

## Wipers overview

Windshield wipers are a crucial part of a race car and cannot afford to fail when rain falls during a race. You can configure the wiper control output from a power box, such as the Centaurus, or an older device such as the IPS, in many ways. This guide provides an overview of how to configure a wiper control output from the **Wipers** node, as well as other 'traditional' methods using maths and logic channels.

## Wipers node

You can configure wiper control outputs from the **Wipers** node. Enable the wiper control output in the **Wipers** node (1), and then select the **LIN Port** to which the wiper is connected (2). Select the **LIN Bit Rate** from the dropdown menu to match the LIN bus bit rate (3.). You can select one of three common wiper models from the dropdown menu (4).

Finally, use the 'browse' button to specify the channel to drive the control (5).

**Note:** The wiper control channel can be a logic channel, a Maths channel, or a channel received over CAN, but the data type must be in U32 format.

**Wipers**

General

Configure the settings for the wiper motor.

Enabled 1

LIN Port 2 LIN 01

LIN Bit Rate 3 19200

Wiper Model 4 F 02U V00 838-03 Lin ID: 0x31

Control Channel 5 Wiper Control Channel

## Manual configuration of wiper output control

### Wiper basic

Analog wipers usually require a minimum of four wires to be connected to control individual motor windings. These wires normally consist of:

- Power
- GND
- Slow coil
- Fast Coil



Unlike LIN wipers, where the logic is controlled on the wiper motor itself, the logic to control the wiper motor needs to be integrated into the setup of the device controlling the motor. In this case, a Centaurus 5 is being used as an example.

The basic requirements to control an analogue wiper motor are outlined in this document, but you can implement further logic to add features such as intermittent settings.

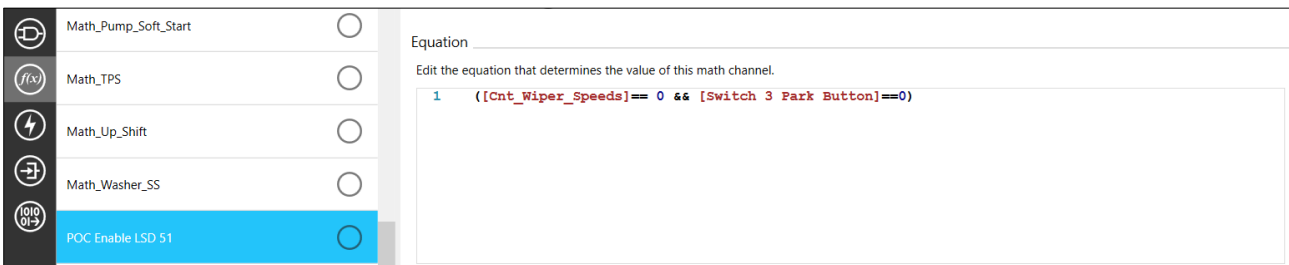
In this example, the control logic is written on the **Logic Channels** node to create a simple setup, but this logic can also be written in the maths channels if required.

The logic for the wiper is triggered on the button press of a switch panel. This can be via an RSP or CAN input from external devices.

## Maths channels

### *POC Enable LSD 51*

The 'Park' feature of the Centaurus (the LSD) needs to be activated via the maths channel **POC Enable LSD 51** option. This channel name must match to invoke the LSD.



	Math_Pump_Soft_Start	<input type="radio"/>
	Math_TPS	<input type="radio"/>
	Math_Up_Shift	<input type="radio"/>
	Math_Washer_SS	<input type="radio"/>
	<b>POC Enable LSD 51</b>	<input checked="" type="radio"/>

Equation

Edit the equation that determines the value of this math channel.

```
1 ((Cnt_Wiper_Speeds)== 0 && [Switch 3 Park Button]==0)
```

The example maths above show, the two conditions of the wiper speeds and the switch park. These conditions are fully customisable but, in this case, a physical button press is being used. These messages can come from anywhere, for example from the CAN bus or from digital inputs.

## Logic channels

### *Wiper button press*

This channel receives the input from the physical button. When the button is pressed, the logic channel returns 'True'.



**General**  
Configure the basic settings that define this condition channel.

Name

Comment

Manufacturer Status  This is a normal item.

---

**Output**  
Configure the output states for this condition channel.

Configure the text and colors for the channel.

when True

when False

---

**Logic**  
Configure the logic that determines the output of this condition channel.

↑ ↓ 🗑️

All

"RSP Switch7 Button" "De-bounced" is Active

+ Group: All  
Type: All

## Wiper counter

This channel is a counter which determines how many times the button has been pressed. It is set to increase on each button press and determines at which speed to run the wipers.

**General**  
Configure the basic settings that define this counter channel.

Name

Comment

Manufacturer Status  This is a normal item.

---

**Counter**  
Configure the settings that define the counter.

Min / Max

Initial

Increment on  edge of  ⋮

Wrap At Limit  Hold At Limit

Decrement on  edge of  ⋮

Wrap At Limit  Hold At Limit

---

**Overrides**  
Configure any overrides to apply to the counter value.

Set to  when  ⋮

⌵

Set to  when  ⋮

⌵

In this example you can only press the button three times for the three different states:

1. Intermittent Wiper
2. Slow Wiper
3. Fast Wiper

### Intermittent wiper

To control the intermittent setting, a timer needs to be used to set the active high status. You can define the timer, specific to your requirements.

### Slow wiper

When this logic returns 'True', the wipers activate in slow mode. This is controlled by the counter and by setting it to return 'True' when 'Cnt\_Wiper\_Speeds' is equal to 1. Other conditions can be added to make sure that wipers do not run when the kill switch is active and when the car is powered off.

Alternatively, if you press the washer button this automatically enables the 'Slow wipe' and bypasses the other parameters.

The screenshot shows a configuration window for a condition channel. It is divided into three main sections: General, Output, and Logic.

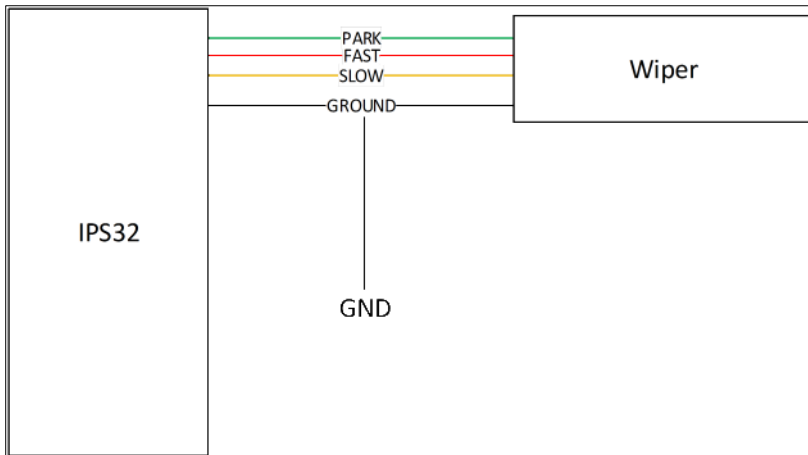
- General:** Contains fields for Name (Lgk\_Slow\_Wipe), Comment, and Manufacturer Status (radio button).
- Output:** Contains fields for 'when True' (True) and 'when False' (False), each with a corresponding color swatch (green for True, grey for False).
- Logic:** Contains a logic tree. The root is 'Any'. Under 'Any', there is an 'All' condition. The 'All' condition has three sub-conditions: '"Lgk\_Ignition" = 1.000', '"Cnt\_Wiper\_Speeds" = 2.000', and '"Cnt Kill" = 0.000'. Below these is a '+' sign. At the bottom of the logic tree is another condition: '"Lgk\_Washer" = 1.000? True after 0.75s, False after 2.00s'. To the right of the logic tree is a 'Group' dropdown menu set to 'Any' and a 'Type' dropdown menu set to 'Any'.

### Wipers advanced

**Note:** This guide describes how you can configure analogue wipers for a IPS32 or IPS48 device. These are only guidelines on how to set up your IPS32 with wipers and the precautions to take. You can modify these guidelines to meet customer requirements, but setups should always be tested before being implemented on a car.

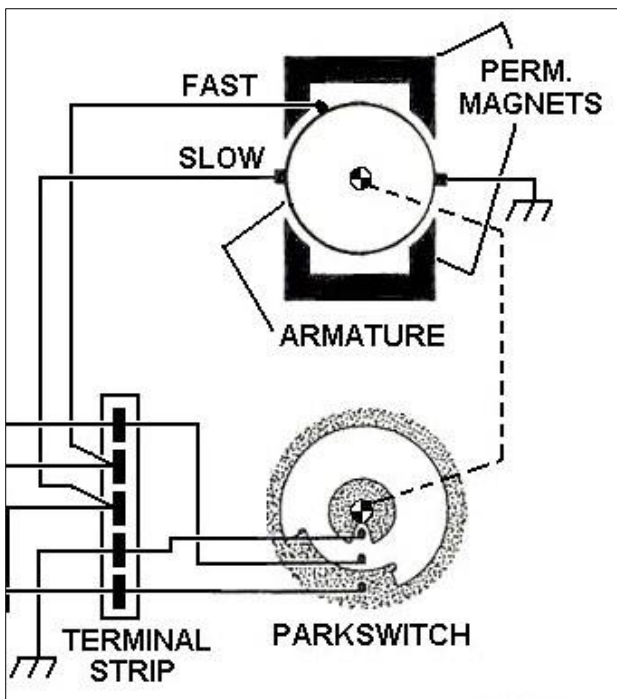
## General principle

Unlike the LIN wiper, the analogue wiper does not have any inbuilt 'intelligence'. The controller (in this case the IPS) must tell the wiper when to go slow, go fast, or stop. The loom between the IPS and the wiper is usually composed of 4 wires.



The ground link is usually already routed directly to the chassis. The FAST and SLOW lines are the 'power' lines which supply 12V to the fast and slow motors, respectively. The SLOW line supplies current to the full length of the motor windings, whereas the FAST line supplies current to half the length of the motor windings. This is how the two different wiper speeds are controlled.

The PARK signal is a signal from the wiper system that the IPS uses to calculate when to stop and apply the brake. It is usually either a tooth or missing tooth on the wiper motor track. As the motor rotates it passes this (missing) tooth once a rotation and a mechanical switch linked to this track outputs a digital signal depending on whether a contact is registered.





## Precautions when you connect to the IPS

To stop the wipers in a controlled and accurate manner, the motor can be 'braked' by removing the live current supply to the motor and connecting it directly to ground. Outputs 1 and 17 on the IPS have this feature, so it is recommended that these outputs are used to supply the slow motor. To apply this brake, either the channel '[POC Enable LSD 1]' or the channel '[POC Enable LSD 17]' must output 1.

Without this feature you can only supply live current to the wiper, so stopping the wiper in the correct position is difficult to do accurately, relying on the friction applied to the wipers and the inertia of the motor to slow it to a halt. With the added complication of varying inertia on the wipers themselves (depending on the condition of the surface of the windscreen and the speed of the vehicle), an active stopping mechanism is required to make sure that the wipers stop in the 'park' position.

**Note:** You must take care when you apply the 'brake' as the IPS is not protected against accidentally supplying live current to the output and connecting it to ground at the same time. This could cause serious damage to both the IPS and the wiper.

## Control the wipers

In this example, five math channels control analogue wipers. Below is an explanation of the function of these five math channels and the names assigned to them:

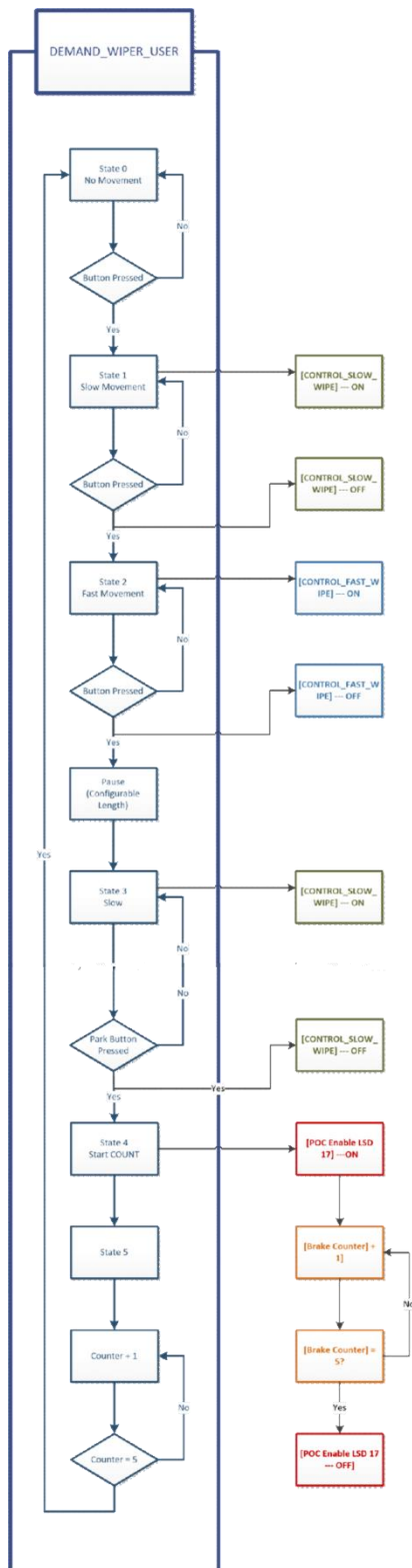
1. One master channel runs as a 'state machine' and controls the state of the system [DEMAND\_WIPER\_USER]
2. One that controls the 'Slow' wipe - [CONTROL\_SLOW\_WIPE]
3. One that controls the 'Fast' wipe - [CONTROL\_FAST\_WIPE]
4. One that controls the 'Brake' - [POC Enable LSD 17]
5. One to run as a counter for the duration that the 'Brake' is applied - [Brake Counter]

The driver in this example has one button (usually on the steering wheel) enables them to switch between SLOW/FAST/OFF.

**Note:** This is only an example, and might not suit every customer (for example, the sequence means that the wiper operates in the following order to be turned off, which may not suit some applications): SLOW > FAST > OFF.

When the system is running Fast and a Stop is requested, it returns to Slow before stopping at the park signal. This means that the process sequence is: SLOW > FAST > (SLOW > BRAKE > OFF).

The flowchart below explains the state machine process that the system follows, with conditions for each change. 'State' represents the value of the output from the channel 'DEMAND\_WIPER\_USER':





## Link to the outputs

The output channels to power the two speeds are '[CONTROL\_SLOW\_WIPE]' and '[CONTROL\_FAST\_WIPE]'. Allocate '[CONTROL\_SLOW\_WIPE]' to either output 1 or 17 (the channels in this example are set up to use output 17).

Allocate '[CONTROL\_FAST\_WIPE]' to output 18. The loom should have pins connected from these sockets to the power inputs on the wiper motor.

Set up the Park signal, received as a digital input from the wiper motor, as a button named '[WIPER\_PARK Button]'.

Set the wiper control button as a button named '[WIPER Button]'. The ground for the wiper must go directly to ground.

## Appendix

### Master wiper channel - DEMAND\_WIPER\_USER

This is the channel that controls the changing state of the wiper system, on which output functions rely and into which other channels feed information. To keep all these synchronised, it is recommended that all wiper channels are calculated at the same rate (for example, 50Hz).

#### DEMAND\_WIPER\_USER

```

a5 ( [WIPER_PARK Button] & 1 );
a6 ( choose ( ( @a5 > @a4 ), 1, ( choose ( (@a5 < @a4 ), 2, 0 ) ) ) );
a4 ( @a5 );
a2 ( choose ( @a2 == 4, 5, @a2 ) );
a0 ( choose( ((shr([WIPER Button], 1) & 1) > @a1, 1, 0) );
a1 ( (shr ( [WIPER Button], 1) & 1) );
a2 ( choose ( ( ( @a0 ) == 1 ), choose ( @a2 == 3, 3, choose ( @a2 < 4, @a2 + 1, 4 ) ), @a2 ) );
a2 ( choose ( ( @a2 == 3 ) && ( @a6 == 2 ), 4, @a2 ) );
a3 ( choose ( ( @a2 == 4 ), 0, @a3 ) );
a3 ( choose ( ( @a3 < 6 ), @a3 + 1, 6 ) );
a2 ( choose ( ( @a3 == 5 ), 0, @a2 ) );
@a2

```



There are seven registers in this channel (a0, a1, a2, a3, a4, a5, a6, a7). Each of these has a different function in the calculation of the channel output. To explain the purpose of each register, from the start of the channel:

- a4, a5, and a6 are all used to calculate whether a rising or falling edge has been received from the 'WIPER\_PARK Button'.
- **a5:** This register takes the value of the wiper park button. When the system is in the park position (once per revolution) the value of a5 is 1. At all other times it is zero.
- **a4:** This register displays the value of a5 from the previous cycle to allow the two to be compared by a6.
- **a6:** This register detects whether a rising edge or falling edge of the WIPER\_PARK Button is received. This is done by comparing a4 and a5: if a5 is larger than a4 (as this is a digital signal, this means that a5 is 1 and a4 is 0), then a rising edge is detected, as the current value is 1 and the previous value is 0. Conversely, if a4 is larger than a5, then a falling edge is detected. a6 displays 1 for a rising edge, 2 for a falling edge, and 0 if no change is seen.
- **a0 and a1 are used to recognise a rising edge of the WIPER Button.**
- **a0:** This register compares the current value from the WIPER Button (1 or 0) with the previous value (held by a1). If a rising edge is detected, the current value is greater than the previous value and a0 outputs a 1, otherwise 0.
- **a1:** This register holds the previous state of the WIPER Button for comparison with a0.
- **a2:** This is the final output from the channel, which has six states.
- **a2 = 0:** Initial state of the system, no power applied until a button is pressed.
- **a2 = 1:** Button has been pressed once, power applied to slow motor.
- **a2 = 2:** Button has been pressed again, power applied to fast motor and not to slow motor.
- **a2 = 3:** Button has been pressed again and a stop requested. This button press sets in motion the remaining states. The system returns to powering the slow motor only and remains in this state until a rising edge of the WIPER\_PARK Button is detected (a6 = 1).
- **a2 = 4:** WIPER\_PARK Button rising edge is detected, power is removed from slow motor, brake applied, counter is started, and state moves to a2 = 5.
- **a2 = 5:** Brake is held on for set duration, then a2 moves back to state 0, with wipers stopped and no power to any motors or brakes.
- **a3 is a counter to keep track of whether the brake is on or off.**
- **a3:** This register counts from 0 to 6. It starts from a3 = 0 and a2 shifts to 0 when it detects that a3 = 5; a3 resets to 0 once it reaches a3 = 6. This register is required to make sure that the brake is not applied at the same time as power is applied to one of the motors.

Initially the channels were set up so that the brake came on once it detected that '[DEMAND\_WIPER\_USER]' was at state 4, stayed on for a set amount of time, and then '[DEMAND\_WIPER\_USER]' moved to state 5 once it detected that the brake was off. However, this was not possible because the two maths channels could not both be dependent on each other. Therefore, two counters were created which worked simultaneously. They both start when they detect that '[DEMAND\_WIPER\_USER]' is at state 4, but one turns off the brake when it reaches a certain value, and the other moves '[DEMAND\_WIPER\_USER]' from state 5 to state 0 when it reaches that same value. Register a3 is the counter which controls the change of state from 5 to 0. A separate channel, 'Brake Counter' controls turning off the brake. '[Brake Counter]' is displayed below:

## Brake Counter

```
a1 ( choose ( ( [DEMAND_WIPER_USER] == 4 ), 0 , choose ( ( @a1 < 6 ), @a1 + 1 , @a1 ) ) );  
@a1
```

This counter is reset to zero when it detects that DEMAND\_WIPER\_USER = 4, then increments up by one each cycle until it reaches 6, then rests at 6 until it detects DEMAND\_WIPER\_USER = 4 again. The brake is controlled by the channel 'POC Enable LSD 17' (shown below):

### POC Enable LSD 17 (Alternatively 'POC Enable LSD 1' if using output 1)

```
a1 ( choose ( ( [DEMAND_WIPER_USER] == 4 ), 1 , ( choose ( ( [Brake Counter] < 5 ), 1 , 0 ) ) ) );  
@a1
```

This channel applies the brake whenever it detects that either DEMAND\_WIPER\_USER = 4 or Brake Counter is below 5. Therefore, as soon as Brake Counter = 6, the brake is off.

## Power Outputs

The remaining two channels are the outputs from the IPS to control the slow and fast motion of the wipers. These are CONTROL\_SLOW\_WIPE and CONTROL\_FAST\_WIPE: **CONTROL\_SLOW\_WIPE**

```
a0 ( choose ( ( [DEMAND_WIPER_USER] == 1 ), 1 , 0 ) );  
  
a1 ( choose ( [DEMAND_WIPER_USER] == 3 , choose ( @a1 == 5 , 5 , @a1 + 1 ), 0 ) );  
  
a2 ( choose ( ( @a0 == 1 || @a1 == 5 ) && ( [POC Enable LSD 17] == 0 ), 1 , 0 ) );  
  
@a2
```

This channel engages the slow mode of the wiper motor when DEMAND\_WIPER\_USER = 1 and when DEMAND\_WIPER\_USER = 3 (after a delay of 5 cycles). In both cases, POC Enable LSD 17 must be zero for the motor to be engaged.

### CONTROL\_FAST\_WIPE

```
choose ( ( [DEMAND_WIPER_USER] == 2 ) && ( [CONTROL_SLOW_WIPE] == 0 ) && ( [POC Enable LSD  
17] == 0 ) , 1 , 0 )
```

This channel engages the fast mode of the wiper whenever all of the following conditions are met: DEMAND\_WIPER\_USER = 2, CONTROL\_SLOW\_WIPE is off and POC Enable LSD 17 is off.

**Note:** Some logical AND (&&) and logical OR (||) functions are used in these math channels. Care must be taken when you use these. It is strongly recommended that parenthesis are used wherever appropriate to separate each member of a logical operation to avoid miscalculations:

```
( xxxxxx > 1 ) && ( yyyyyy == 2 )
( ( xxxxxx == 0 ) || ( yyyyyy > 4 ) ) && ( zzzzzz != 0 )
```

## Function explanations

Function	Description	Syntax
<b>choose</b>	The choose function is a logical 'if', which gives one output if the statement is true, and another output if the statement is false.	choose('Statement' , Output if statement true , Output if statement false )
Registers	Variables to which values can be set. Up to 7 registers can be used in any channel, identified by their number (ranging from 0 - 6)	To set register value: a1( <i>value</i> )  To reference register in other function: @a1  Available registers:  a0 /@a0 a1 / @a1 a2/ @a2 a3 /@a3 a4 /@a4 a5 /@a5 a6 /@a6
		Example:  a2 ( 7 ) ; ----> set a2 value to 7  choose ( @a2 == 7 , 1 , 0 ) ; ----> If a2 is equal to 7, output 1, otherwise output 0
==	Logical 'is equal to'	choose ( @a4 == 1 , 3 , 4 ) ; ----> If register @a4 is equal to 1 then output 3, otherwise output 4
<	Logical 'is less than'	choose ( @a1 < @a5 , 9 , 2 ) ; ----> If register @a1 is less than the register @a5, then output 9, otherwise output 2
>	Logical 'is greater than'	choose ( @a1 > @a5 , 9 , 2 ) ; ----> If register @a1 is greater than the register @a5, then output 9, otherwise output 2
+	Mathematical 'plus'	( @a6 + 1 ) ;
&&	Logical 'and'	choose ( ( @a2 == 3 ) && ( @a4 == 4 ) , 6 , 2 ) ; ----> If register @a2 is equal to 3 and register @a4 is equal to 4, then output 6, otherwise output 2

	Logical 'or'	choose ( ( @a3 == 3 )    ( @a3 == 4 ) , 8 , 5 ); ---> If register @a3 is equal to 3 or register @a3 is equal to 4, then output 8, otherwise output 5
shr	Shifts byte right by X bits, inserting zeroes to the left	shr ( [Input 4] , 3 ); ---> Shifts the input from [Input 4] right by 3 bits, with 3 zeroes inserted to the left
&	Returns a U32 'ANDed' with another U32	[Input 7] & [Input 6]; ---> Returns 1 if [Input 7] and [Input 6] have the same value