LMI TECHNOLOGIES

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

**Title**: Gocator 4.x Scripting
**Revision:** 1.2

## Table of Contents

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

# 1 Introduction

This application note demonstrates how to create Gocator scripts through a series of examples. A script measurement can be used to program a custom measurement using a simplified C-based syntax. Similar to other measurement tools, a script measurement can produce measurement values and measurement decisions.

*The script examples in this document are also available in text files in the Zip archive containing this document. Copying from the text files and pasting into the Gocator script editor will preserve the formatting and spacing of the code.*

Outputs from one or several measurement tools can be used as inputs to the script, but the 3D profile data itself is not available in the script. The interpreter executing the script on the Gocator's CPU is too slow to process large 3D point clouds. This type of heavy data processing is better suited on the PC, using either the Gocator SDK or a supported 3rd party vision packages (e.g. HexSight, LabVIEW, Halcon, Common Vision Blox).

# 2 Script Development

## 2.1 Typical Use

A typical script would take results from measurement tools and perform some simple calculations, e.g. logical combinations of several measurement decisions or an average across several measurement tool values. However, it's also possible to create more advanced solutions. The script has access to all stamp information associated with the capture of each profile, which includes the time of capture and the encoder position. The script also supports storing intermediate results in persistent memory. This means that the user can implement a script that counts results over many frames, e.g. to accumulate volume, or a script that has state, e.g. to automatically detect objects under the sensor and measure the length of the object (in the direction of travel).

## 2.2 Language

The Gocator script supports the basic elements of the C language, but pointers are not supported. *Please refer to the Gocator User's Manual for up-to-date documentation of which C elements are supported, as well as all built-in functions.* This information is subject to change in future firmware releases and is therefore not copied into this document.

## 2.3 Syntax Checking

The Gocator scripting engine does not have any syntax checking. A syntax error in the script will show as "Invalid" when the sensor is running.

*Because of the lack of syntax checking it is strongly recommended to develop the script incrementally, with very small blocks of code added each time. For each iteration, test that the script runs as intended before moving on to add more code again*.

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

## 2.4 Reserved Names

The following names are reserved and cannot be used for variable names by the user in the script. Using these variable names will cause a script error.

- “measurement”
- “value”
- “decision”
- “exists”

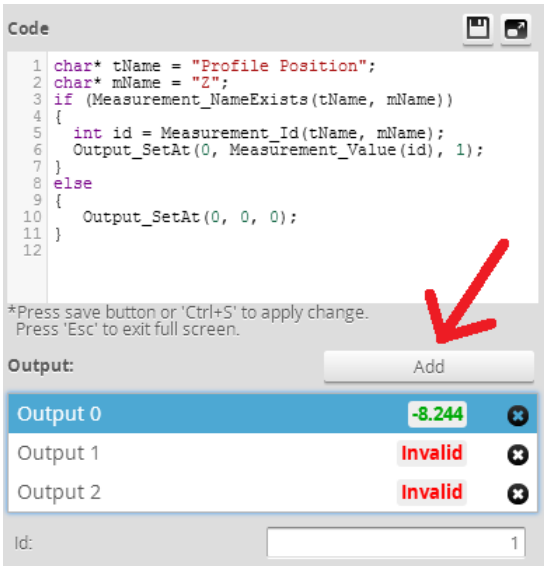
## 2.5 Output of multiple results

Gocator supports multiple script outputs. Add script outputs with the Add button. For each script output that is added, an Output will be added to the list and a unique ID will be generated.

To send a value and decision on a specific script output, use the function:

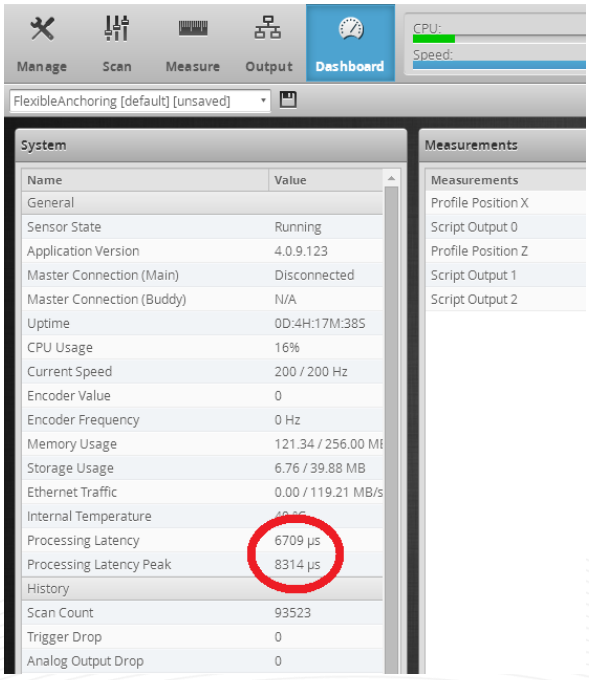
Output_SetAt(unsigned int index, double value, int decision), where:

- index – script output index
- value – value output by the script
- decision – decision value output by the script. Can be 1 for Pass, 0 for Fail, -1 for Invalid Value, -2 for Invalid Anchor




## 2.6 CPU load and processing latency

The script engine in the Gocator has limited processing capability. In case of any larger scripts, the user should take care to ensure that CPU load and processing latency is acceptable for the application. The CPU load is displayed prominently at the top of the Gocator web interface and the processing latency is displayed in the Dashboard as the “Output Latency”.



+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

Below is a table showing the processing latency of some basic operations on integer numbers:

| Basic Operation (Integer Numbers) | Approx. Processing Latency |
|---|---|
| Arithmetic Operators (+, -, *, or /) | 10us |
| Square Root | 85us |
| Exponential | 90us |
| Trigonometry (tan/cos/sin) | 90 to 100us |

**Table 1: Processing Latency of Basic Operations on Integer Numbers.**

In addition, to give the user an idea of the processing impact of a script, each script example below includes a table summarizing the processing load. This data was collected on a Gocator 2330 with factory default settings.

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

# 3 Example #1 – Converting from millimeters to inches

This example is a simple unit conversion, demonstrating the most commonly used built-in functions and basic concepts.

**NOTE**: The example is available in a text file in the Zip archive containing this document.

- The result from a measurement tool is accessed in the script through the tool's unique ID. This ID is displayed in the upper right corner of the tool's panel in the web UI.


- It is recommended to always check first that the measurement input to the script is valid, using the function Measurement_Valid, otherwise the script may operate on undefined values and produce unexpected results.
- That the decision limits for the converted output is given in thousands of an inch, highlights that the input to the script from the measurement tool is scaled by a factor of 1000.
- The built-in constant INVALID_VALUE is sent on the output if the measurement tool does not exist. Inside the script and on the native Gocator Ethernet protocol, this constant is set as the minimum 64-bit signed value. For all PLCs protocols, this constant is sent as the minimum 32-bit signed value.
- Processing Latency cost of this script is approximately 200 µs.

```
//Decision limits in inches
double DECISION_MIN = 0.6;
double DECISION_MAX = 0.7;

//Check that a measurement tool with ID 0 has been configured and is reporting a valid
result
if (Measurement_Exists(0) && Measurement_Valid(0))
{
  //Get value from measurement tool with ID 0
  double mm_value = Measurement_Value(0);

  //Convert to inches
  double inch_value = mm_value / 25.4;

  //Check decision limits
  if (inch_value > DECISION_MIN && inch_value < DECISION_MAX)
  {
    //Send value out and set decision to 1 for PASS decision
    Output_Set(inch_value, 1);
  }
  else
  {
    //Send value out and set decision to 0 for FAIL decision
    Output_Set(inch_value, 0);
  }
}
else
{
```

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

```
  //No valid data to process, send INVALID out from the script
  Output_Set(INVALID_VALUE, 0);
}
```

# 4 Example #2 – Logical combination of measurement decisions

This example is showing how to perform a logical combination of decisions from several measurement tools.

**NOTE**: The example is available in a text file in the Zip archive containing this document.

- It is strongly recommended to first check if the inputs to the script are valid and use the INVALID_VALUE output to indicate if that is not the case. Otherwise the logic combination may produce unexpected results and the device receiving the results from the Gocator will not know about it.
- Logical AND is represented by the **&&** operator.
- Logical OR is represented by the **||** operator.
- Logical NOT is represented by the **!** operator.
- Processing Latency cost of this script is approximately 190 µs.

```
//Check that the measurement tools are reporting a valid result
if (Measurement_Valid(0) && Measurement_Valid(1) && Measurement_Valid(2))
{
  //Get decision from measurement tool with ID 0
  int decision0 = Measurement_Decision(0);

  //Get decision from measurement tool with ID 1
  int decision1 = Measurement_Decision(1);

  //Get decision from measurement tool with ID 2
  int decision2 = Measurement_Decision(2);

  //Logic using operators && = AND, || = OR, ! = NOT
  int logic_combination = (decision0 && !decision1) || decision2;

  //Send combined decision out
  Output_Set(0, logic_combination);
}
else
{
  //No valid data to process, send INVALID out from the script
  Output_Set(INVALID_VALUE, 0);
}
```

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

# 5 Example #3 – Trigonometry and square root

This is an example showing how to use the built-in trigonometry functions, as well as the square root. The Law of cosines is used to calculate the length of the third side of a triangle, where an Intersect Angle tool is used to measure the angle and two Distance tools are used to measure the lengths of the two enclosing sides of the triangle.

**NOTE**: The example is available in a text file in the Zip archive containing this document.

- The Gocator script allows the user to create helper functions in order to simplify the script and avoid multiple instances of identical code, exemplified here by the angle2radians function.
- The trigonometry in the script works in radians. Before calling the trigonometry functions, the user must first convert the angle value produced by the tool into a radians value, before calling the trigonometry functions.
- Processing Latency cost of this script is approximately 670 µs.

```
//Constants
double PI = 3.14159265;

//Helper function -------
double angle2radians(double angle_degrees)
{
  //Convert from degrees to radians
  double angle_radians = angle_degrees * PI / 180;

  return angle_radians;
}

//Main program ------
//Check that the measurement tools are reporting a valid result
if (Measurement_Valid(0) && Measurement_Valid(1) && Measurement_Valid(2))
{
  //Get angle from Intersect Angle tool with ID 0
  double angle  = Measurement_Value(0);

  //Get length of one side of triangle from Distance tool with ID 1
  double a  = Measurement_Value(1);

  //Get length of other side of triangle from Distance tool with ID 2
  double b  = Measurement_Value(2);

  //Convert angle to radians
  double angle_radians = angle2radians(angle);

  //Calculate third side of triangle using the Law of cosines
  double c = sqrt(a*a + b*b - 2*a*b*cos(angle_radians));

  //Send output
  Output_Set(c, 0);
}
else
{
  //No valid data to process, send INVALID out from the script
  Output_Set(INVALID_VALUE, 0);
}
```

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

# 6 Example #4 – Persistent memory and multiple outputs

This is an example showing how to use persistent memory to track the minimum and maximum value of a tool over time, and then how to send the two values out using Output_SetAt().

**NOTE**: The example is available in a text file in the Zip archive containing this document.

- Using constants with suitable names is recommended when working with several memory locations. This will make the code easy to read and help avoid mistakes reading data from the wrong location.
- The first time the script is called after the sensor is started the frame count is always zero. Calling the built-in function Stamp_Frame() is a good way to initialize memory in the first frame.
- Processing Latency cost of this script is approximately 250 µs.

```
//Constants
int INDEX_MIN = 0;
int INDEX_MAX = 1;

//Check that a measurement tool with ID 0 has been configured and is reporting a valid
result
if (Measurement_Exists(0) && Measurement_Valid(0))
{
  //Get value from measurement tool with ID 0
  double v = Measurement_Value(0);

  if (Stamp_Frame() == 0)
  {
    //Initialize min and max in memory
    Memory_Set64f(INDEX_MIN, v);
    Memory_Set64f(INDEX_MAX, v);
  }

  //Check if current value is smaller than min
  if (v < Memory_Get64f(INDEX_MIN))
  {
    Memory_Set64f(INDEX_MIN, v);
  }

  //Check if current value is larger than max
  if (v > Memory_Get64f(INDEX_MAX))
  {
    Memory_Set64f(INDEX_MAX, v);
  }

  //Get updated min and max from memory
  double min = Memory_Get64f(INDEX_MIN);
  double max = Memory_Get64f(INDEX_MAX);

  //Send min value on output index 0
  Output_SetAt(0, min, 1);

  //Send max value on output index 1
  Output_SetAt(1, max, 1);
}
else
{
  //No valid data to process, send INVALID out from the script
```

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

```
  Output_Set(INVALID_VALUE, 0);
}
```

# 7 Example #5 – Detecting object and measuring length

This is an example of a script with state, which detects the leading and trailing edge of an object, to calculate the length of the object using the encoder stamps.

**NOTE**: The example is available in a text file in the Zip archive containing this document.

- In this example, the valid flag of the measurement tool is used for object detection. In other words, when there are no data points within the tool's region(s), the script considers it as no object present under the sensor. An alternative approach could for example be to use a threshold on a Height tool to trigger the presence of an object.
- This technique can also be used to store various measurements in memory of an object as it is scanned. The script could then calculate object level properties when the trailing edge is detected. In such a scenario the user needs to monitor the processing time of the script when the "heavy" processing is executed at the end of the object. Because of the slow execution speed of the script, the time to process an array of only a few hundred elements could be several seconds.
- Processing Latency cost of this script is approximately 340 µs.

```
//Constants
int INDEX_SCAN_STATE = 0;
int INDEX_ENCODER = 1;

long long STATE_WAITING = 0;
long long STATE_SCANNING = 1;

//Encoder resolution in mm/tick
float ENCODER_RESOLUTION = 0.03

if (Stamp_Frame() == 0)
{
  //Initialize state to waiting
  Memory_Set64s(INDEX_SCAN_STATE, STATE_WAITING);
}

//Get current state
long long scan_state = Memory_Get64s(INDEX_SCAN_STATE);

//Get measurement value from measurement tool with ID 0
double v = Measurement_Value(0);

//Check state
if (scan_state == STATE_WAITING)
{
  //Use valid flag for object detection
  if (!Measurement_Valid(0))
  {
    //Still waiting for object...
    Output_Set(0, 0);
  }
  else
  {
    //Object detected!
```

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada

```
    //Change state
    Memory_Set64s(INDEX_SCAN_STATE, STATE_SCANNING);

    //Store encoder count for leading edge of object in memory
    Memory_Set64s(INDEX_ENCODER, Stamp_Encoder());

    Output_Set(0, 1);
  }
}
else if (scan_state == STATE_SCANNING)
{
  if (Measurement_Valid(0))
  {
    //Still scanning...

    Output_Set(0, 1);
  }
  else
  {
    //Object end detected!

    //Change state
    Memory_Set64s(INDEX_SCAN_STATE, STATE_WAITING);

    //Get encoder count at trailing edge of object
    long long endEnc = Stamp_Encoder();

    //Recall encoder count at leading edge from memory
    long long startEnc = Memory_Get64s(INDEX_ENCODER);

    //Calculate object length in mm
    double length = ENCODER_RESOLUTION * (endEnc - startEnc);

    Output_Set(length, 0);
  }
}
else
{
  //This should never happen
  Output_Set(INVALID_VALUE, 0);
}
```

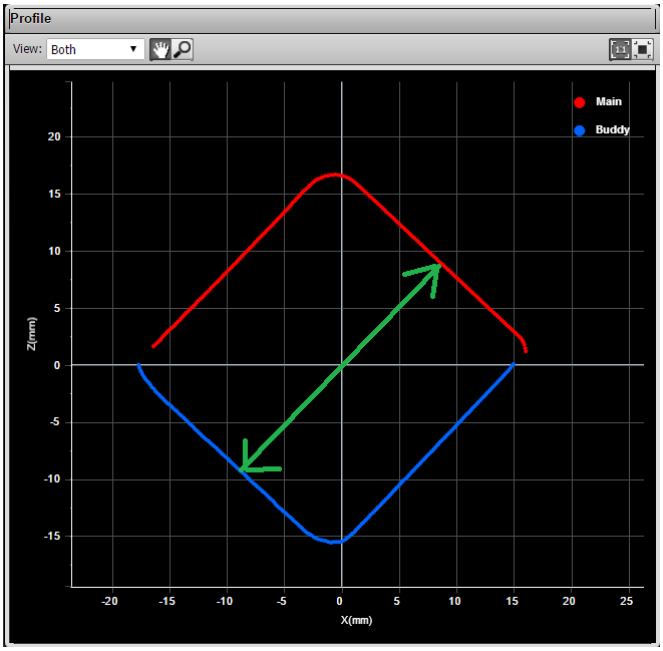
## 8 Example #6 – Calculate the distance between 2 parallel lines

**NOTE**: The example is available in a text file in the Zip archive containing this document.

If line 1 is described as z1 = rc * x1 + b1 and line 2 as z2 = rc * x2 + b2:
Then the distance between the 2 lines is |b2 – b1| / sqrt(rc * rc + 1)

In the Gocator opposite setup, the profiles look like:

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
1673 Cliveden Avenue
Delta BC V3M 6V5
Canada




Gocator tools setup:

Both the top and bottom profile need an “Profile Intersect Angle” with reference to X-axis, “Profile Position X” and “Profile Position Z” tool to describe the line.

```
// This script calculates the distance between 2 parallel lines.
// line1: Z1 = rc*X1 + b1
// line2: Z2 = rc*X2 + b2
// distance between 2 parallel lines is |(b2 - b1)| / sqrt(rc * rc +1)
long long a1 = Measurement_Value(0); // angle 1 from tool
long long a2 = Measurement_Value(3); // angle 2 from tool
long long x1 = Measurement_Value(1); // x1 from tool
long long x2 = Measurement_Value(4); // x2 from tool
long long z1 = Measurement_Value(2); // z1 from tool
long long z2 = Measurement_Value(5); // z2 from tool
float pi = 3.141592654;
float ang1e1 = pi / 180 * a1 / 1000; // angle 1 in radians
float angle2 = pi / 180 * a2 / 1000; // angle 2 in radians
float rc1 = tan(ang1e1);
float rc2 = tan(angle2);
float rc = (rc1 + rc2) / 2; // rc is average angle (assume 2 lines are approximately parallel)
float X1 = x1 / 1000; //X1 in mm
float X2 = x2 / 1000; // X2 in mm
float Z1 = z1 / 1000; // Z1 in mm
float Z2 = z2 / 1000; // Z2 in mm
float b1 = Z1 - (rc * X1);
float b2 = Z2 - (rc * X2);
float d = fabs(b1 - b2) / sqrt(rc * rc + 1);
Output_SetAt(0, d * 1000, 0); // d in um
```