Android can bus driver

Continue

Android can bus settings. Android can bus interface. Android can bus apk.

This section explains the steps required to send and receive CAN bus messages in Squish tests. CAN is a message bus standard that allows the microcontroller and other devices (collectively known as the ECU) to communicate without a central host or bus manager. It originated in the automotive industry but has since been adopted for many other applications. For a detailed description of the Squish CAN API, see the CAN Bus API documentation. Where testing requires complex interactions or detailed ECU simulation, it may be more beneficial to use third-party software that specializes in CAN bus simulation and can interface with Squish via FMI support. In a typical test setup, the built-in AUT capable device will be connected to the CAN bus. To enable bus communication in a test scenario, the test controller must have a compatible CAN controller connected to the same bus. CAN Bus Squish Test Setup Diagram CAN Bus Design All references to a CAN interface must refer to its own application which identifies the particular system connected to the bus. An application context that supports the CAN bus API can be created using the ApplicationContext startCAN(options) function. You can then connect to the CAN controller and start sending and receiving messages. javascript" checked/>javascriptpythonperl" />perlrubytcljavascript">var canContext = startCAN(); var device = new CanBusDevice("socketcan", "can0"); var frame = new CanBusFrame(0x100, "4121999a"); device.writeFrame (frame);canContext = startCAN() device = CanBusDevice("socketcan", "can0") frame = CanBusFrame(0x100, "4121999a") device.writeFrame(frame) my $canContext = startCAN(); my $device = CanBusDevice ->new("socketcan", "can0"); my $frame = CanBusFrame->new(0x100, "4121999a"); $device->writeFrame(frame); canContext = startCAN(); device = CanBusDevice.new("socketcan", "can0");frame = CanBusFrame.new(0x100, "4121999a");set canContext [startCAN] set device [CanBusDevice new socketcan can0] set frame [CanBusFrame new 0x100 00deadbeef00] CanBusDevice invoke $device writeFrame $frame Supported CAN drivers and devices that use them can be listed using the List CanBusDevice.pluginNames() and Static methods CanBusDevice at .availableDevices(driver). javascript" checked/>javascript">javascriptpythonperl" />perlrubytclvar canContext = startCAN(); var plugins = CanBusDevice.pluginNames(); for (var i in plugins) { var plugin = plugins[i]; test.startSection(plugin) ; try { var devices = CanBusDevice.availableDevices(plugin); for (var j in devices) { test.log("Device: " + devices[j].deviceName); } } catch (e) { test.log(" Error : " + e.message); } test.endSection(); } canContext = startCAN() plugins = CanBusDevice.pluginNames() for plugin in plugins: test.startSection(plugin) try: devices = CanBusDevice.availableDevices(plugin) for device in devices: test.log("Device: %s" % device.deviceName) except exception as e: test.log("Failed: %s" % str(e)) test.endSection() my $canContext = startCAN (); my @plugins = CanBusDevice->pluginNames(); foreach (@plugins) { my $plugin = $_; test::startSection($plugin); eval { @devices = CanBusDevice-> availableDevices($plugin) ; foreach (@devices) { test::log("Device: $_->dev iceName"); } } or { test::log("Error: $@"); } test::endSection(); } canContext = Squish.startCAN() plugins = CanBusDevice.pluginNames() plugins.each { |plugin| Test.startSection(plugin) begin devices = CanBusDevice.availableDevices(plugin) devices.each { |device| Test.log("Device: " + device.deviceName) } save Exception => e Test.log("Failed: " + e.message) end Test.endSection() } startCAN set plugins [call canBusDevice pluginNames] for each plugin $ plugins { test startSection $plugin if { [catch { set devices [call canBusDevice availableDevices $plugin] foreach device $devices { test log [concat "Device: " [property get $device deviceName]] } } err] } { test log"Error: " $err] } test endSection } Frame content The CAN standard does not define the content of the frame payload - this is left to the developers of CAN networks and engine control units. Because of this, Squish can only interpret the frame load as a hexadecimal string without additional information. Since using such a frame representation is very cumbersome, Squish provides a way to describe the contents of selected frame types. To use it, you can create a descriptor file that can be passed to the ApplicationContext startCAN(options) function. Using the descriptor file above, the elements frames are available in the test scenario. javascriptpythonperl" />perlrubytclvar canContext = startCAN({schema: File.open(fileName,"r").read()}); var device = new CanBusDevice("socketcan", "can0"); var frame = new ThermometerFrame( ); frame.temperature = 10.1; test.log(device.hexPayload); // logs "4121999a" device.writeFrame(frame); canContext = startCAN() device = CanBusDevice("socketcan", "can0") frame = ThermometerFrame () frame.temperature = 10.1 test.log(device.hexPayload) # Logs "4121999a" device.writeFrame(frame) my $canContext = startCAN(); my $device = CanBusDevice->new("socketcan", "can0"); my $frame = ThermometerFrame->new(); $frame->temperature = 10.1; test::log($frame->hexPayload); # logs "4121999a" $device->writeFrame(frame) ; canContext = startCAN (); device = CanBusDevice.new("socketcan", "can0"); frame = ThermometerFrame.new(); frame.temperature = 10.1; Test.log(frame.hexPayload); # Logs " 4121999a" device.writeFrame(frame); set canContext [startCAN] set device [Can BusDevicesocketcan can0] set frame [ThermometerFrame new] ThermometerFrame set $frame Temperature 10.1 test log [ThermometerFrame get hexPayload $th] # Call logs "4121999a" [$device writeFrame $frame] A detailed description of the allowed field types can be found in CAN . Framework documentation. Sending CAN Frames Frames can be sent to a CAN device using the CanBusDevice.writeFrame(frame) function. However, important CAN frames are often sent at short intervals. To simulate this behavior of a specific ECU, you can create an object of the CanBusFrameRepeater class. Such an object resends copies of the specified frame while it is active. javascriptpythonperl" />perlrubytclvar canContext = startCAN({schema: File.open(fileName,"r").read()}); var device = new CanBusDevice("socketcan", "can0"); var frame = new ThermometerFrame( ); frame.temperature = 10.1; var Repeater = new CanBusFrameRepeater(device, frame); Repeater.interval = 200; // 200ms interval canContext = startCAN(open(fileName, "r").read()) device = CanBusDevice ("socketcan", "can0") frame = ThermometerFrame() frame.temperature = 10.1 Repeater = CanBusFrameRepeater(device, frame) Repeater.interval = 200 # 200ms interval my $canContext = startCAN(); my $device = CanBusDevice- >new("socketcan", "can0"); my $frame = ThermometerFrame->new(); $frame->temperature = 10.1; Repeater = CanBusFrameRepeater->new(device, frame) Repeater.interval = 200 # Interval 200 ms canContext = startCAN(); device = CanBusDevice.new("socketcan", "can0"); frame = ThermometerFrame.new(); frame.temperature = 10.1; repeater = CanBusFrameRepeater.new(device, frame); Repeater.Interwall = 200; # 200ms interval set canContext [startCAN] set device [CanBusDevice new socketcan can0] set frame [ThermometerFrame new] ThermometerFrame set $frame Temperature 10.1 set Repeater [CanBusFrameRepeater new $device $frame] CanBusFrameRepeater set interval0000 interval2 you can change interval0 $repeater # originalobject at any time during the test. Changes will be immediately reflected in the output of the repeater. javascriptpythonperl" />perlrubytcl[…] // Measured temperature change frame.temperature = 12.1; // or Repeaters.frame.temperature = 12.1; […] # Measured temperature change frame.temperature = 12 ,1 # or Repeater.frame.temperature = 12.1 […] # Measured temperature change $frame->temperature = 12.1 # or $repeater->frame->temperature = 12.1;[…] # Measured temperature change frame Temperature = 12.1; # or Repeater.frame.temperature = 12.1;[[[]] Measured temperature change ThermometerFrame set frame Temperature 12.1 // or ThermometerFrame set [CanBusFrameRepeater get $repeater frame] Temperature 12.1 Receiving frames Receiving frames is possible using the CanBusraDemetimeout function CanBusraDemetimeout function) However using this function requires the test writer to check all incoming frames in time.While it offers the most flexibility, it is not the most convenient tool in a typical test.Instead, you can use an object of the CanBusF class ra meReceiver. The object s d collects incoming frames from the CAN device and stores the history of received frames with the specified identifiers. javascriptpythonperl" />perlrubytclvar canContext = startCAN({schema: File.open(fileName,"r").read()}); var device = new CanBusDevice("socketcan", "can0"); var Receiver = new CanBusFrameReceiver(device); Receiver.setHistorySize(AirConditioningFrame.frameId, 1); delay(5); // log the last set temperature

test.log(receiver.lastFrame(AirConditioningFrame.frameId).targetTemp); canContext = startCAN() device = CanBusDevice ("socketcan", "can0") Receiver = CanBusFrameReceiver(device) Receiver.setHistorySize(AirConditioningFrame.frameId, 1) snooze(5) # Writes the last set temperature test.log(receiver.lastFrame(AirConditioningFrame.frame) my). $ canContext = startCAN(); my $device = CanBusDevice->new("socketcan", "can0"); my $receiver = CanBusFrameReceiver->new($device);one); postpone (5); // Record the last set temperature; test::log($receiver->lastFrame(Squish::AirConditioningFrame->frameId)->targetTemp); canContext = startCAN(); device = CanBusDevice.new("socketcan", "can0"); receiver = CanBusFrameReceiver.new(device); Receiver.setHistorySize(AirConditioningFrame.frameId, 1); postpone (5); # Logs the last set temperature Test.log(receiver.lastFrame(AirConditioningFrame.frameId).targetTemp) set canContext [startCAN] set device [CanBusDevice new socketcan can0] set Receiver [CanBusFrameReceiver new $device] CanBusFrameReceiver new $Recei CanBus new device] CanBusFrameReceiver frameId] 1 defer 5 # Records the last set temperature lastFrame [CanBusFrameReceiver invoke $receiver lastFrame [AirConditioningFrame get frameId]] test log [AirConditioningFrame get $lastFrame targetTemp] You can also wait for a specific frame with a specific ID and field values. javascriptpythonperl" />perlrubytcl[…] Receiver.setHistorySize(AirConditioningFrame.frameId, 1); var frame = Receiver.waitForFrame({frameId: AirConditioningFrame.frameId, targetTemp: 18}); test.log received("Await); […] Receiver.setHistorySize(AirConditioningFrame.frameId, 1) var frame = Receiver.waitForFrame({"frameId": AirConditioningFrame.frameId, "targetTemp": 18}) test.log("Received expected frame") [. ..] $receiver->setHistorySize(Squish::AirConditioningFrame->frameId, 1); my %query = (frameId => Squish::AirConditioningFrame->frameId, targetTemp => 18); var frame = $receiver->waitForFrame(%query); test::log("Expected frame received"); […] Receiver.setHistorySize(AirConditioningFrame.frameId, 1); frame = Receiver.waitForFrame(("frameId"=>Squish:: AirConditioningFrame->frameId, "targetTemp"=>18)); Test.log("Received expected frame"); […] set frameId [AirConditioningFrame get frameId] CanBusFrameReceiver call $receiver setHistorySize $frameId 1 set frame [Can BusFrameReceiver call waitForFrame(frameId $frameId targetTemp 18)] test log "Expectedreceived" The function CanBusFrameReceiver.waitForFrame(filter, timeout) searches the current history for a matching frame and, if not found, waits for a matching frame to be received. This prevents the Wait API from being called too late and not being displayed. Frame just received.