CANSwitch

Ladder Manual

Revision H
26 December 2017

Important information

This manual assumes you have already read the CANSwitch User Manual. If not, please familiarise yourself with the user manual before continuing to read this manual.

The newest firmware, PC utilities, user manuals, installation diagrams, etc is available from www.canswitch.co.za

Copyright

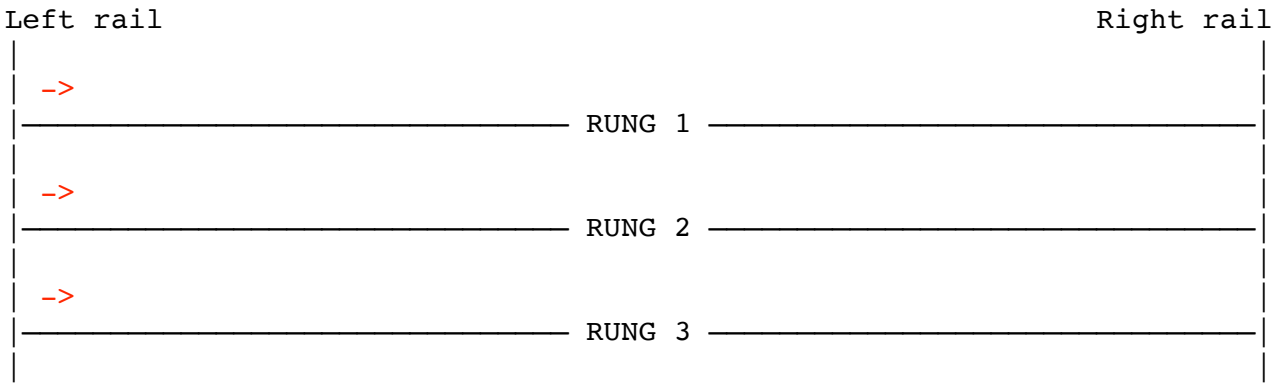
Introduction

CANSwitch controls your accessories by following rules that define how the accessories should be controlled. For example, a rule could say “Set the spotlights to 100% brightness when the high beam is switched on.” These rules are created in the form of ladder logic diagrams. The ladder editor program allows you to create and edit these diagrams.

Before using the ladder editor, it is essential to understand how ladder logic works. This manual introduces you to ladder logic and gives examples of its use.

Ladder Logic

Ladder logic is really a very simplistic way of connecting inputs (e.g. switches) to outputs (e.g. lights). Ladder logic is so called because the diagrams look like a ladder, with vertical rails on the sides, and rungs that span horizontally between them. There are one or more rungs spanning between the two rails. The rungs represent the list of rules you want CANSwitch to follow.

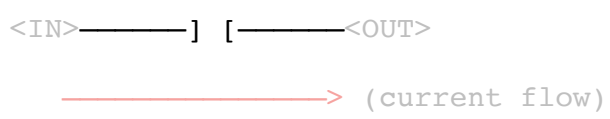


The ladder above has 3 rungs. In a ladder diagram, current flows from the left rail, across each rung, to the right hand rail, as indicated by the arrows. Various tokens can be placed on each rung, controlling whether or not the current can continue flowing to the right. A ladder diagram can have many rungs, and each rung many tokens. CANSwitch evaluates each rung in turn, starting at the top. In the diagram above, rung 1 would be evaluated first, followed by rung 2 and then rung 3. One cycle of evaluating all the rungs is called a scan. Once a scan is complete the process starts over again.

CANSwitch performs a scan every 5mS (milliseconds), so it evaluates the complete ladder 200 times every second.

Tokens

Items that you place on the rung are called tokens. A token is something that can make a decision, or perform an action, and are the building blocks of the ladder diagram. A token has an <IN> end and an <OUT> end.

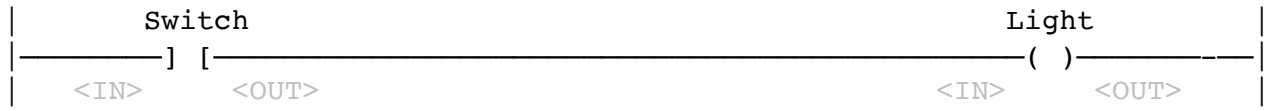


Current flows through the token from the <IN> end to the <OUT> end. The token can control whether the current continues flowing or not. For example, a token that represents a switch can prevent the current from flowing if the switch is off.

Tokens can follow each other on a rung, such that its <IN> end is connected to the previous token's <OUT> end, and its own <OUT> is connected to the next token's <IN>.

If current flows into the token, it's <IN> is said to be on. If the token allows current to continue flowing, its <OUT> is said to be on. If the token does not allow the current to continue flowing then its <OUT> is off.

On a rung this would be as follows:



Although indicated here, the <IN> and <OUT> are not drawn in the actual ladder diagrams.

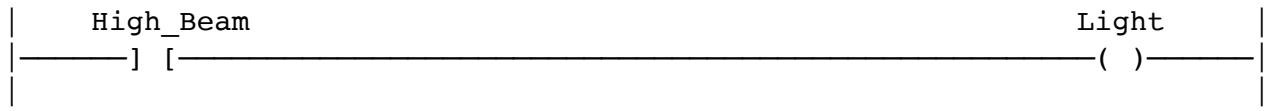
Accessories

Items that you install on your motorcycle (and control with CANSwitch) are called accessories. An example of this would be additional spotlights, a remote transmitter, an extra brake light, etc. Accessories are represented in your ladder diagram as outputs, as CANSwitch's outputs will provide power to your accessories.

Switches, coils and lights

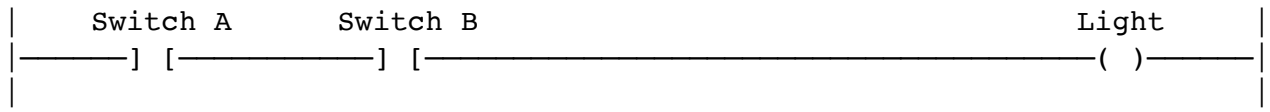
Tokens can represent the switches on your handlebar controls, like the high beam switch. The token is placed on a rung, and your high beam switch is then represented by that token in the ladder diagram.

The switch can then control a light that is placed on the same rung:



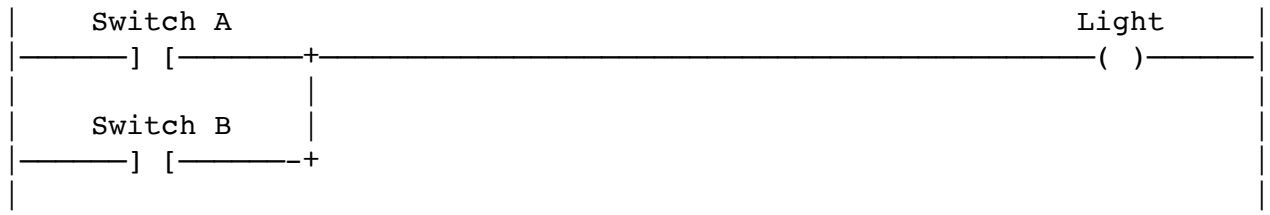
If the switch is off, the current will not continue flowing. If the switch is on, the current will continue to flow, making the light turn on.

It is as easy as that. Now you can add more switches:



In this rung, both switch A and switch B must be on for the light to illuminate. If either of the switches are off, the current will not reach the light, and the light remains off. This kind of switch arrangement is called an *AND* configuration, as both A *and* B must be on in order to illuminate the light.

An *OR* arrangement is also possible:



If either switch A, or switch B is on then the light will illuminate. Unlike the *AND* configuration, only one of the switches needs to be on for the light to illuminate. Both could also be on, but it would not make any difference if only one or both switches are on.

Switches are placed to the left of the lights on the rung. The light is just a representation of an accessory that is connected to one of CANSwitch's outputs, for example a spotlight. In ladder logic the term "coil" is used when referring to outputs.

Switches are indicated by the —] [— symbol. Coils are indicated by the —( )— symbol.

Switch types

There are two types of switches. A normal switch, and an inverted switch. A normal switch allows the current to flow if the switch is on. An inverted switch allows the current to flow if the switch is off. Their tokens looks as follows:

```
<IN>———] [———<OUT>  Normal Switch
<IN>———]/[———<OUT>  Inverted Switch
```

Substituting an inverted switch in our previous example:

```
|    Switch                                                    Light     |
|———————]/[———————————————————————————————————————————————( )———————|
|                                                                        |
```

This will cause the light to illuminate when the switch is turned off. You might wonder why you would want something to turn on, when you turn the switch off? Sometimes, monitoring a switch for an *OFF* condition is handy, for example if you want something to happen when the ignition turns off, instead of when it turns on.

Using your motorcycle's existing handlebar controls

Most of your motorcycle's handlebar controls are detected by CANSwitch. It does this by analysing the CAN bus data of the motorcycle to see which handlebar switches are on or off, or which buttons are pressed. The handlebar switches can then be used in a ladder diagram. For example, a switch in the ladder diagram can be associated with the horn button, so that when the horn is activated the switch is on.

```
|     Horn                                                     Light     |
|———————] [———————————————————————————————————————————————( )———————|
|                                                                        |
```

The rung above uses the horn as a switch. When the horn activates the switch is on. To use one of your motorcycle's handlebar controls in the ladder diagram, simply write its name above the switch symbol.

Coils

A coil is used when the ladder diagram must turn something on or off, or adjust its brightness, or simply store a value. There are also different types of coils:

<IN>——( )——<OUT> Normal coil.
This coil is turned on when its <IN> is on, and turned off when its <IN> is off (in real terms it means that the accessory associated with this coil is turned on or off).

<IN>——(S)——<OUT> Set coil.
This coil turns on when its <IN> is on, but does not turn off when its <IN> turns off. It is a latching coil, once it has been turned on it will stay on (unless the same coil is turned off elsewhere in the ladder diagram).

<IN>——(R)——<OUT> Reset coil.
Opposite to a *Set* coil. This coil turns off when its <IN> is on, and remains off until the same coil is turned on again elsewhere in the ladder diagram. Using a Set coil and Reset coil is a convenient way of turning something on under certain conditions, but turning it off under different conditions.

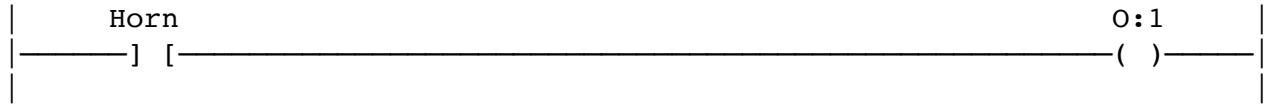
<IN>——(A)——<OUT> Analog coil.
This coil is given an analog value when its <IN> is on. *Analog* means it can have varying values, instead of simply being on or off. It could even be halfway on. This is useful for controlling the brightness of lights. For example, setting the Analog coil to 50%, and having it connected to a light, will make the light illuminate at half its brightness. The percentage value can be specified as a number between 0 and 1000, this represents a percentage from 0 to 100% in steps of 0.1%.

<IN>——(T)——<OUT> Toggle coil.
The coil is toggled (set to the opposite state) when its <IN> is on. If the coil was on it will turn off, and if it was off it will turn on. If its <IN> is off then it maintains its last state.

<IN>——(P)——<OUT> Pulsed coil.
If the <IN> is on then this coil is pulsed. You can specify how many times it is pulsed, how long each pulse is, and how long to wait before starting the next pulse. It will continue its pulsing operation for the count specified, even if <IN> turns off again. If, however, the <IN> is still on when the pulsing finishes, the pulsing is started again.

Using coils

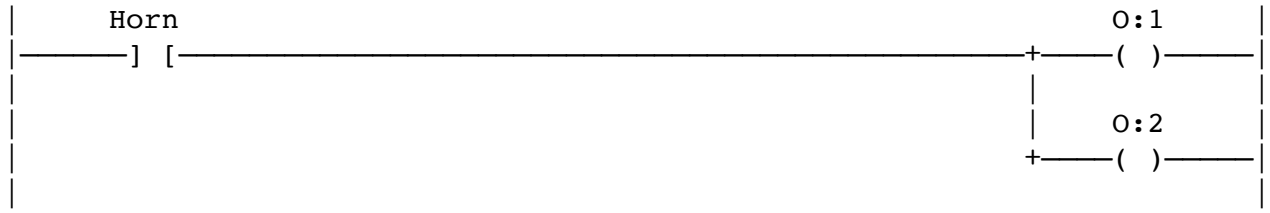
When you place a coil on a rung, you need to associate it with your accessory by telling CANSwitch which output the accessory is connected to. This is done by writing the output number above the coil.



The diagram above will make output #1 turn on when the horn is used. The accessory connected to output #1 will then turn on.

Coils do not always have to be associated with an accessory, though. A coil could also be associated with a memory (so you can store a value in the memory), or with a Master relay. These are explained in more detail later.

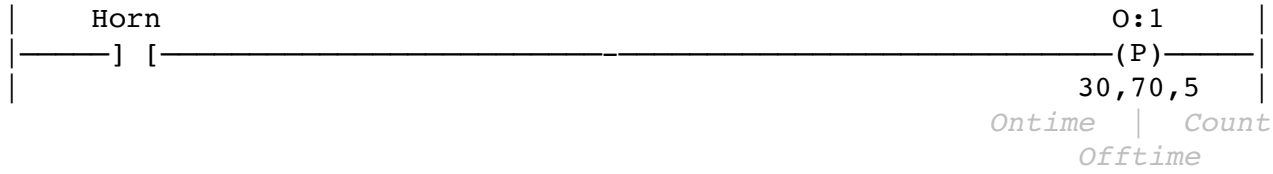
You can also control multiple coils simultaneously.



This rung will turn both output #1 and output #2 on when the Horn is operated.

Some coils need parameters. The parameters specify how the coil operates. The pulsed coil needs 3 parameters that tells it how to pulse the output. These parameters are called *Ontime*, *Offtime* and *Count.*

The *Ontime*, *Offtime* and *Count* values are written below the Pulsed Coil:

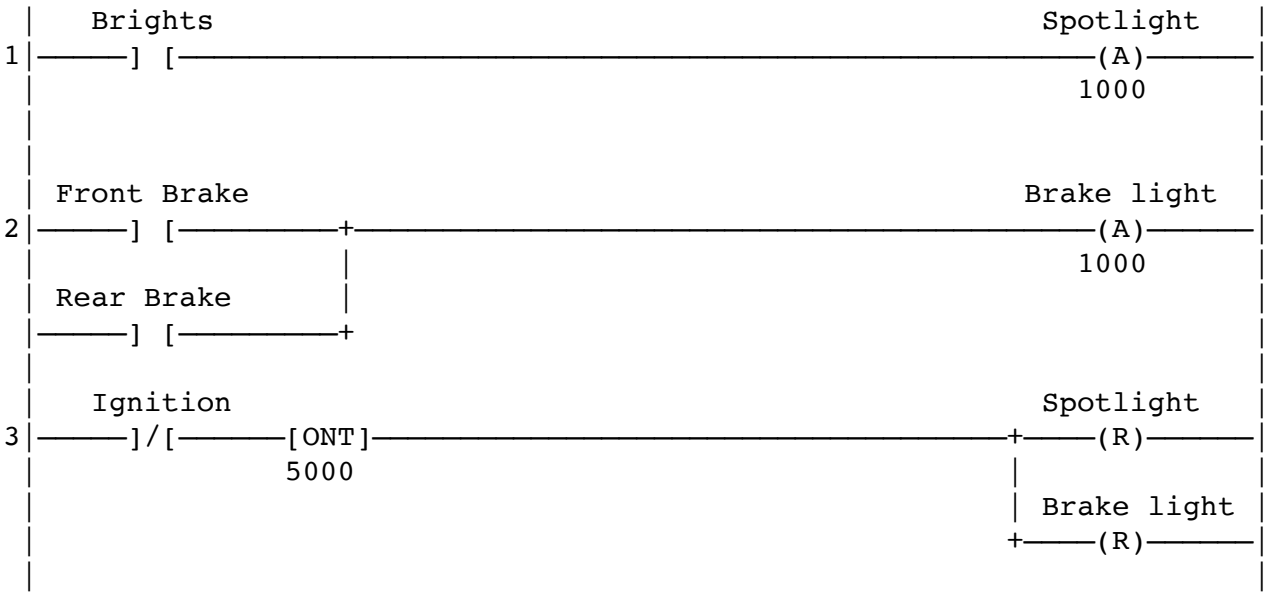


In this example, when the Horn is activated, output O:1 will be pulsed. Each pulse lasts for 30mS, and then CANSwitch waits 70mS before the next pulse. This is repeated 5 times.

The CANSwitch outputs are numbered O:1 to O:8 (Series 4) and O:1 to O:6 (Series 3). These correspond to the output wire colours:

O:1 - Brown
O:2 - Orange
O:3 - Yellow
O:4 - Green
O:5 - Blue
O:6 - Violet
O:7 - Grey
O:8 - White

Basic ladder example



In this example:
1. When the Brights (high beam) is on, the Spotlight is set to 100.0% brightness.
2. When the Front Brake or the Rear Brake is activated the Brake light is turned on to 100.0%.
3. If the Ignition is off (*notice the inverted switch*) for longer than 5000mS (5 seconds) then the Spotlight and Brake light are Reset, thus turned off.

Timers

This example also demonstrates the use of the —[ONT]— symbol, an On timer. The On timer is like a switch that only turns on after a set time period, 5000 ms in the example above. This has the effect of delaying the ignition OFF state for 5 seconds. So, 5 seconds after the ignition has turned off the Spotlight and Brake light will also turn off.

An Off timer (symbol —[OFT]—) does the opposite of an On timer. It is like a switch that stays ON until its input side (left side) has been OFF for the time specified.

Counters

With Counters, you can count how many times something happens. This is useful if you want an output to activate only after something has happened a number of times. For example, you can make the spotlights flash when you press high beam button twice.

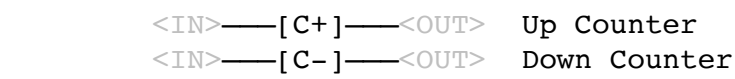
With a counter, its <IN> must turn on in order to make it count. It counts once when its <IN> turns on, and then the <IN> must turn off and back on to make it count again.

Counter Parameters

A counter has a *Step* parameter. This specifies by how much it counts. If *Step* is 1, it will count in steps of 1. The counter can count up or down. Counting down is achieved by using a negative *Step* value (e.g. -1).

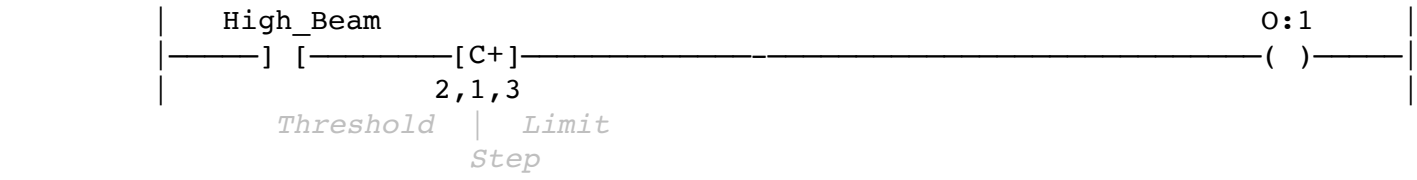
The counter's <OUT> will be on if the count reaches the *Threshold*. There is also a *Limit*, the counter will stop counting when it reaches this value.

The symbol for a counter is:



The symbol has a '+' to indicate an Up counter, or a '-' for a Down counter.

Specify the *Threshold*, *Step* and *Limit* parameters below the counter:



In this example, each time the high beam is turned on, 1 is added to the counter. When it reaches 2 then output #1 is turned *ON.* The count value will not exceed 3 (the *Limit*, 3 is an arbitrary value, as long as it is higher than the *Threshold*).

Time Counters

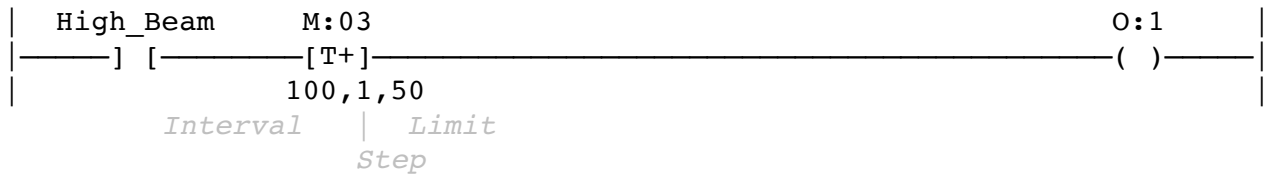
Time counters are similar to Up/Down counters. However, a Time counter continues counting as long as its <IN> remains on. You can specify how fast the Time counter must count. This is done with its *Interval* parameter. It can also count downwards by specifying a negative *Step* value.

The Time counter's <OUT> is always the same as its <IN>. The Time counter simply counts as long as <IN> is on.

```
<IN>———[T+]———<OUT>  Up Time Counter
<IN>———[T-]———<OUT>  Down Time Counter
```

The '+' and '-' symbol is representative of the Time counter counting up or down. The actual direction of counting is controlled by the *Step* parameter being a positive value or a negative value.

To use a Time Counter, specify the *Interval*, *Step* and *Limit* parameters below the counter:



Here, while the high beam is on, the memory *M:03* is incremented by 1 every 100ms, up to a maximum value of 50. The state of the coil on the right is not influenced by the Time counter, it is only determined by the high beam switch. The high beam switch simply flows through the Time counter, and the Time counter times how long the high beam is on (and stores that time in memory *M:03*).

The *M:03* is a memory, just like that of a calculator. Memories are discussed in more depth below.

Variables

At this point we need to take a detour from the ladder diagram and its tokens, and discuss variables. To understand ladder diagrams it is also necessary to understand the concept of variables.

A variable is so called because it is a value that can change (hence *variable*). Variables can have different types of values, CANSwitch uses these two types:

1. Boolean - its value can only be on or off (like a switch).
2. Number - its value can range from -32768 to 32767 (speed, RPM, engine temperature, etc).

CANSwitch reads various items from the motorcycle's CAN bus, and stores these values in CAN variables. These CAN variables are different for each model of motorcycle. To see which CAN variables are available for your motorcycle, open the ladder editor utility and select the CAN map relevant to your motorcycle. All the CAN variables listed for your motorcycle model are usable in your ladder diagrams.

Variables are grouped by their type. These are all the groups of variables:

Type	Prefix	Range	How Many
CAN	No Prefix	-32768 to 32767 or Boolean	Model dependant
Memories	M:	-32768 to 32767	64 (M:01 to M:64)
Physical outputs	O:	Boolean (on or off) or 0..100%	*8 (O:1 to O:8)
Physical Inputs	I:	Boolean (on or off)	2 (I:1 and I:2)
Master Relays	MR:	Boolean (on or off)	4 (MR:1 to MR:4)
User Variables	U:	-32768 to 32767	64 (uses M:01 to M:64)
System Variables	S:	Boolean (on or off)	5

** CANSwitch series 4 has 8 physical outputs, while series 3 has 6 physical outputs.*

The variables above are discussed in more detail in following chapters.

Memories

Memories are the same as the memory in a calculator, where you can store a value for later use. CANSwitch has 64 memories, named M:01 to M:64. You can also use your own names for these memories, to make it easier to remember.

HOW CAN MEMORIES BE USED?

- A rung can write a value to memory instead of to an output. The memory remembers the last written value (e.g. set M:01 to 200).
- A memory can be used to control an output (e.g. set output #1 to M:01's value).
- A memory can be used as part of a rule's condition (e.g. must be > 50)

WHAT PURPOSE DOES MEMORY SERVE?

Let's take a use case as an example. Let's say you have 3 basic rules:

1. When it is daylight, set the spotlights to 75%.
2. When it is night, set the spotlights to 25%.
3. When the high beam is on, set the spotlights to 100%.

Now assume it is daylight and you are riding, the lights are on 75% brightness. It seems a bit bright for conditions, so you use the navigator scroll wheel to dim them down to 50%. While going through a tunnel you turn the high beam on, and the spotlights go up to 100%. When you exit the tunnel and you turn the high beam off again, should the spotlights now go back to the 50% you adjusted them to? Or back to the 75% that rule #1 (above) specifies? In most cases you would want them to go back to the 50% you had them on before you turned on the high beam. However, this requires that CANSwitch remembers that you adjusted the light to 50% before switching the high beam on.

Memories help CANSwitch remember values temporarily. In order to create the above scenario with rules you would create the following rules:

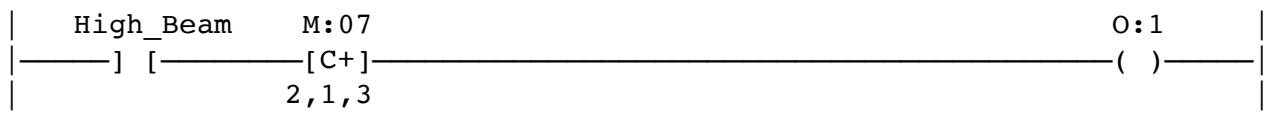
1. Set memory M:01, M:02 and M:03 default (startup) values to 75%, 25% and 100% respectively.
2. When it is daylight, set the spotlight to M:01.
3. When it is night, set the spotlights to M:02.
4. When the high beam is on, set the spotlight to M:03.
5. When the scroll wheel is turned up, increase M:01 by 10%.
6. When the scroll wheel is turned down, decrease M:01 by 10%.

As you can see, adjusting the scroll wheel now affects the value stored in M:01, irrespective of what the spotlight is doing. So, having the adjusted values stored in memory, you can simply make the spotlight use a memory value instead of setting it to a fixed brightness values. The memory always remembers the last values stored in it.

Apart from storing a value, or being used as the value for a physical output, memories can also be used in rule conditions. For example, you can say, "If memory M:01 is greater than 50 then switch on output 4", or "If M:01 < 5, pulse output 6", etc.

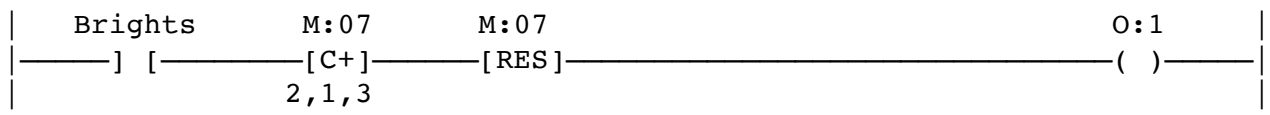
Using memories with Counters

When you use an Up or Down counter you can specify the memory where the count value must be stored, by writing it above the counter:



Here, the counter value is stored in memory M:07. This is helpful if you want to manipulate the count value somewhere else in the diagram, perhaps by making it count down under different conditions - then you could have an Up counter as well as a Down counter using the same memory.

Our example has one caveat, though. The counter's threshold is 2, so when the count reaches 2 the <OUT> will turn on. In order to restart counting from 0 we need to somehow clear the count value once it reaches 2. This can be done with a RESET (symbol —[RES]—). Let's add it to our rung:

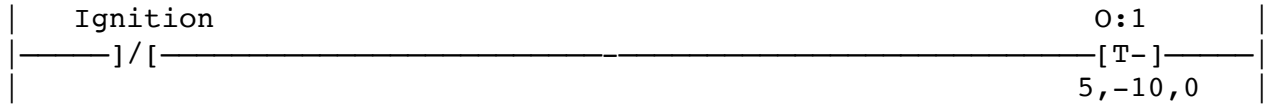


Now, when the count reaches 2, the counter's <OUT> is turned on, allowing the memory to be cleared (RESET to 0). This lets us start counting again when the high beam is activated again.

It is not strictly necessary to specify a memory when using a Counter. If you don't specify one then CANSwitch will automatically use an available memory to hold the count. But, we do not know which memory CANSwitch will use, so if you want to be able to manipulate the count value (e.g. the RESET above) then you must tell CANSwitch which memory to use.

Using memories with Time Counters

When you use a Time counter you must specify the memory where the count value must be stored, by writing it above the counter. You can also use one of CANSwitch's outputs instead of a memory, then the Time counter can be used to make your spotlights fade to off instead of turning off abruptly when the ignition turns off:



Remember an output can be set to a value between 0 and 1000 (0 to 100%)? Well, this trick allows us to make the value of output #1 count down (i.e. go dimmer) when the ignition turns off. This timer will go down by 1% every 5ms, until it reaches 0. This gives us a nice fade-out effect when the spotlights need to turn off.

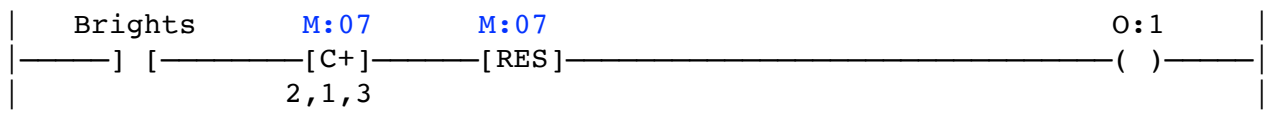
User Variables

You can create your own variables, and use these in your diagram any place you would normally use a memory variable.

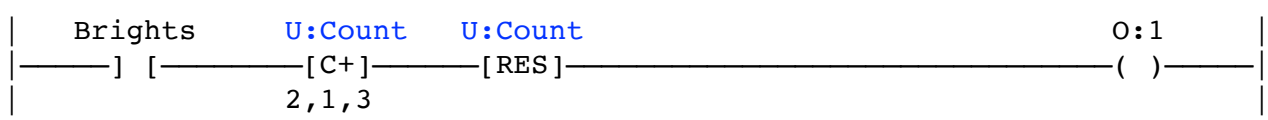
When you create a user variable its name is prepended with a “U:”, to distinguish it from other variables (e.g. CAN variables).

User variables are stored in one of the 64 memories. CANSwitch will automatically select an unused memory to store your variable in.

The previous example of using a memory with a counter can be changed to use a user variable instead, as follows:



M:07 can be replaced with a user variable:



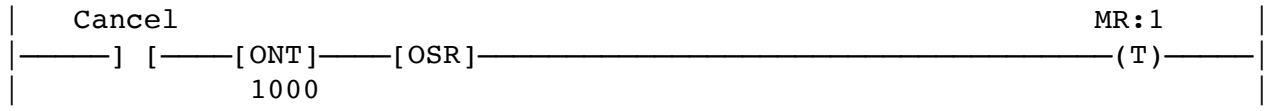
Here the user variable named “Count” is used instead of a memory.

Master Relays

CANSwitch has 4 Master Relays. A Master Relay is like a main switch, similar to the safety trip switch in your house's electrical box. If the switch is off, then no matter what other light switch you turn on, the lights are going to stay off.

Each of the physical outputs of CANSwitch is controlled by a Master Relay. A Master Relay can control more than one physical output, though. For example, if outputs O:1 and O:2 are both controlled by Master Relay 1, then turning off Master Relay 1 will cause both outputs O:1 and O:2 to be turned off, irrespective of any other ladder diagram elements that wants to turn it on. The physical outputs will only be usable once the Master Relay is turned back on again.

A Master Relay does not have a separate symbol in a ladder diagram, it is turned on or off by writing the Master Relay number above a coil. In the Ladder Editor program you can select which outputs of the CANSwitch are associated with which Master Relays.



In this example, Master Relay #1 is toggled on or off every time the handlebar's Indicator Cancel button is pressed and held for 1 second. All outputs associated with Master Relay #1 will then turn off or back on.

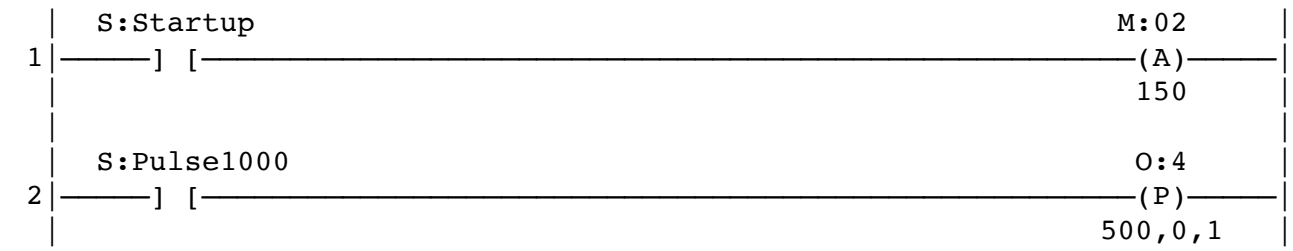
System Variables

CANSwitch has a few System Variables. These are variables that are predefined by CANSwitch itself, but which you can use in your ladder diagram.

System Variable	Usage	Description
S:Startup	Startup	This variable is like a switch that is on only once - during the first scan of the diagram. Thereafter it remains off. This can be used as a switch to set initial values for memories as soon as the CANSwitch starts.
S:Pulse10	10mS Pulse	This variable is on once every 10mS
S:Pulse100	100mS Pulse	This variable is on once every 100mS
S:Pulse1000	1000mS Pulse	This variable is on once every 1000mS (1 second)
S:Power*	Power Control	You can turn this variable on and off with a coil. When on, the CANSwitch remains powered - even if the motorcycle's ignition is turned off. When off, the CANSwitch will turn off when the motorcycle's ignition switches off.

** S:Power is only available on CANSwitch Series 4. It has no effect on Series 3.*

Here's an example of using S:Startup (startup pulse) and S:Pulse1000 (1 second pulse):



Rung 1 will set memory M:02 to a value of 150 when CANSwitch starts up (when the ignition is turned on).

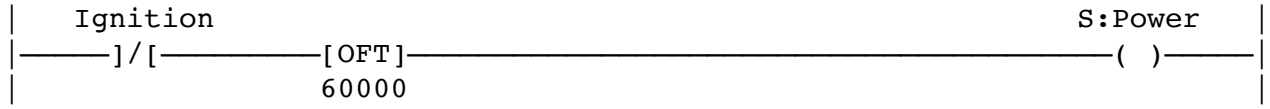
Rung 2 will cause output #4 to be pulsed for 500mS once every second.

Controlling Power

CANSwitch is turned on (powered up) when the motorcycle's ignition is turned on. When the motorcycle ignition is turned off the CANSwitch will remain powered for approximately 60 seconds on the R1200GS LC models. Some models, like the R1200GS air cooled, turns off immediately when you turn off the ignition, which makes it hard for CANSwitch to keep an accessory powered while the ignition is off.

With CANSwitch Series 4 it is possible to keep the CANSwitch powered on, even after the motorcycle's ignition is off. This can be done by using the system variable S:Power as a coil. When this coil is on it will keep the CANSwitch powered. As soon as the coil is turned off (and the ignition is also off) the CANSwitch will turn off.

Here is an example of keeping the CANSwitch powered on:



In this example the coil, S:Power, will stay on as long as the ignition is on. Once the ignition turns off, the Off timer (—[OFT]—) will keep the coil on for a further 60 seconds. Then the coil is turned off, resulting in CANSwitch itself turning off.

S:Power is off by default when you turn the motorcycle's ignition on. Your ladder diagram needs to turn it on if you want the CANSwitch to remain powered after the ignition turns off. If your ladder diagram turns on S:Power, but never turns it off again, then CANSwitch will stay turned on. This will drain the battery, so make sure you turn S:Power off after a period of time and do not leave it on permanently.

Pulse Timers

A Pulse Timer measures how long a condition remains true for. This can be used to time the duration of events.

```
<IN>———[PT]———<OUT>  Pulse Timer
```

When the <IN> of a Pulse Timer turns on, the timer starts measuring. During this time the <OUT> remains off. As soon as <IN> turns off, the measured time (in ms) is stored, and <OUT> turns on for 1 scan cycle. During the following scan cycle the <OUT> will turn off again.

As an example, this functionality can be used to make the ABS switch activate a gate remote if the ABS button is pressed only briefly. Normally when you use the ABS button to activate or deactivate the motorcycle's ABS or traction control system, you would press the button for 1 or 2 seconds. We don't want the transmitter to turn on in that case, we only want it to turn on if the button is pressed briefly. So with a Pulse timer we can measure how long the ABS button is pressed for.

```
|      ABS     M:02     M:02                                      O:3      |
|———————] [———[PT]———[<=]———————————————————————————————————(P)———————|
|                      500                                  1000,1,1    |
```

Here, the Pulse Timer will measure how long the ABS button is pressed for, and store the duration in memory M:02. It will also pulse its <OUT> once when the measurement is complete. This single pulse of the <OUT> enables the comparator (more about that later), which in turn will pulse output #3 if the duration is less than or equal to 500ms.

When you use a Pulse Timer you must specify a memory where the measured value must be stored, by writing it above the timer.

Comparators

A comparator compares 2 values, and turns its <OUT> on if the comparison holds true, or turns it off if the comparison is false. The comparison is only performed if the <IN> is on. If the <IN> is not on then the comparison is not performed, and the <OUT> remains off.

Two values are compared, they are called the *LValue* and the *RValue*:

LValue : *RValue*

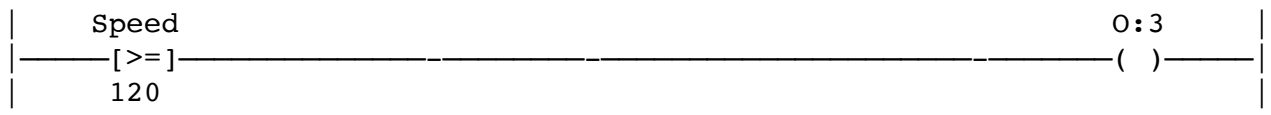
The *LValue* is the one to the left of the operator, the *RValue* value is on the right.

The *LValue* can be any of the CAN values (speed, RPM, temperature, etc) or a memory (M:01 to M:64). The same goes for the *RValue.* The *RValue* can also be a number in the range -32768 to 32767.

The operator between the *LValue* and the *RValue* determines what kind of comparison is performed. Available operators are:

```
<IN>———[<]———<OUT>  Less than
<IN>———[>]———<OUT>  Greater than
<IN>———[=]———<OUT>  Equals
<IN>———[<>]——<OUT>  Not equal
<IN>———[<=]——<OUT>  Less than or equal to
<IN>———[>=]——<OUT>  Greater than or equal to
```

How its used:



In this example, the output #3 is turned on if the speed is greater than or equal to 120Km/h.

Detecting Rising and Falling edges

The term, Rising Edge, is used to signify the moment that something changes from off to on. Conversely, a Falling Edge is used to signify the moment a state changes from on to off.

This can used when you want an action to be performed only once when a condition becomes true, instead of being performed repeatedly while the condition remains true.

```
<IN>——[OSR]——<OUT>  One Shot Rising
<IN>——[OSF]——<OUT>  One Shot Falling
```

When the <IN> of an —[OSR]— turns on, and it was off before, then the <OUT> turns on.
When the <IN> of an —[OSF]— turns off, and it was on before, then the <OUT> turns on.

Consider this example:

```
|     ABS                                                          O:3     |
|——————] [———[ONT]———[OSR]————————————————————————————————————(T)———————|
|             1000                                                         |
```

In the above, if the ABS button is held for 1 second then output #3 is toggled (turned on if it was off, or turned off if it was on). If the —[OSR]— was not there then the output would keep toggling on and off for as long as the ABS button was pressed (after the initial 1000mS delay). Adding the —[OSR]— ensures that output #3 is only toggled once, and the ABS button must be released and pressed again to repeat the action.

Symbol Reference

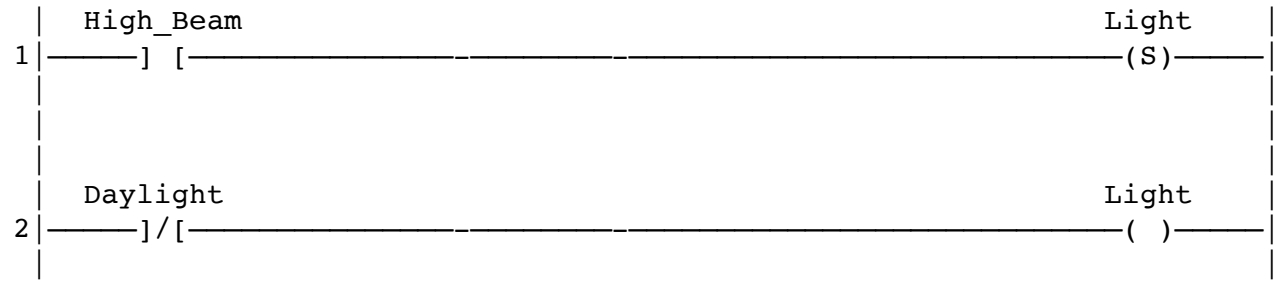
Token	Name
<Var> <IN>——] [——<OUT>	**Normal Switch**. <Var> can be any CAN variable or M:01 to M:64. When using a memory, any non-zero value turns the switch on and 0 turns it off.
<Var> <IN>——]/[——<OUT>	**Inverted Switch**. <Var> van be any CAN variable or M:01 to M:64. When using a memory, any non-zero value turns the switch off and 0 turns it on.
<IN>——[OSR]—<OUT>	**One Shot Rising Edge**. <OUT> is on once when <IN> turns on.
<IN>——[OSF]—<OUT>	**One Shot Falling Edge**. <OUT> is on once when <IN> turns off.
<Var> <IN>——( )——<OUT>	**Normal Coil**. <Var> is on while <IN> is on. <Var> can be O:1 to O:8, MR:1 to MR:4 or M:01 to M:64.
<Var> <IN>——(I)——<OUT>	**Inverted Coil**. <Var> is on while <IN> is off. <Var> can be O:1 to O:8, MR:1 to MR:4 or M:01 to M:64.
<Var> <IN>——(S)——<OUT>	**Set Only Coil.** <Var> turns on when <IN> turns on, and stays on even if <IN> turns off again. <Var> can be O:1 to O:8, MR:1 to MR:4 or M:01 to M:64.
<Var> <IN>——(R)——<OUT>	**Reset Only Coil**. <Var> turns off when <IN> turns on, and stays off even if <IN> turns off again. <Var> can be O:1 to O:8, MR:1 to MR:4 or M:01 to M:64.
<Var> <IN>——(T)——<OUT>	**Toggle Coil.** <Var> is toggled every time when <IN> is on. <Var> can be O:1 to O:8, MR:1 to MR:4 or M:01 to M:64.
<Var> <IN>——(A)——<OUT> <Value>	**Analog Coil**. <Value> is stored into <Var> when <IN> is on. <Var> can be O:1 to O:8, M:01 to M:64 or MR:1 to MR:4. When <Var> is O:1 to O:8 then a <Value> from 0 to 1000 sets it to 0..100% brightness.
<Var> <IN>——(P)——<OUT> <On>,<Off>,<Count>	**Pulsed Coil.** <Var> is pulsed <Count> times. Each time it is on for <On> ms, and then off for <Off> ms. <Var> can be O:1 to O:8, MR:1 to MR:4 or M:01 to M:64. During the on cycle of the pulse a value of 1000 is stored if <Var> is a memory, and the value 0 is stored during the off cycle.
<Var> <IN>——[C+]——<OUT> <Thres>,<Step>,<Lim>	**Up Counter**. The value in <Var> in incremented with <Step> every time <IN> goes on. If the count reaches or exceeds <Thres> then <OUT> is on. Counting will stop when <Lim> is reached. <Var> is optional.
<Var> <IN>——[C-]——<OUT> <Thres>,<Step>,<Lim>	**Down Counter**. The value in <Var> in decremented with <Step> every time <IN> goes on. If the count reaches <Thres> then <OUT> is on. Counting will stop when <Lim> is reached. <Var> is optional.
<Var> <IN>——[T+]——<OUT> <Int>,<Step>,<Lim>	**Up Time Counter**. The value in <Var> in incremented with <Step> every <Int> ms, as long as <IN> is on. Counting will stop when <Lim> is reached. <OUT> is on if <IN> is on, and off otherwise.
<Var> <IN>——[T-]——<OUT> <Int>,<Step>,<Lim>	**Down Time Counter**. The value in <Var> in decremented with <Step> every <Int> ms, as long as <IN> is on. Counting will stop when <Lim> is reached. <OUT> is on if <IN> is on, and off otherwise.
<Var> <IN>——[RES]—<OUT>	**Reset**. If <IN> is on then <Var> is set to 0. If <IN> is not on then <Var> is not affected.
<IN>——[ONT]—<OUT> <Time>	**On Timer**. <OUT> is turned on <Time> ms after <IN> has turned on.

Token	Name
<IN>——[OFT]—<OUT> <Time>	**Off Timer**. <OUT> is turned off <Time> ms after <IN> has turned off.
<Var> <IN>——[PT]——<OUT>	**Pulse Timer**. The time, in ms, that <IN> is on for is measured, and stored in <Var> the moment <IN> goes off. <OUT> goes on for 1 cycle when the measurement is completed (i.e. when <IN> goes off).
<LValue> <IN>——[<]——<OUT> <RValue>	**Less than Comparator**. <OUT> is on if <IN> is on and <LValue> is less than <RValue>, otherwise <OUT> is off. <LValue> can be any CAN variable, M:01 to M:64 or MR:1 to MR:4. <RValue> can be too, or can be a number from -32768 to 32767.
<LValue> <IN>——[>]——<OUT> <RValue>	**Greater than Comparator**. <OUT> is on if <IN> is on and <LValue> is greater than <RValue>, otherwise <OUT> is off. <LValue> can be any CAN variable, M:01 to M:64 or MR:1 to MR:4. <RValue> can be too, or can be a number from -32768 to 32767.
<LValue> <IN>——[=]——<OUT> <RValue>	**Equals Comparator**. <OUT> is on if <IN> is on and <LValue> equals <RValue>, otherwise <OUT> is off. <LValue> can be any CAN variable, M:01 to M:64 or MR:1 to MR:4. <RValue> can be too, or can be a number from -32768 to 32767.
<LValue> <IN>——[<>]——<OUT> <RValue>	**Not equals Comparator**. <OUT> is on if <IN> is on and <LValue> does not equal <RValue>, otherwise <OUT> is off. <LValue> can be any CAN variable, M:01 to M:64 or MR:1 to MR:4. <RValue> can be too, or can be a number from -32768 to 32767.
<LValue> <IN>——[<=]——<OUT> <RValue>	**Less than or equal to Comparator**. <OUT> is on if <IN> is on and <LValue> is less than or equal to <RValue>, otherwise <OUT> is off. <LValue> can be any CAN variable, M:01 to M:64 or MR:1 to MR:4. <RValue> can be too, or can be a number from -32768 to 32767.
<LValue> <IN>——[>=]——<OUT> <RValue>	**Greater than or equal to Comparator**. <OUT> is on if <IN> is on and <LValue> is greater than or equal to <RValue>, otherwise <OUT> is off. <LValue> can be any CAN variable, M:01 to M:64 or MR:1 to MR:4. <RValue> can be too, or can be a number from -32768 to 32767.

Output prioritisation

During a scan of the ladder diagram a coil may affect a physical output, causing it to turn on, off or be set to any brightness value. Once a physical output is affected by a rung, no other rungs below that one can affect the output during that scan. This is called output prioritisation.

Consider the following diagram:



In this diagram, rung #2 will turn the light on at night, and turn it off during the day (the Daylight variable is obtained form the CAN bus). But, if the high beam is on then the light is switched on by means of the —(S)— coil. During this time you do not want the daylight sensor to turn the light on or off.

Because the high beam is actively controlling the light (turning it on), the daylight sensor has no effect on the light. Only when the high beam is turned off again can the daylight sensor continue controlling the light.

Important notes

Keep these things in mind when using ladder diagrams:

1. The complete ladder is scanned (or processed) every 5ms. When using the pulsed coil you should use time values that are in steps of 5ms.
2. The Pulse timer measures in steps of 5ms.
3. When using the Time counter, specify interval values in steps of 5mS.
4. Take care when using System Variable S:Power to keep the CANSwitch powered on. If your ladder diagram turns S:Power on, but never turns it off again the CANSwitch will stay on, placing a continuous load on the motorcycle's battery (assuming the motorcycle's ignition is also off and thus the engine is not running to charge the battery). This could drain the battery, which may prevent you from being able to start your motorcycle. Draining your battery completely will drastically reduce its life expectancy.

Liability

The CANSwitch is designed to withstand the harsh electrical environment of motorcycles. While it protects itself against short circuits, reverse voltages and high currents, prolonged exposure to such adverse conditions may damage it and render it inoperable. The CANSwitch is rigorously tested before shipment, and guaranteed to be in proper working order. However, due to the self-installation of the CANSwitch, we do not accept any responsibility or liability for damage to the unit or the motorcycle it is installed on. We do not undertake any repairs to such damaged units or motorcycles, nor do we provide a refund or exchange a damaged unit.

While CANSwitch was designed with ease of installation in mind, and while it eliminates the need to cut into any of the motorcycle's existing wires and thereby helps protect the motorcycle's warrantee, it has not been approved by any motorcycle manufacturer as an aftermarket accessory. The onus is on the user to ensure proper installation and the user thus carries any and all risks.

If you have any doubt whatsoever about any electrical connections to be made we advise you to seek the help of an auto electrician. Help is also always available via email to info@canswitch.co.za