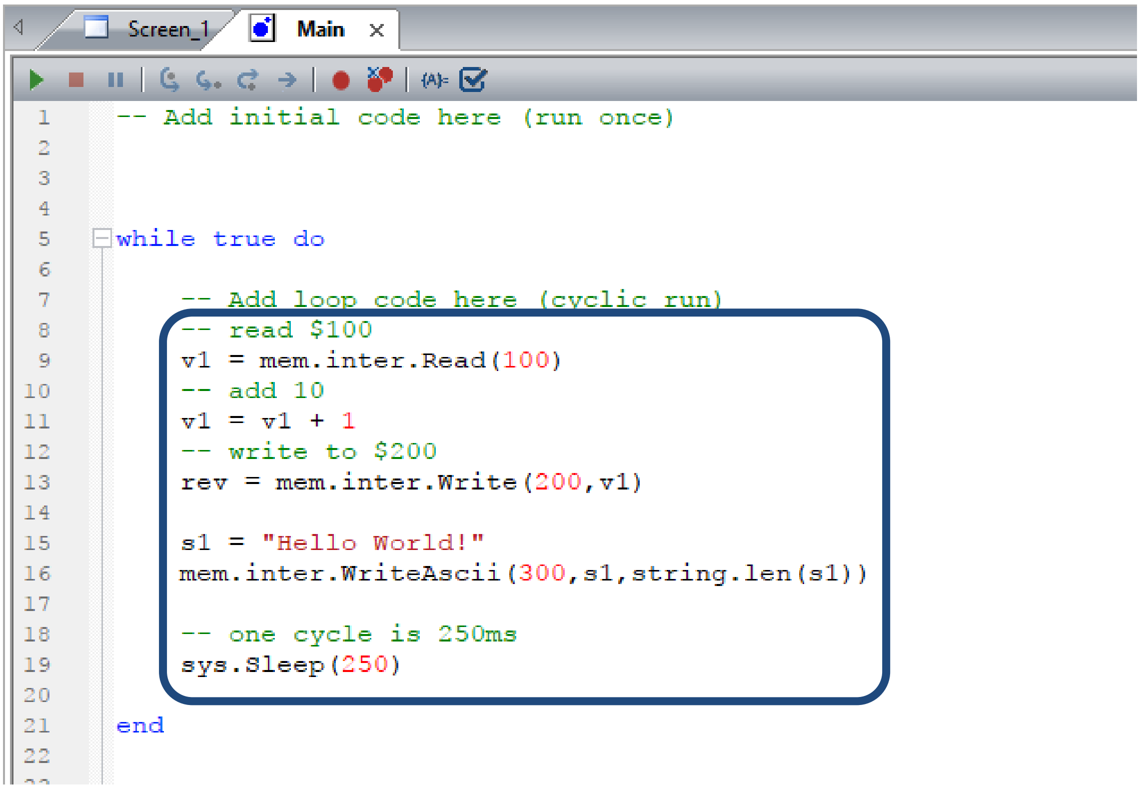
</td></tr>
</table>

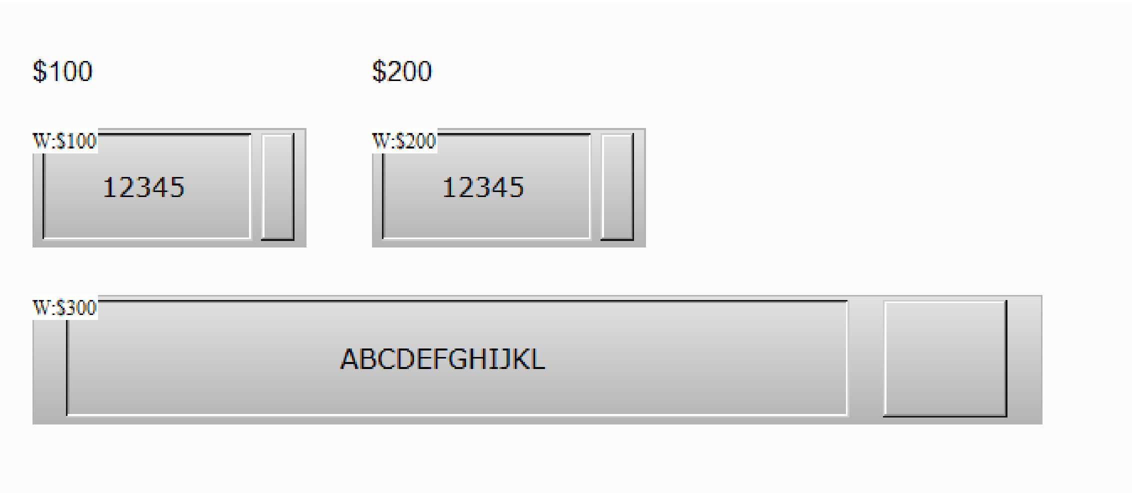
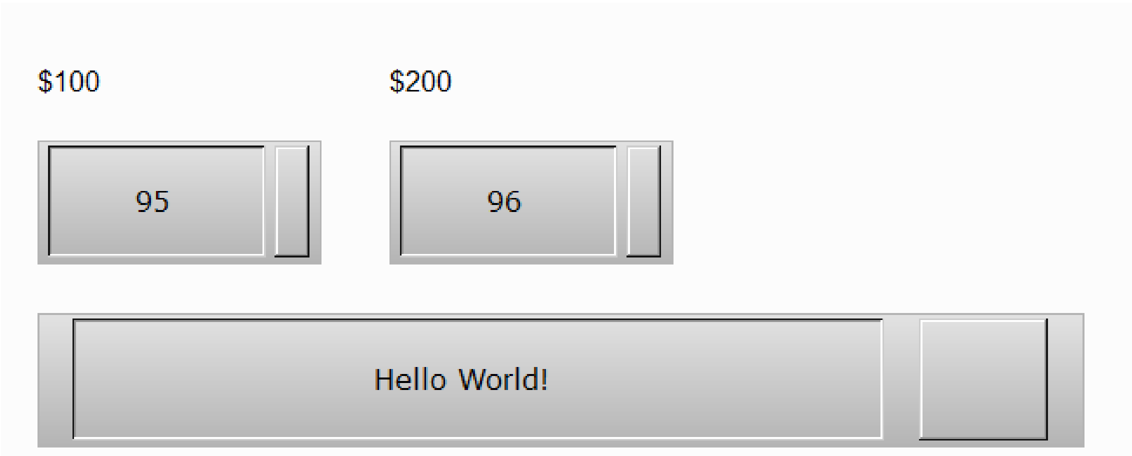
1. Lua programming

Edit [Main] program

Step 2: add the following string to the [Main] program.
This program reads the value from $100 and write this value plus 1 to $200. Next, write the string "Hello World!" to $300.

```
-- read $100
v1 = mem.inter.Read(100)
-- add 10
v1 = v1 + 1
-- write to $200
rev = mem.inter.Write(200,v1)

s1 = "Hello World!"
mem.inter.WriteAscii(300,s1,string.len(s1))
```

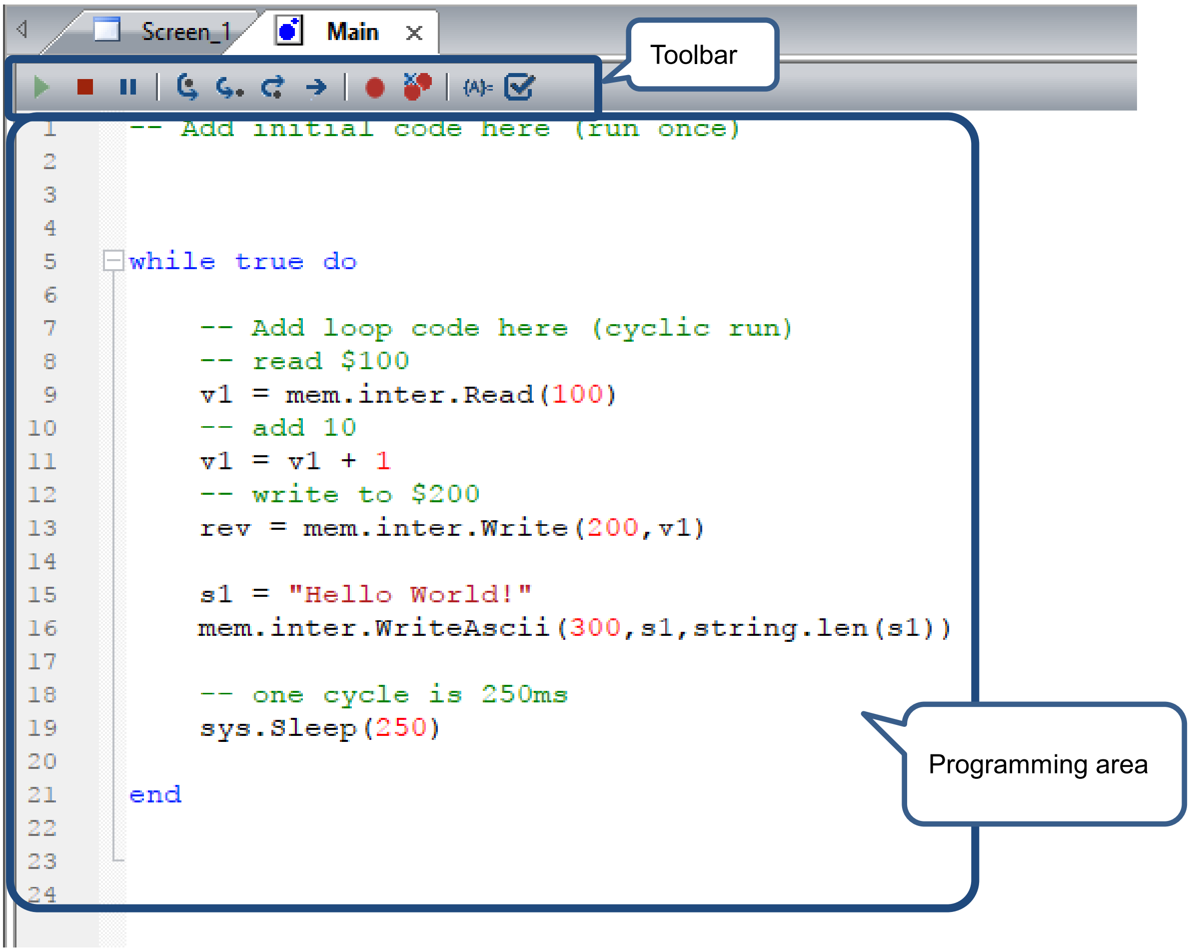


1. Lua programming	
Edit HMI screen	Go to Screen_1 to create two Numeric Entry elements with the read addresses as $100 and $200 respectively. Then, create one Character Entry element with the read address as $300 and the string length as 12. 
Execution results	Step 1: download this project to the HMI. The display is as follows after HMI power-on.  Step 2: change the value of $100 to 95, and the value of $200 changes to 96 immediately. 

1.1 Lua program window

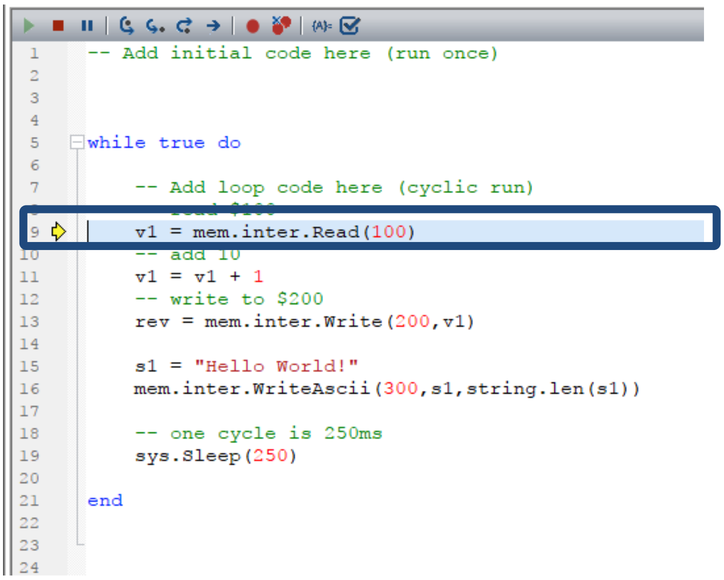
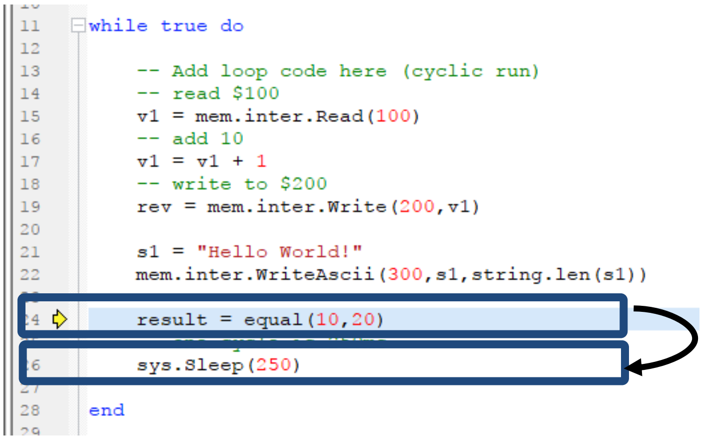
1.1.1 Lua programming window

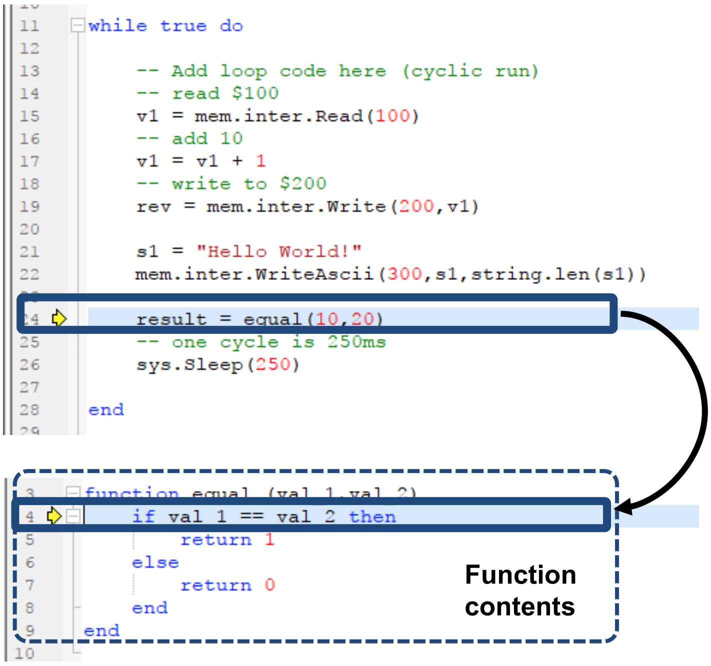
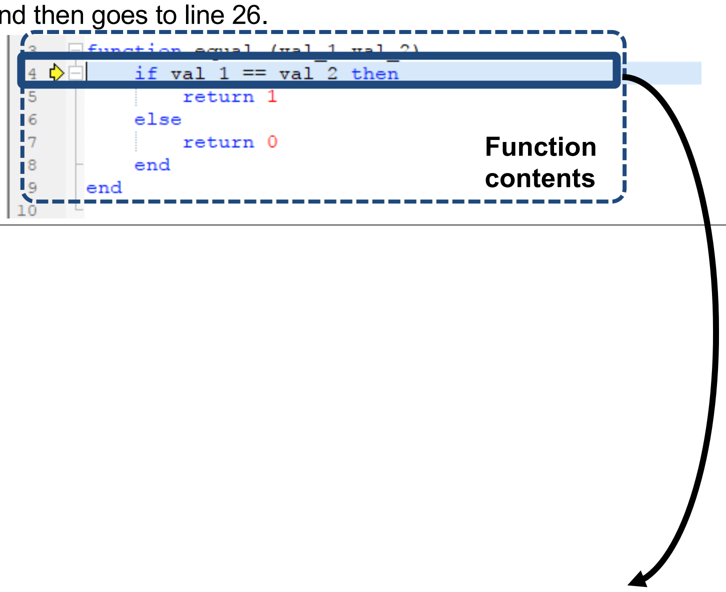
Double-click on the Lua Program to open the Lua programming window as shown in the figure below.

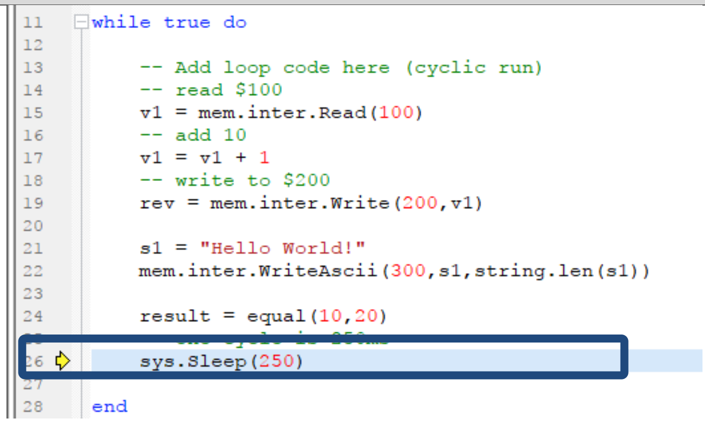
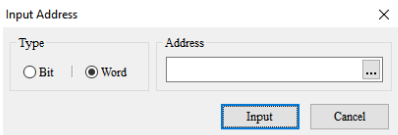
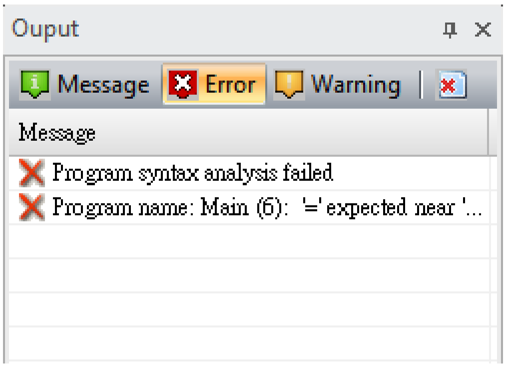


Toolbar function description for the Lua programming window is as follows:

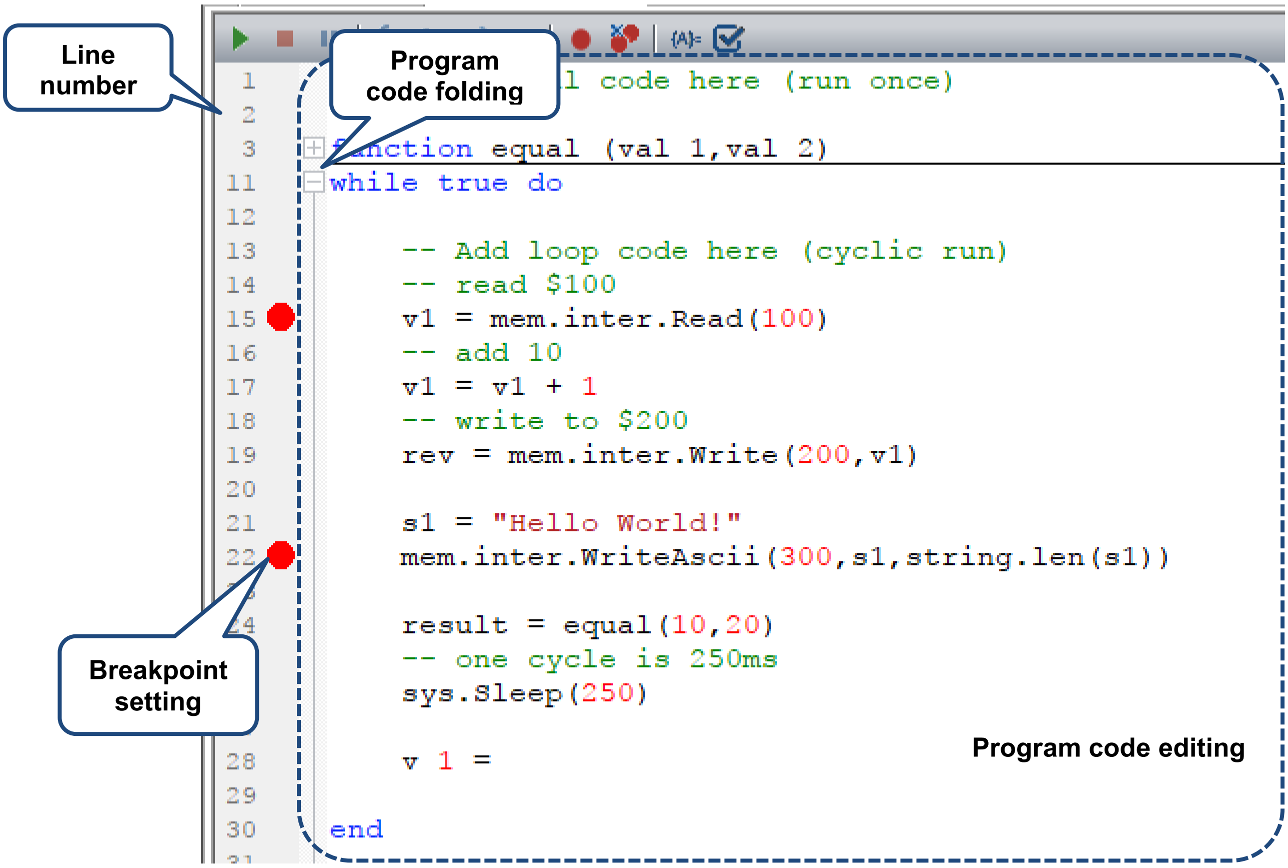
Toolbar for the Lua programming window			
Symbol	**Name**	**Hotkey**	**Description**
	Start online program debugging	F5	Enable online debugging of the Lua program; the debugging target can be the HMI or simulator.
	Stop program debugging	Shift+F5	Stop the current Lua program in execution.

<table>
<tr><th colspan="4">Toolbar for the Lua programming window</th></tr>
<tr><th>Symbol</th><th>Name</th><th>Hotkey</th><th>Description</th></tr>
<tr><td></td><td>Pause program debugging</td><td>-</td><td>Pause the current Lua program in execution. When pausing, a yellow arrow points to the the next instruction to be executed.

</td></tr>
<tr><td></td><td>Run line by line</td><td>F10</td><td>Run line by line. If the instruction to be executed is a function, this instruction is executed completely. In the example below, if you execute [Run line by line] at line 24, the program jumps to line 26.

</td></tr>
</table>

<table>
<tr><th colspan="4">Toolbar for the Lua programming window</th></tr>
<tr><th>Symbol</th><th>Name</th><th>Hotkey</th><th>Description</th></tr>
<tr><td></td><td>Jump into</td><td>F11</td><td>When the instruction to be executed is a function, jump into the function to be executed. In the example below, line 24 “result = equal (10,20)” is a function, and its contents are in line 4 - 8. At line 24, if you execute “Jump into”, it jumps to execute line 4.
</td></tr>
<tr><td></td><td>Jump out</td><td>Shift+F11</td><td>If the instructions in the function are in execution, jump out of the current function and go to the next instruction. As shown in the example below, the function contents are in line 4 - 8 that are called by line 24. Execute “Jump out” at line 4, it automatically runs line 4 - 8 and then goes to line 26.
</td></tr>
</table>

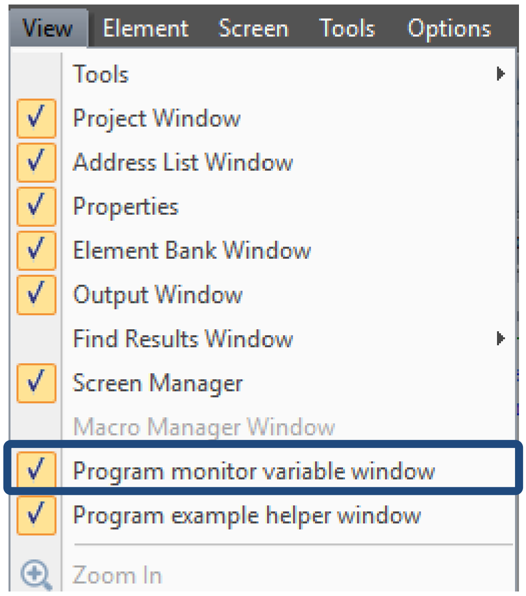
Toolbar for the Lua programming window			
Symbol	**Name**	**Hotkey**	**Description**
			
	Resume	F5	Carry on executing the program after the program pauses.
	Breakpoint	F9	Set the breakpoint. During online debugging, the program execution pauses at a breakpoint. You can set multiple breakpoints at the same time.
	Delete all breakpoints		Delete all breakpoints in the program.
	Input Address		Open the Input Address window to input the bit or word address. 
	Program syntax check		Check if the Lua syntax in the project is correct. If there is any incorrect syntax, the output window displays the error description. 

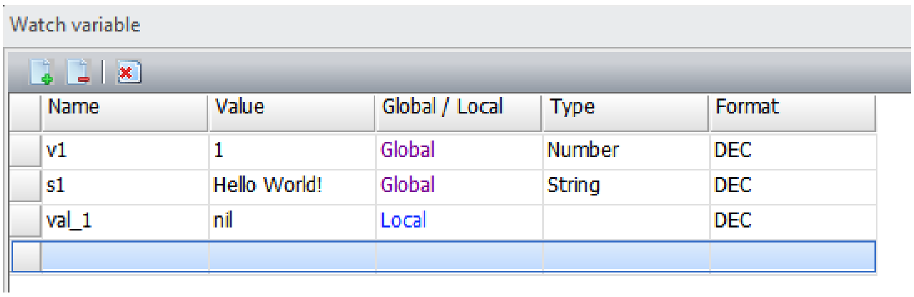
The Lua program editing window includes four parts, “Line number”, “Breakpoint setting”, “Program code folding”, and “program code editing”. You can click on [Breakpoint setting] to set or cancel the breakpoints.



1.1.2 Program monitor variable window

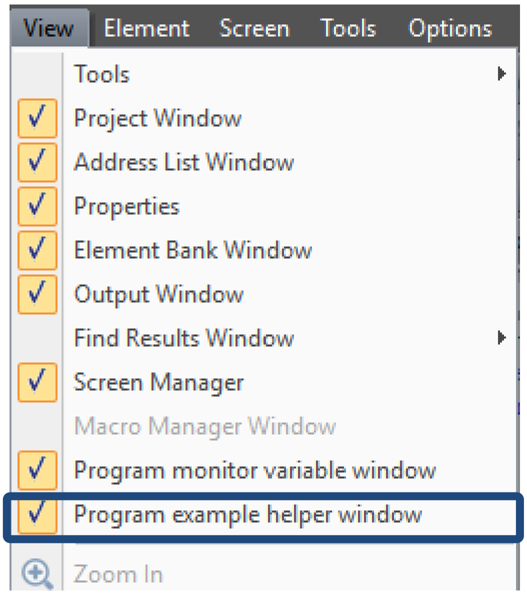
You can go to [View] to open the [Program monitor variable window]. This window allows you to monitor the Lua program execution results by specifying the variables. You can also change the variable in this window. For the operation example, please refer to the description in 3.2 Debug mode.





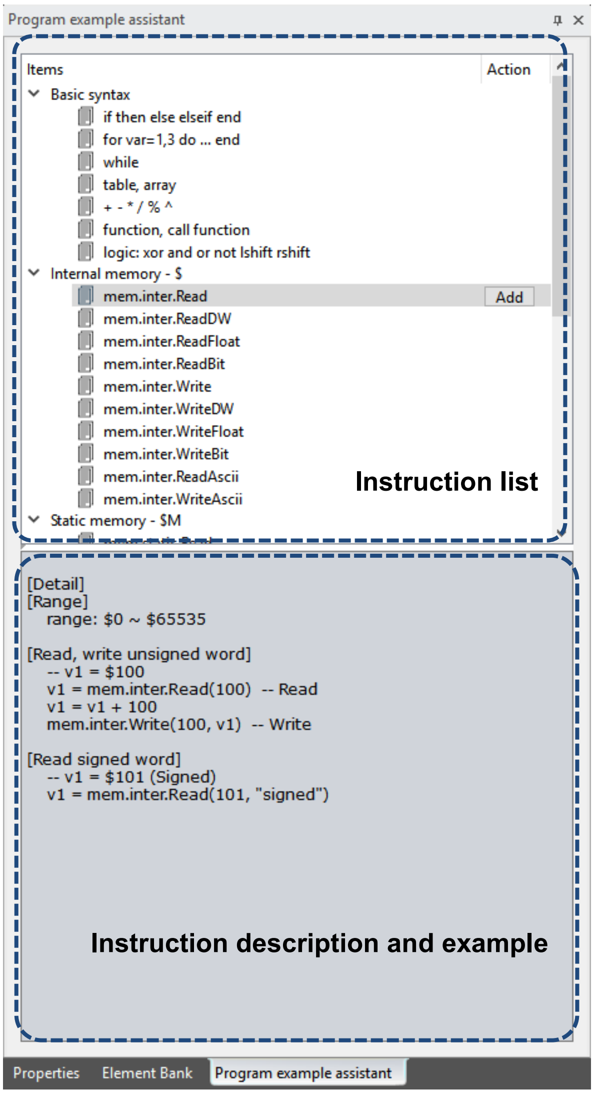
1.1.3 Program example helper window

You can go to [View] to open the [Program example helper window].



This window lists all the Lua functions, including Basic syntax, Internal memory -$ (for reading and writing), Static memory- $M (internal register for non-volatile reading and writing), External link (reading / writing address for controller), File read / write / export / list, Math, Screen (for screen operation), String, System library, Serial port communication, Text encoding, and Utility.

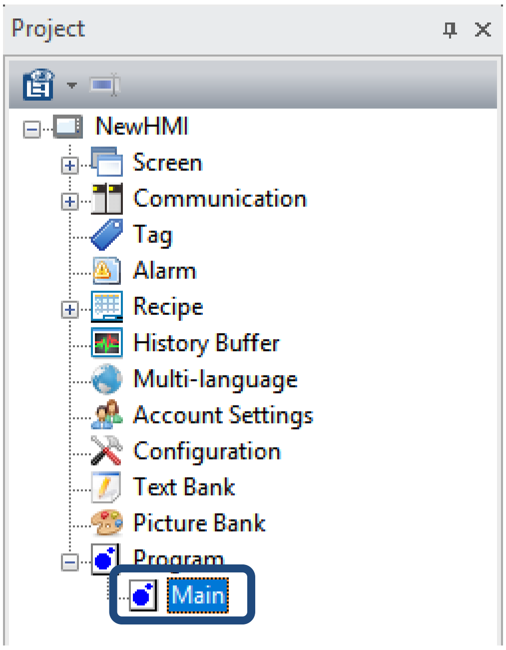
The window has two parts, the upper part is the instruction list and the lower is the description and example of each instruction.

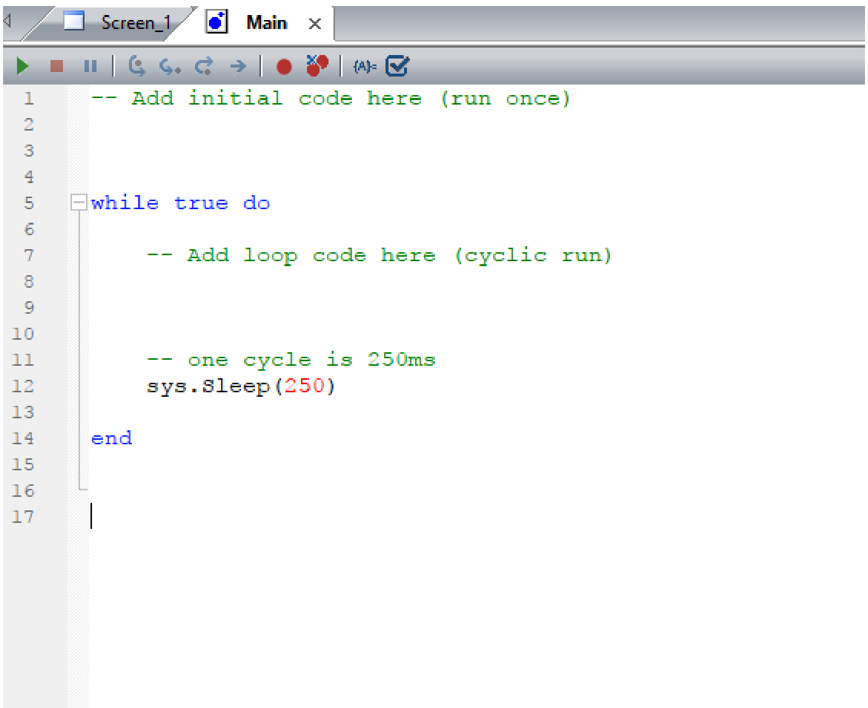


By clicking the **Add** button, you can add the instruction example in the Lua program. Please refer to Table 3.2 Example of adding Lua instruction.

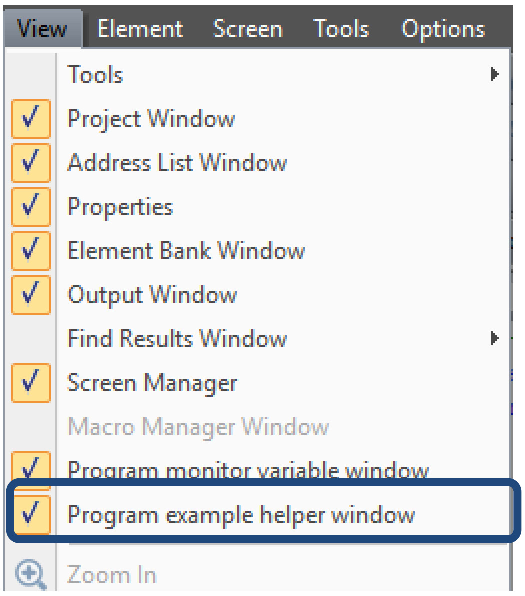
Table 1.2 Example of adding Lua instruction

<table>
<tr><th colspan="2">1. Lua program</th></tr>
<tr><td>Open programming window</td><td>Step 1: double-click the [Main] tab in the DOPSoft project tree to enter the editing interface.

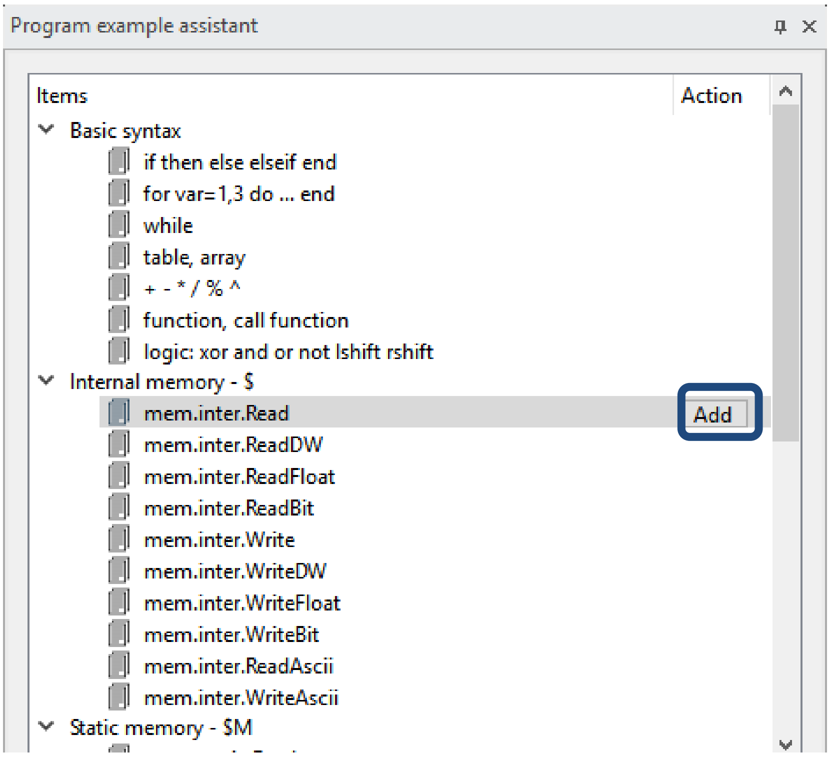




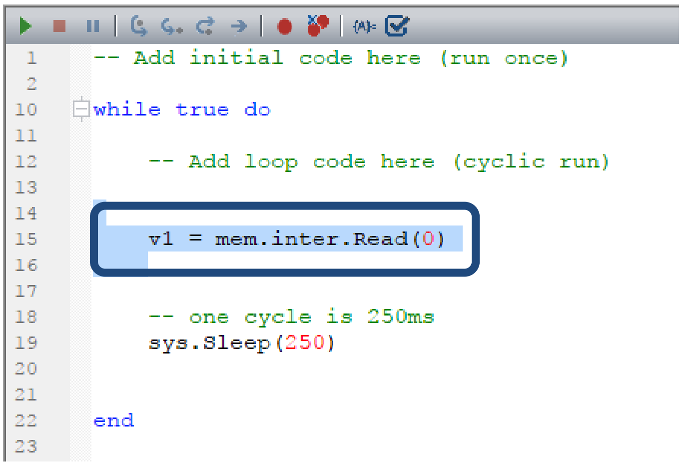
</td></tr>
</table>

1. Lua program	
Open programming window	Step 2: go to [View] to open the [Program example helper window] 

<table>
<tr><th colspan="2">1. Lua program</th></tr>
<tr><td>Add Lua instruction mem.inter.Read</td><td>
Step 1: execute [Internal memory - $] > [mem.inter.Read] > click the Add button.



Step 2: a new instruction is added to the Lua program editing interface. Users can edit the program by referring to this example.

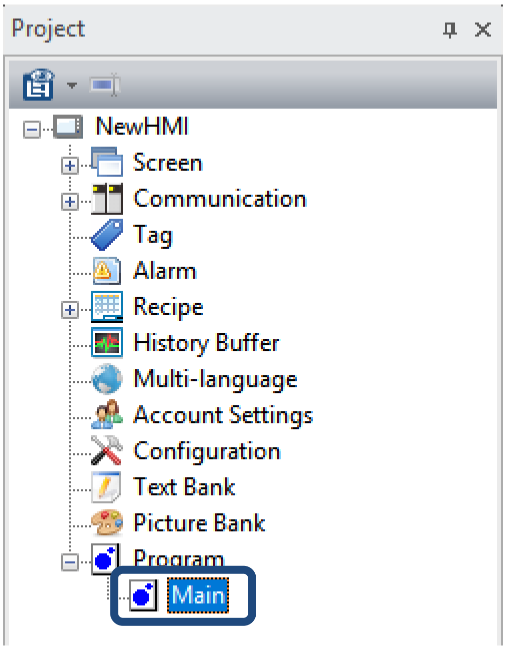
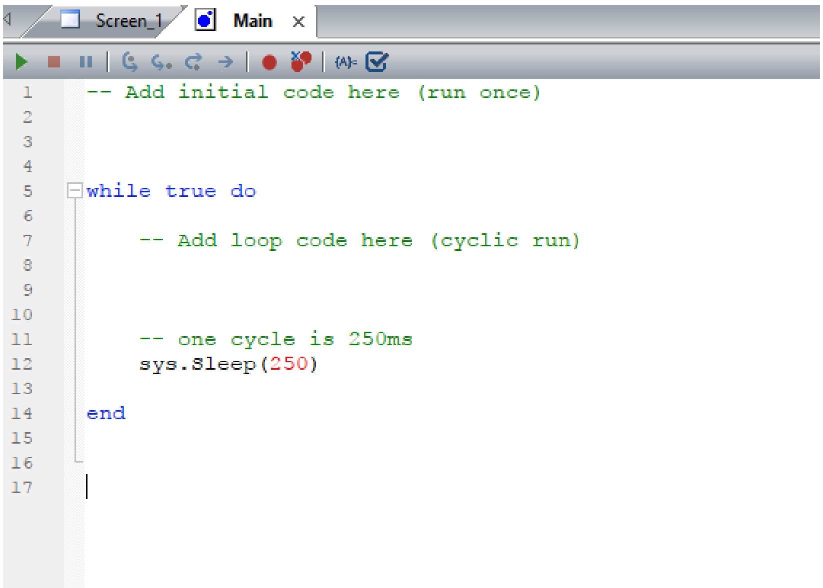


</td></tr>
</table>

1.2 Debug mode

For the Lua program debugging method, please refer to Table 3.3 Lua program debugging example.

Table 1.3 Lua program debugging example

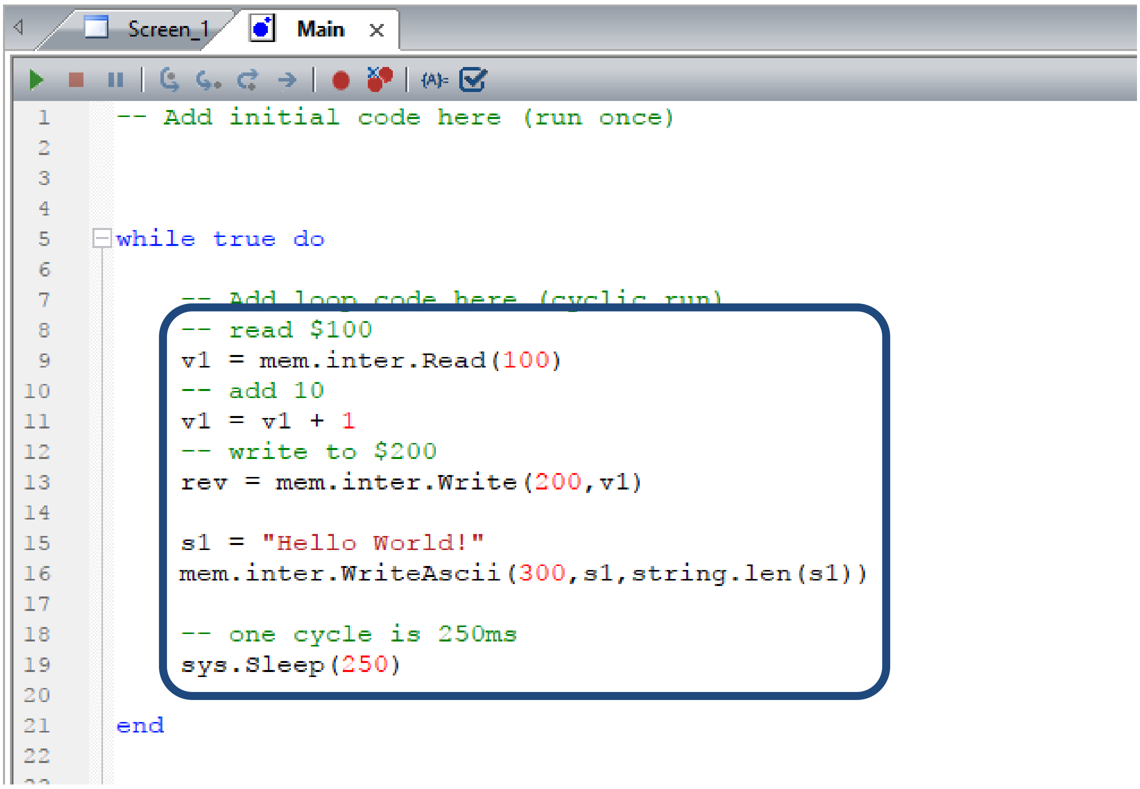
1. Lua programming	
Edit [Main] program	Step 1: double-click the [Main] tab in the DOPSoft project tree to enter the editing interface.  

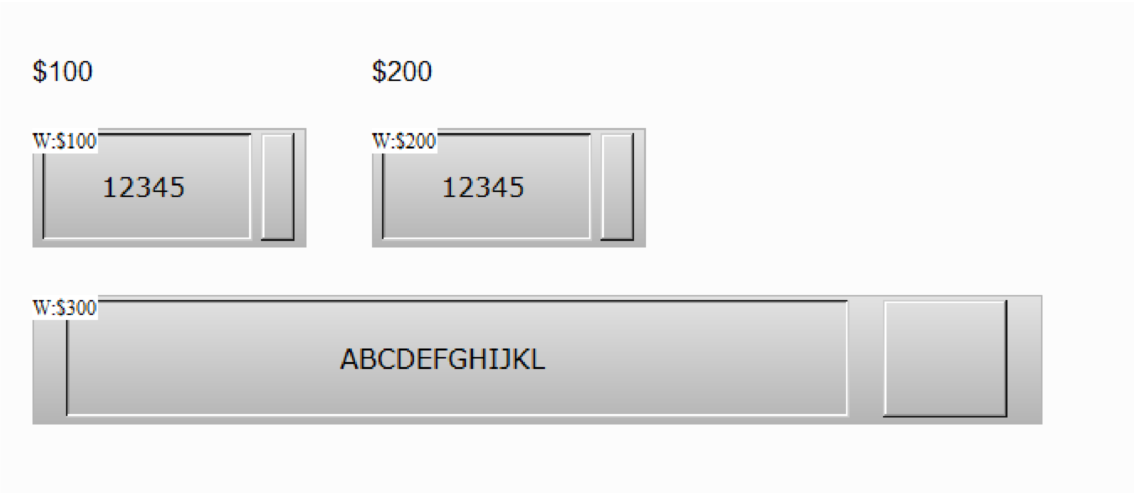
1. Lua programming

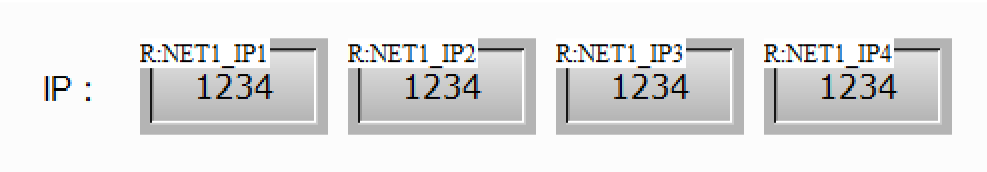
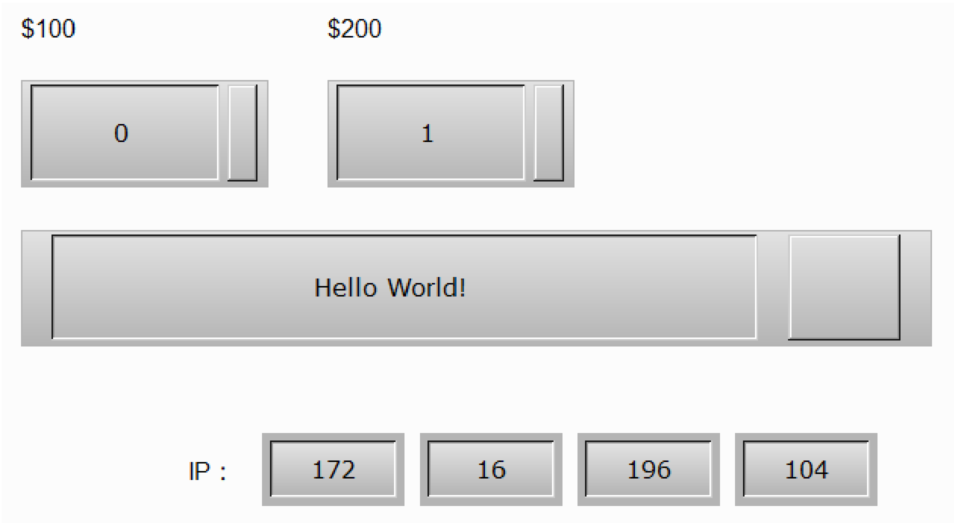
Edit [Main] program

Step 2: add the following string to the [Main] program.
This program reads the value from $100 and write this value plus 1 to $200. Next, write the string "Hello World!" to $300.

```
-- read $100
v1 = mem.inter.Read(100)
-- add 10
v1 = v1 + 1
-- write to $200
rev = mem.inter.Write(200,v1)

s1 = "Hello World!"
mem.inter.WriteAscii(300,s1,string.len(s1))
```



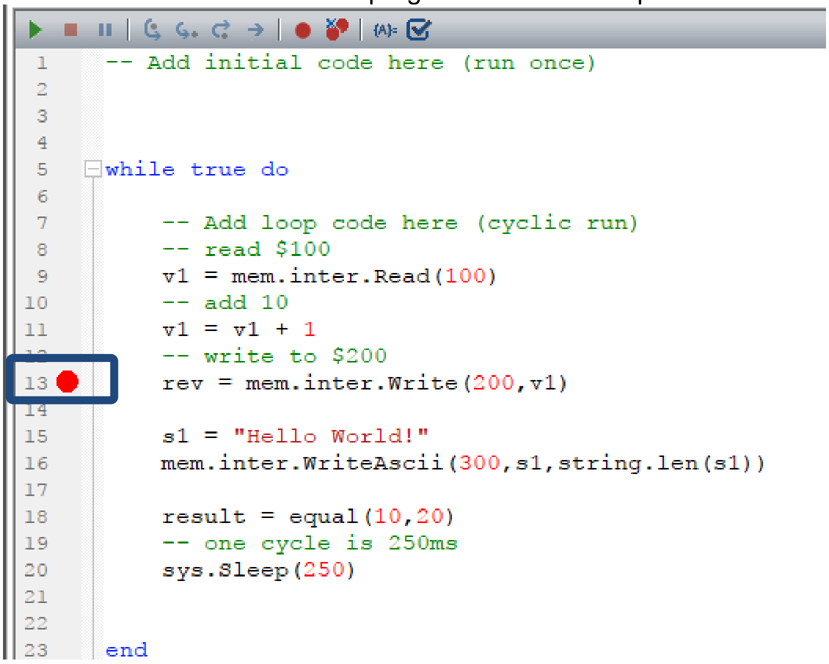
<table>
<tr><th colspan="2">1. Lua programming</th></tr>
<tr><td>Edit HMI screen</td><td>Step 1: go to Screen_1 to create two Numeric Entry elements and set the read addresses as $100 and $200 respectively. Then, create one Character Entry element with the read address as $300 and the string length as 12.


Step 2: create four numeric display elements on Screen_1. Set the read address as NET1_IP1, NET1_IP2, NET1_IP3, and NET1_IP4 of Internal Parameter.

</td></tr>
<tr><td>Execution results</td><td>Step 1: download the project to the HMI. The HMI display is as follows:

</td></tr>
</table>

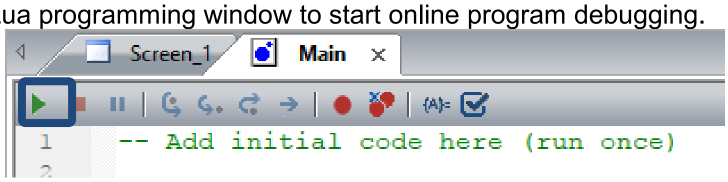
1. Lua programming

Execution results

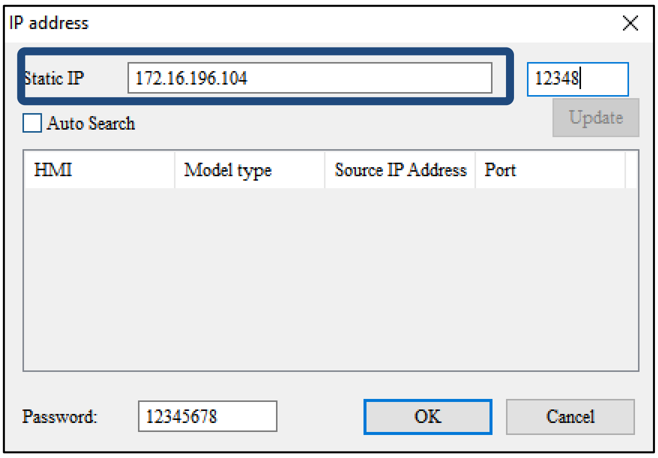
Step 2: click the left side of line 13 in the Lua program to set a breakpoint.



Step 3: go to the Lua programming window to start online program debugging.

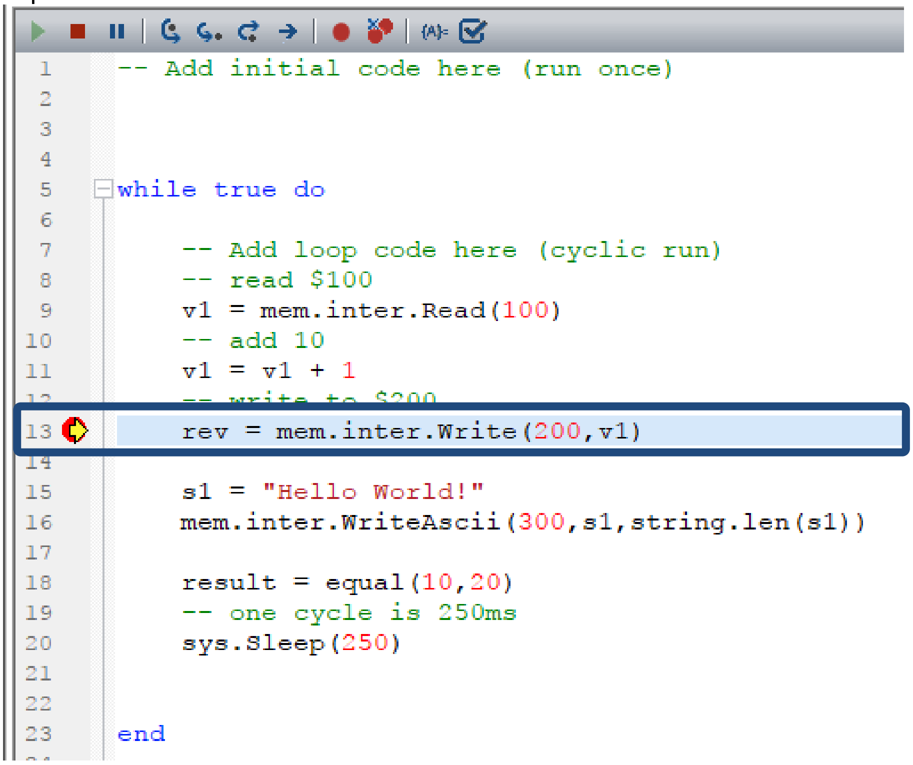


Step 4: specify the address as the HMI IP.

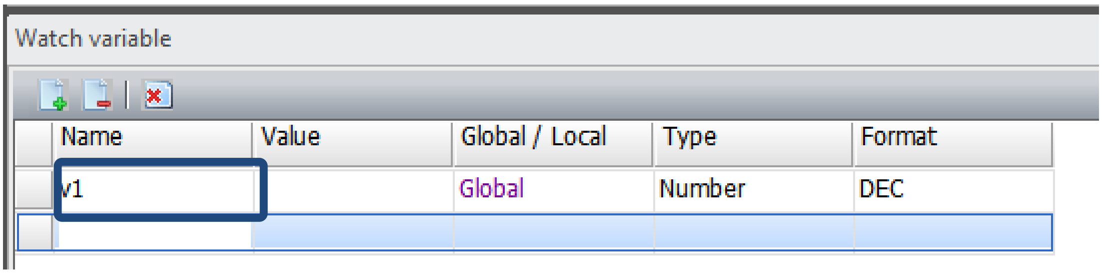


1. Lua programming

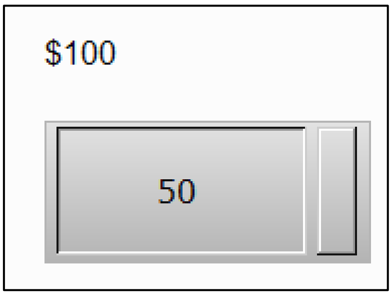
Execution results	Step 5: click **OK** and the Lua program starts running online. Then, it stops at line 13 where the breakpoint is placed.



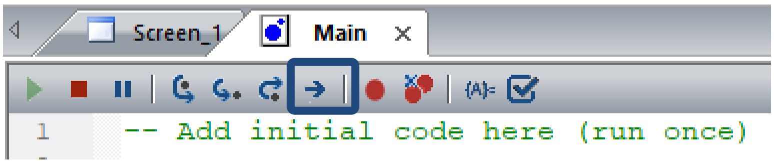
<table>
<tr><th colspan="2">1. Lua programming</th></tr>
<tr><td>Monitoring variable</td><td>
Step 1: enter “v1” in the monitoring variable window.



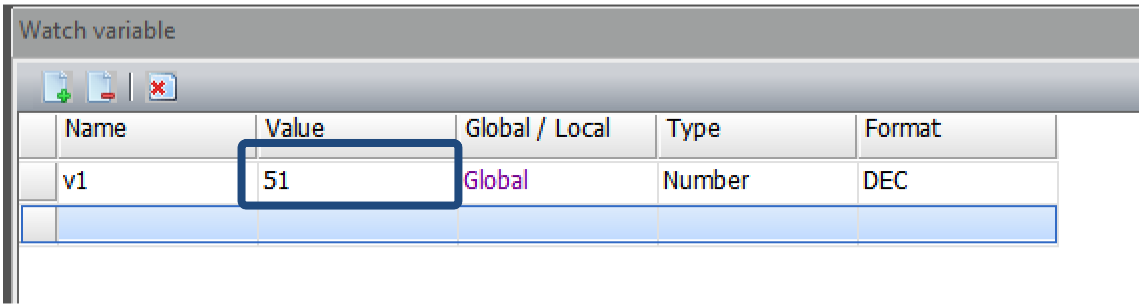
Step 2: set 50 for $100 on the HMI.



Step 3: click on the arrow button to carry on the execution in the Lua program editing window.



Step 4: Lua program runs and then stops again at line 13, where the breakpoint is placed. Because of the instructions of line 9 and 11, “v1” reads the value of $100 and plus 1, then “v1” in the variable monitoring window is 51.



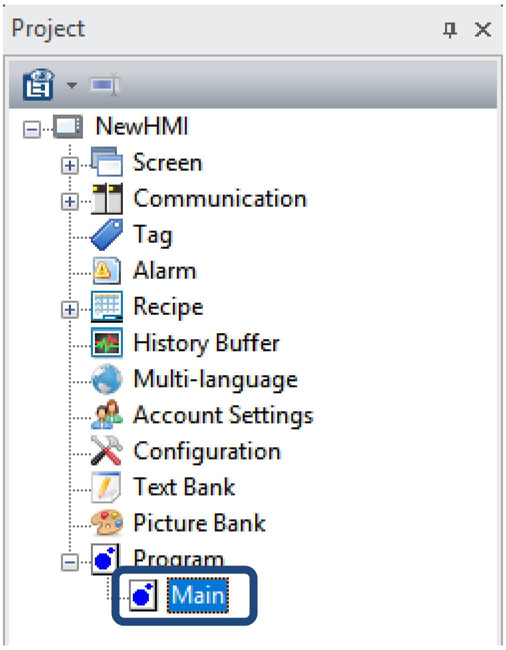
</td></tr>
</table>

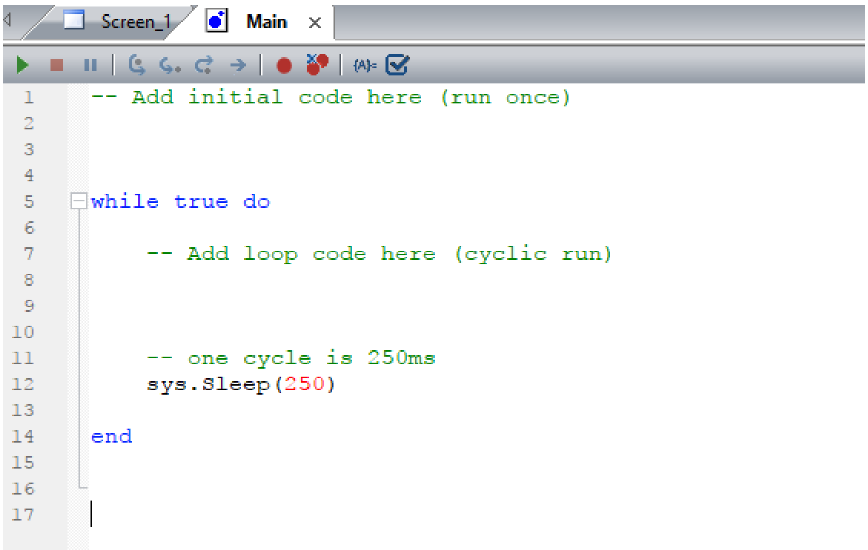
1.3 Subroutine execution

You can use subroutines in the Lua program to simplify the program for easy maintenance and development. In the Main program, you can use a “require” instruction to load the subroutine. Please refer to Table 1.4 Subroutine execution example.

Table 1.4 Subroutine execution example

<table>
<tr><th colspan="2">1. Lua programming</th></tr>
<tr><td>Edit [Main] program</td><td>Step 1: double-click [Main] in the DOPSoft project tree to enter the editing interface.





</td></tr>
</table>

1. Lua programming

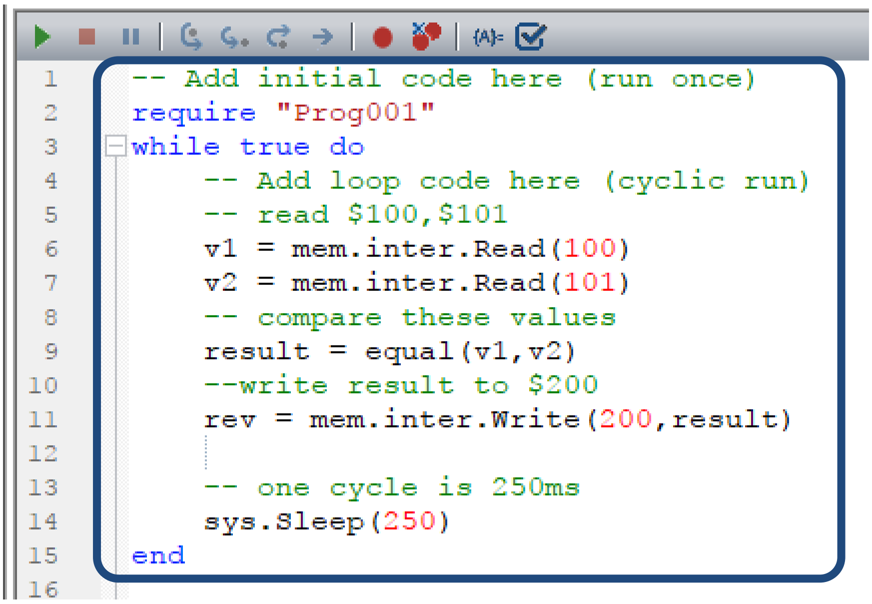
Edit [Main] program

Step 2: add the following string to the [Main] program.

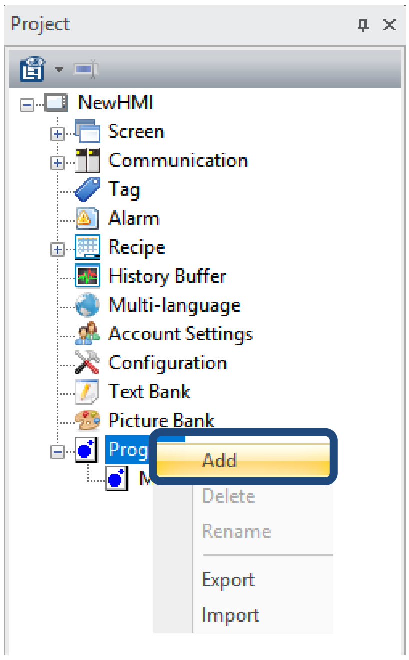
This part of the program compares the values of $100 and $101, and then write the result to $200. If these two values are equal, the result is 1; if not, the result is 0.

[require "Prog001"] in the second line loads Prog001. The instruction in line 9 calls the function in the subroutine.

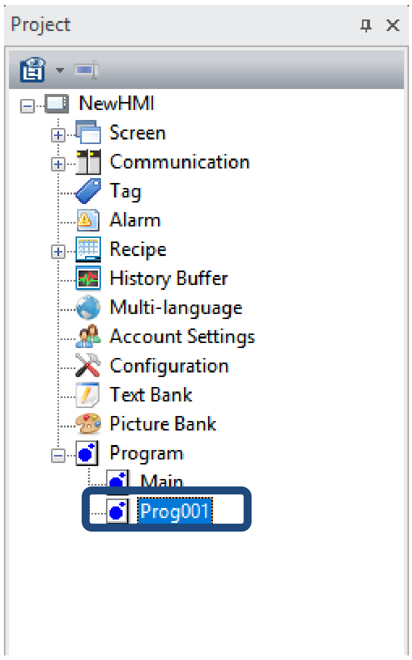
```
    require "Prog001"
while true do
    -- Add loop code here (cyclic run)
    -- read $100,$101
    v1 = mem.inter.Read(100)
    v2 = mem.inter.Read(101)
    -- compare these values
    result = equal(v1,v2)
    --write result to $200
    rev = mem.inter.Write(200,result)

    -- one cycle is 250ms
    sys.Sleep(250)
end
```



<table>
<tr><th colspan="2">1. Lua programming</th></tr>
<tr><td>Edit sub-program</td><td>
Step 1: right-click [Program] in the DOPSoft project tree to add a new program.



Step 2: double-click Prog001 to enter the editing interface.



</td></tr>
</table>

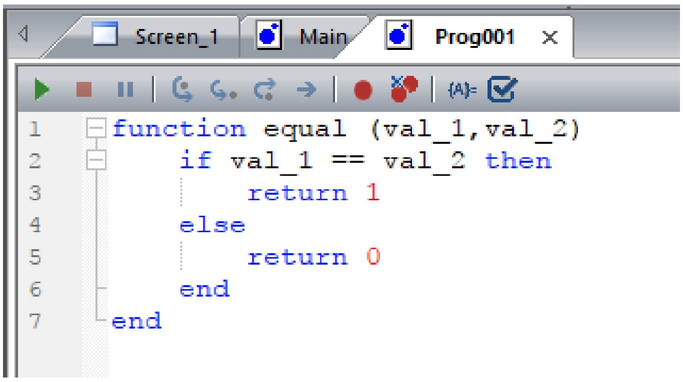
1. Lua programming

Edit sub-program

Step 3: add the string below to Prog001.

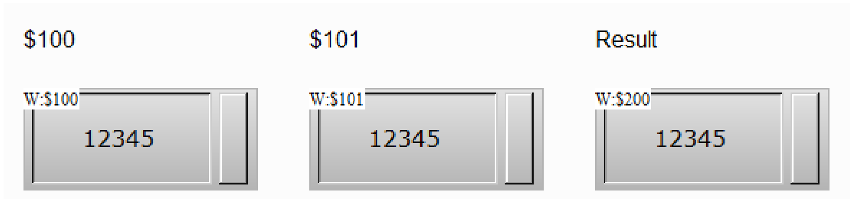
This program is a function and compares the two values to check if they are equal. If these two values are equal, the result is 1; if not, the result is 0.

```
    function equal (val_1,val_2)
    if val_1 == val_2 then
        return 1
    else
        return 0
    end
end
```



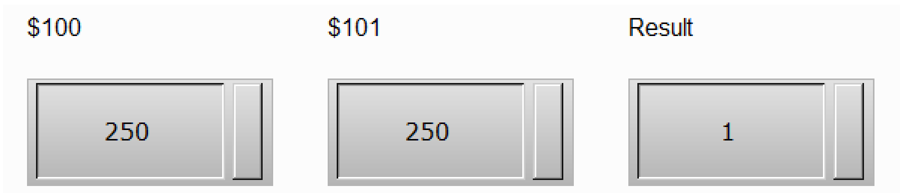
Edit HMI screen

Go to Screen_1 to create three Numeric Entry elements with the read addresses as $100, $101, and $200 respectively.



Execution results

Step 1: download the project to the HMI. Set 250 for both $100 and $101, and the HMI displays the comparison result as 1 as shown below:



Step 2: change the value of $100 to 200, and the HMI displays the comparison result as 0 as shown below: