Version: 4.0.1

Introduction to Scripting

Behavior Tree 4.X introduces a simple but powerful new concept: a scripting language with XML.

The implemented scripting language has a familiar syntax; it allows the user to quickly read from / write to the variables of the blackboard.

The simpler way to learn how scripting works is using the built-in action **Script**, which was introduced in the second tutorial

Assignment operators, strings and numbers

Example:

```
param_A := 42
param_B = 3.14
message = 'hello world'
```

- The first line assigns the number 42 to the blackboard entry **param_A**.
- The second line assigns the number 3.14 to the blackboard entry **param_B**.
- The third line assigns the string "hello world" to the blackboard entry **message**.

> TIP
>
> The difference between the operator **":="** and **"="** is that the former may create a new entry in the blackboard, if it doesn't exist, whilst the latter will throw an exception if the blackboard doesn't contain the entry.

You can also use **semicolons** to add multiple commands in a single script.

```
A:= 42; B:=24
```

Arithmetic operators and parenthesis

Example:

```
param_A := 7
param_B := 5
param_B *= 2
param_C := (param_A * 3) + param_B
```

The resulting values of `param_B` is 10 and `param_C` is 31.

The following operators are supported:

Operator	Assign Operator	Description
+	+=	Add
-	-=	Subtract
*	*=	Multiply
/	/=	Divide

Note that the addition operator is the only one that also works with string (used to concatenate two strings).

Bitwise operator and hexadecimal numbers

These operators work only if the value can be cast to an integer number.

Using them with a string or real number will cause an exception.

Example:

```
value:= 0x7F
val_A:= value & 0x0F
val_B:= value | 0xF0
```

The value of `val_A` is 0x0F (or 15); `val_B` is 0xFF (or 255).

Binary Operators	Description
|	Bitwise or
&	Bitwise and
^	Bitwise xor

Unary Operators	Description
~	Negate

Logic and comparison operators

Operators which return a boolean.

Example:

```
val_A := true
val_B := 5 > 3
val_C := (val_A == val_B)
val_D := (val_A && val_B) || !val_C
```

Operators	Description
true/false	Booleans. Castable to 1 and 0 respectively
&&	Logic and
||	Logic or

Operators	Description
!	Negation
==	Equality
!=	Inequality
<	Less
<=	Less equal
>	Greater
>=	Greater equal

Ternary operator if-then-else

Example:

```
val_B = (val_A > 1) ? 42 : 24
```

C++ example

Demonstration of the scripting language, including how to use enums to represent **integer values**.

The XML:

```
<root >
    <BehaviorTree>
        <Sequence>
            <Script code=" msg:='hello world' " />
            <Script code=" A:=THE_ANSWER; B:=3.14; color:=RED " />
            <Precondition if="A>B && color!=BLUE" else="FAILURE">
                <Sequence>
                  <SaySomething message="{A}"/>
                  <SaySomething message="{B}"/>
```

```
                <SaySomething message="{msg}"/>
                <SaySomething message="{color}"/>
              </Sequence>
            </Precondition>
          </Sequence>
      </BehaviorTree>
</root>
```

The C++ code to register the Nodes and the enums:

```
int main()
{
  // Simple tree: a sequence of two asynchronous actions,
  // but the second will be halted because of the timeout.

  BehaviorTreeFactory factory;
  factory.registerNodeType<SaySomething>("SaySomething");

  enum Color { RED=1, BLUE=2, GREEN=3 };
  // We can add these enums to the scripting language
  factory.registerScriptingEnums<Color>();

  // Or we can do it manually
  factory.registerScriptingEnum("THE_ANSWER", 42);

  auto tree = factory.createTreeFromText(xml_text);
  tree.tickWhileRunning();
  return 0;
}
```

Expected output:

```
Robot says: 42.000000
Robot says: 3.140000
Robot says: hello world
Robot says: 1.000000
```

Note as, under the hood, an ENUM is always interpreted as its numerical value.

Edit this page