aws

Developer Guide

AWS IoT FleetWise

AWS IoT FleetWise: Developer Guide

Table of Contents

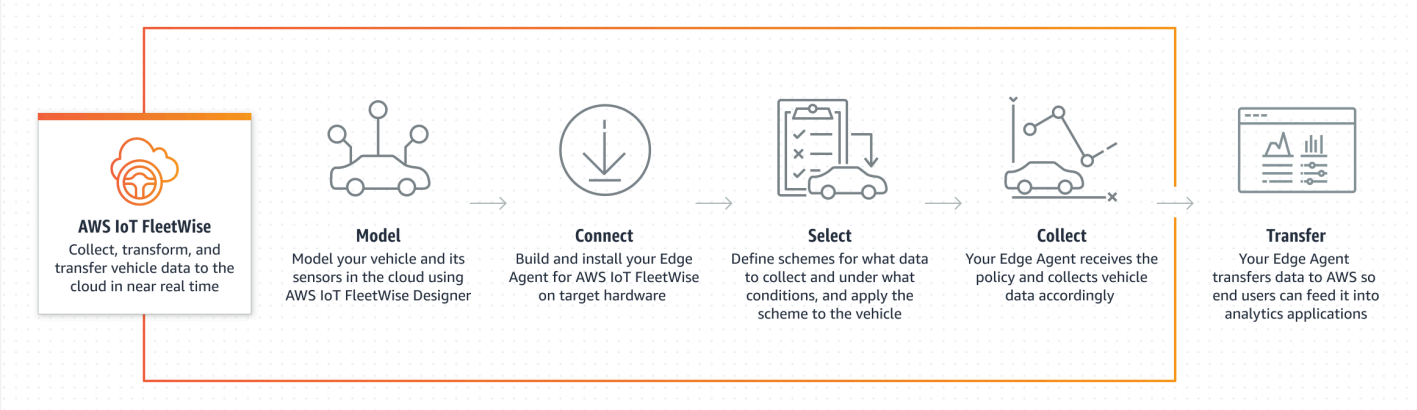
What is AWS IoT FleetWise?

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

AWS IoT FleetWise is a managed service that you can use to collect vehicle data and organize it in the cloud. You can use the collected data to improve vehicle quality, performance, and autonomy. With AWS IoT FleetWise, you can collect and organize data from vehicles that use different protocols and data formats. AWS IoT FleetWise helps to transform low-level messages into human-readable values and standardize the data format in the cloud for data analyses. You can also define data collection campaigns to control what vehicle data to collect and when to transfer that data to the cloud.

When the vehicle data is in the cloud, you can use it for applications that analyze vehicle fleet health. This data can help you to identify potential maintenance issues, make in-vehicle infotainment systems smarter, and improve advanced technologies like autonomous driving and driver-assistance systems with analytics and machine learning (ML).

The following diagram shows the basic architecture of AWS IoT FleetWise.



Topics

Benefits

The key benefits of AWS IoT FleetWise are:

Collect vehicle data more intelligently

Improve data relevance with intelligent data collection that sends only the data you need to the cloud for analysis.

Easily analyze standardized, fleet-wide data

Analyze standardized data from a fleet of vehicles without needing to develop a custom data collection or logging system.

Automatic data synchronization in the cloud

Gain a unified view of data collected from both standard sensors (telemetry data) and vision systems (data from cameras, radars, and lidars), and keep it automatically synchronized in the cloud. AWS IoT FleetWise keeps both structured and unstructured vision system data, metadata, and standard sensor data automatically synchronized in the cloud. This streamlines the process to assemble a full picture view of events and gain insights.

Store data at the Edge and forward it under optimal conditions

Reduce transmission costs by temporarily storing data on vehicles. You can forward selected data to the cloud under specified, optimal conditions--such as when vehicles connect to Wi-Fi.

> **Note**
>
> Vision system data is in preview release and is subject to change.

Use cases

The scenarios in which you can use AWS IoT FleetWise include the following:

Train AI/ML models

Continuously improve machine learning models used for autonomous and advanced driver assistance systems by collecting data from production vehicles.

Enhance the digital customer experience

Use data from infotainment systems to make in-vehicle audiovisual content and in-app insights more relevant.

Maintain vehicle fleet health

Use insights from fleet data to monitor EV battery health and charge levels, manage maintenance schedules, analyze fuel consumption, and more.

Create and manage commands

Use commands to execute commands on a vehicle from the cloud. You can remotely send commands to a vehicle, and within a few seconds, the vehicle will execute the command. For example, you can configure commands to lock a vehicle's door or set the temperature.

Create and manage state templates

State templates provide a mechanism for vehicle owners to track the state of their vehicle. The AWS IoT FleetWise Edge Agent that runs on the vehicle collects and sends signal updates to the cloud.

Important notice

Vehicle data collected through your use of AWS IoT FleetWise is intended for informational purposes only (including to help you train cloud-based artificial intelligence and machine learning models), and you may not use AWS IoT FleetWise to control or operate vehicle functions. You are solely responsible for all liability that may arise in connection with any use outside of AWS IoT FleetWise's intended purpose and in any manner contrary to applicable vehicle regulations.

Vehicle data collected through your use of AWS IoT FleetWise should be evaluated for accuracy as appropriate for your use case, including for purposes of meeting any compliance obligations

you may have under applicable vehicle safety regulations (such as safety monitoring and reporting obligations). Such evaluation should include collecting and reviewing information through other industry standard means and sources (such as reports from drivers of vehicles). You and your End Users are solely responsible for all decisions made, advice given, actions taken, and failures to take action based on your use of AWS IoT FleetWise.

Are you new to AWS IoT FleetWise?

If you're new to AWS IoT FleetWise, we recommend that you begin by reading the following sections:

- Key concepts and features of AWS IoT FleetWise
- Set up AWS IoT FleetWise
- Tutorial: Get started with AWS IoT FleetWise
- Ingest AWS IoT FleetWise data to the cloud

Accessing AWS IoT FleetWise

You can use the AWS IoT FleetWise console or API to access AWS IoT FleetWise.

Pricing for AWS IoT FleetWise

Vehicles send data to the cloud through MQTT messages. You pay at the end of each month for the vehicles that you created in AWS IoT FleetWise. You also pay for messages that you collect from vehicles. For current information about pricing, see the AWS IoT FleetWise Pricing page. To learn more about the MQTT messaging protocol, see MQTT in the *AWS IoT Core Developer Guide*.

Related services

AWS IoT FleetWise integrates with the following AWS services to improve the availability and scalability of your cloud solutions.

- **AWS IoT Core** – Register and control AWS IoT devices that upload vehicle data to AWS IoT FleetWise, and remotely send commands to a vehicle. For more information, see What is AWS IoT in the *AWS IoT Developer Guide*.

- **Amazon Timestream** – Use a time series database to store and analyze your vehicle data. For more information, see What is Amazon Timestream in the *Amazon Timestream Developer Guide*.
- **Amazon S3** – Use an object storage service to store and manage your vehicle data. For more information, see What is Amazon S3 in the *Amazon Simple Storage Service User Guide*.

Key concepts and features of AWS IoT FleetWise

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

The following sections provide an overview of AWS IoT FleetWise service components and how they interact.

After you read this introduction, see the Set up AWS IoT FleetWise section to learn how to set up AWS IoT FleetWise.

Topics

Key concepts

AWS IoT FleetWise provides a vehicle modeling framework for you to model your vehicle and its sensors and actuators in the cloud. To enable the secure communication between your vehicle and the cloud, AWS IoT FleetWise also provides a reference implementation to help you develop Edge Agent software that you can install in your vehicle. You can define data collection schemes in the cloud and deploy them to your vehicle. The Edge Agent software running in your vehicle uses data collection schemes to control what data to collect and when to transfer it to the cloud.

The following are the core concepts of AWS IoT FleetWise.

Signal

Signals are fundamental structures that you define to contain vehicle data and its metadata. A signal can be an attribute, a branch, a sensor, or an actuator. For example, you can create a

sensor to receive in-vehicle temperature values, and to store its metadata, including a sensor name, a data type, and a unit. For more information, see Manage AWS IoT FleetWise signal catalogs.

Attribute

Attributes represent static information that generally doesn't change, such as manufacturer and manufacturing date.

Branch

Branches represent signals in a nested structure. Branches demonstrate signal hierarchies. For example, the `Vehicle` branch has a child branch, `Powertrain`. The `Powertrain` branch has a child branch, `combustionEngine`. To locate the `combustionEngine` branch, use the `Vehicle.Powertrain.combustionEngine` expression.

Sensor

Sensor data reports the current state of the vehicle and change over time, as the state of the vehicle changes, such as fluid levels, temperatures, vibrations, or voltage.

Actuator

Actuator data reports the state of a vehicle device, such as motors, heaters, and door locks. Changing the state of a vehicle device can update actuator data. For example, you can define an actuator to represent the heater. The actuator receives new data when you turn on or off the heater.

Custom structure

A custom structure (also known as a struct) represents a complex or higher-order data structure. It facilitates logical binding or grouping of data that originates from the same source. A struct is used when data is read or written in an atomic operation, such as to represent a complex data type or higher-order shape.

A signal of struct type is defined in the signal catalog using a reference to a struct data type instead of a primitive data type. Structs can be used for all types of signals including sensors, attributes, actuators, and vision system data types. If a signal of struct type is sent or received, AWS IoT FleetWise expects all included items to have valid values, so all items are mandatory. For example, if a struct contains the items Vehicle.Camera.Image.height, Vehicle.Camera.Image.width, and Vehicle.Camera.Image.data – it's expected that the sent signal contains values for all of these items.

> **Note**
>
> Vision system data is in preview release and is subject to change.

Custom property

A custom property represents a member of the complex data structure. The data type of the property can be either primitive or another struct.

When representing a higher-order shape using a struct and custom property, the intended higher-order shape is always defined and visioned as a tree structure. The custom property is used to define all the leaf nodes while the struct is used to define all the non-leaf nodes.

Signal catalog

A signal catalog contains a collection of signals. Signals in a signal catalog can be used to model vehicles that use different protocols and data formats. For example, there are two cars made by different automakers: one uses the Control Area Network (CAN bus) protocol; the other one uses the On-board Diagnostics (OBD) protocol. You can define a sensor in the signal catalog to receive in-vehicle temperature values. This sensor can be used to represent the thermocouples in both cars. For more information, see Manage AWS IoT FleetWise signal catalogs.

Vehicle model (model manifest)

Vehicle models are declarative structures that you can use to standardize the format of your vehicles and to define relationships between signals in the vehicles. Vehicle models enforce consistent information across multiple vehicles of the same type. You add signals to create vehicle models. For more information, see Manage AWS IoT FleetWise vehicle models.

Decoder manifest

Decoder manifests contain decoding information for each signal in vehicle models. Sensors and actuators in vehicles transmit low-level messages (binary data). With decoder manifests, AWS IoT FleetWise is able to transform binary data into human-readable values. Every decoder manifest is associated with a vehicle model. For more information, see Manage AWS IoT FleetWise decoder manifests.

Network interface

Contains information about the protocol that the in-vehicle network uses. AWS IoT FleetWise supports the following protocols.

Controller Area Network (CAN bus)

A protocol that defines how data is communicated between electronic control units (ECUs). ECUs can be the engine control unit, airbags, or the audio system.

On-board diagnostic (OBD) II

A further developed protocol that defines how self-diagnostic data is communicated between ECUs. It provides a number of standard diagnostic trouble codes (DTCs) that help identify what is wrong with your vehicle.

Vehicle middleware

The vehicle middleware defined as a type of network interface. Examples of vehicle middleware include Robot Operating System (ROS 2) and Scalable service-Oriented MiddlewarE over IP (SOME/IP).

> **Note**
>
> AWS IoT FleetWise supports ROS 2 middleware for vision system data.

Custom interfaces

You can also use your own interface to decode signals at the Edge. This can save you time since you don't need to create decoding rules in the cloud.

Signal decoder

Provides detailed decoding information for a specific signal. Every signal specified in the vehicle model must be paired with a signal decoder. If the decoder manifest contains CAN network interfaces, it must contain CAN decoder signals. If the decoder manifest contains OBD network interfaces, it must contain OBD signal decoders.

The decoder manifest must contain message signal decoders if it also contains vehicle middleware interfaces. Or, if the decoder manifest contains custom decoding interfaces, it must also contain custom decoding signals.

Vehicle

A virtual representation of your physical vehicle, such a car or a truck. Vehicles are instances of vehicle models. Vehicles created from the same vehicle model inherit the same group of signals. Each vehicle corresponds to an AWS IoT thing.

Fleet

A fleet represents a group of vehicles. Before you can easily manage a fleet of vehicles, you must associate individual vehicles to a fleet.

Campaign

Contains data collection schemes. You define a campaign in the cloud and deploy it to a vehicle or fleet. Campaigns give the Edge Agent software instructions on how to select, collect, and transfer data to the cloud.

Data partition

Configure partitioned data in a campaign to temporarily store signal data. You configure when and how to forward the data to the cloud.

Data collection scheme

Data collection schemes give the Edge Agent software instructions on how to collect data. Currently, AWS IoT FleetWise supports the condition-based collection scheme and the time-based collection scheme.

Condition-based collection scheme

Use a logical expression to recognize what data to collect. The Edge Agent software collects data when the condition is met. For example, if the expression is `$variable.myVehicle.InVehicleTemperature >35.0`, the Edge Agent software collects temperature values that are greater than 35.0.

Time-based collection scheme

Specify a time period in milliseconds to define how often to collect data. For example, if the time period is 10,000 milliseconds, the Edge Agent software collects data once every 10 seconds.

Commands

Commands execute commands on a vehicle from the cloud. You can remotely send commands to a vehicle, and within a few seconds, the vehicle will execute the command. For example, you can configure commands to lock a vehicle's door or set the temperature.

The command is a resource that's managed by AWS IoT Device Management. It contains reusable configurations that are applied when sending a command execution to the vehicle. For more information, see AWS IoT commands in the *AWS IoT Core Developer Guide*.

State templates

State templates provide a mechanism for vehicle owners to track the state of their vehicle. The Edge Agent software Agent that runs on the vehicle collects and sends signal updates to the cloud. Each state template contains a list of signals from which data is collected.

Features of AWS IoT FleetWise

The following are the key features of AWS IoT FleetWise.

Vehicle modeling

Build virtual representations of your vehicles and apply a common format to organize vehicle signals. AWS IoT FleetWise supports Vehicle Signal Specification (VSS) that you can use to standardize vehicle signals.

Scheme-based data collection

Define schemes to transfer only high-value vehicle data to the cloud. You can define condition-based schemes to control what data to collect, such as data in-vehicle temperature values that are greater than 40 degrees. You can also define time-based schemes to control how often to collect data.

Edge Agent for AWS IoT FleetWise software

The Edge Agent software running in vehicles facilitates communication between vehicles and the cloud. While vehicles are connected to the cloud, the Edge Agent software continually receives data collection schemes and collects data accordingly.

AWS Region and feature availability in AWS IoT FleetWise

For a list of AWS Regions that support AWS IoT FleetWise, see AWS IoT FleetWise endpoints and quotas. AWS IoT FleetWise features differ in their regional support.

> **Note**
>
> Access to the Asia Pacific (Mumbai) Region and some AWS IoT FleetWise features are currently gated. To request access to this AWS Region and all gated features, contact your account manager or the AWS Support Center.

The following table shows feature support by Region:

Features/Regions	US East (N. Virginia)	Europe (Frankfurt)	Asia Pacific (Mumbai) NOTE: Gated access only
Signal catalogs	Yes	Yes	Gated
Vehicle models	Yes	Yes	Gated
Decoder manifests	Yes	Yes	Gated
Vehicles	Yes	Yes	Gated
Fleets	Yes	Yes	Gated
Campaigns	Yes	Yes	Gated
Vision system data (in preview release)	Yes	Yes	Gated
MQTT topic as a campaign data destination	Gated	Gated	Gated
Store and forward	Gated	Gated	Gated
Commands	Gated	Gated	Gated
Last known state	Gated	Gated	Gated
Network agnostic data collection using a custom decoding interface	Gated	Gated	Gated
Diagnostic trouble code (DTC) fetching*	Gated	Gated	Gated

*DTC fetching offers a range of capabilities that go beyond basic DTC data retrieval. This functionality includes custom features that enable you to define functions at the edge and invoke them by name within condition-based campaign expressions. Additionally, it supports the collection of unbounded strings, providing flexible string data type handling. The Edge Agent can fetch data either on a periodic basis or triggered by specific conditions, enhancing its adaptability and efficiency in data collection processes. For more information, see the custom function guide and the DTC data collection reference implementation in the *Edge Agent Developer Guide*.

Set up AWS IoT FleetWise

Before you use AWS IoT FleetWise for the first time, complete the steps in the following sections.

Topics

Set up your AWS account

Complete the following tasks to sign up for AWS and create an administrative user.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

> **Note**
>
> You can use a service-linked role with AWS IoT FleetWise. Service-linked roles are predefined by AWS IoT FleetWise and include the permissions that AWS IoT FleetWise needs to send metrics to Amazon CloudWatch. For more information, see Using service-linked roles for AWS IoT FleetWise.

Get started in the console

If you aren't already signed in to your AWS account, sign in, then open the AWS IoT FleetWise console. To get started with AWS IoT FleetWise, create a vehicle model. A vehicle model standardizes the format of your vehicles.

1. Open the AWS IoT FleetWise console.
2. In **Get started with AWS IoT FleetWise**, choose **Get started**.

For more information about creating a vehicle model, see Create an AWS IoT FleetWise vehicle model.

Configure your AWS IoT FleetWise settings

You can use the AWS IoT FleetWise console or API to configure settings for Amazon CloudWatch Logs metrics, Amazon CloudWatch Logs, and encrypt data with an AWS managed key.

With CloudWatch metrics, you can monitor AWS IoT FleetWise and other AWS resources. You can use CloudWatch metrics to collect and track metrics, such as to determine if there is an exceeded

service limit. For more information about CloudWatch metrics, see Monitor AWS IoT FleetWise with Amazon CloudWatch.

With CloudWatch Logs, AWS IoT FleetWise sends log data to a CloudWatch log group, where you can use it to identify and mitigate any issues. For more information about CloudWatch Logs, see Configure AWS IoT FleetWise logging.

With data encryption, AWS IoT FleetWise uses AWS managed keys to encrypt data. You can also choose to create and manage keys with AWS KMS. For more information about encryption, see Data encryption in AWS IoT FleetWise.

Configure settings (console)

If you aren't already signed in to your AWS account, sign in, then open the AWS IoT FleetWise console.

1. Open the AWS IoT FleetWise console.
2. On the left pane, choose **Settings**.
3. In **Metrics**, choose **Enable**. AWS IoT FleetWise automatically attaches a CloudWatch managed policy to the service-linked role and enables CloudWatch metrics.
4. In **Logging**, choose **Edit**.
    a. In the **CloudWatch logging** section, enter the **Log group**.
    b. To save your changes, choose **Submit**.
5. In the **Encryption** section, choose **Edit**.
    a. Choose the type of key that you want to use. For more information, see Key management in AWS IoT FleetWise.
        i. **Use AWS key** – AWS IoT FleetWise owns and manages the key.
        ii. **Choose a different AWS Key Management Service key** – You manage AWS KMS keys that are in your account.
    b. To save your changes, choose **Submit**.

Configure settings (AWS CLI)

In the AWS CLI, register the account to configure settings.

IAM permission setup for account registration

To invoke the `RegisterAccount` API successfully, you need to include `iam:CreateServiceLinkedRole` in your IAM policy document. This API creates a service-linked role in your account that is used to publish AWS IoT FleetWise metrics to your CloudWatch. To verify whether the account is registered successfully, invoke the `GetRegisterAccountStatus` API and make sure the registration status is `REGISTRATION_SUCCESS`.

The following example shows a sample policy document for setting up permissions to `RegisterAccount` and `GetRegisterAccountStatus`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotfleetwise:RegisterAccount",
        "iotfleetwise:GetRegisterAccountStatus",
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

1. To configure settings, run the following command.

   ```
   aws iotfleetwise register-account
   ```

2. To verify your settings, run the following command to retrieve the registration status.

> **Note**
>
> The service-linked role is only used to publish AWS IoT FleetWise metrics to CloudWatch. For more information, see Using service-linked roles for AWS IoT FleetWise.

```
aws iotfleetwise get-register-account-status
```

Example response

```
{
    "accountStatus": "REGISTRATION_SUCCESS",
    "creationTime": "2022-07-28T11:31:22.603000-07:00",
    "customerAccountId": "012345678912",
    "iamRegistrationResponse": {
        "errorMessage": "",
        "registrationStatus": "REGISTRATION_SUCCESS",
        "roleArn": "arn:aws:iam::012345678912:role/AWSIoTFleetwiseServiceRole"
    },
    "lastModificationTime": "2022-07-28T11:31:22.854000-07:00",
    }
}
```

The registration status can be one of the following:

- `REGISTRATION_SUCCESS` – The AWS resource is successfully registered.
- `REGISTRATION_PENDING` – AWS IoT FleetWise is processing the registration request. This process takes approximately five minutes to complete.
- `REGISTRATION_FAILURE` – AWS IoT FleetWise can't register the AWS resource. Try again later.

Making requests to AWS IoT FleetWise using IPv6

You can communicate with AWS IoT FleetWise over Internet Protocol version 6 (IPv6) and IPv4 to manage your resources. Dual-stack endpoints support requests to AWS IoT FleetWise APIs over IPv6 and IPv4. There are no additional charges for communication over IPv6.

The IPv6 protocol is the next generation IP standard with additional security features. It offers 128-bit long address space while IPv4 has 32-bit long address. IPv4 can generate 4.29 x 10^9 addresses while IPv6 can have 3.4 x 10^38 addresses.

IPv6 prerequisites for control plane endpoints

IPv6 protocol support is automatically enabled for control plane endpoints. When using the endpoints for control plane clients, you must provide the Server Name Indication (SNI) extension. Clients can use the SNI extension to indicate the name of the server being contacted, and whether it's using the regular endpoints or the dual-stack endpoints. See Using dual-stack endpoints.

IPv6 support for AWS PrivateLink endpoints

AWS IoT FleetWise supports IPv6 communication to interface VPC endpoints using AWS PrivateLink.

Testing IPv6 address compatibility

If you're using use Linux/Unix or Mac OS X, you can test whether you can access a dual-stack endpoint over IPv6 by using the curl command as shown in the following example:

```
curl -v https://iotfleetwise.<us-east-1>.api.aws
```

You get back information similar to the following example. If you're connected over IPv6, the connected IP address will be an IPv6 address.

```
* Host iotfleetwise.us-east-1.api.aws:443 was resolved.
* IPv6: ::ffff:3.82.78.135, ::ffff:54.211.220.216, ::ffff:54.211.201.157
* IPv4: (none)
*   Trying [::ffff:3.82.78.135]:443...
* Connected to iotfleetwise.us-east-1.api.aws (::ffff:3.82.78.135) port 443
* ALPN: curl offers h2,http/1.1
```

If you're using Microsoft Windows 7 or Windows 10, you can test whether you can access a dual-stack endpoint over IPv6 or IPv4 by using the ping command as shown in the following example.

```
ping iotfleetwise.<us-east-1>.api.aws
```

Using IPv6 addresses in IAM policies

Before you use IPv6 for your resources, you must ensure that any IAM polices that are used for IP address filtering include IPv6 address ranges. For more information about managing access permissions with IAM, see Identity and Access Management for AWS IoT FleetWise.

IAM policies that filter IP addresses use IP Address Condition Operators. The following policy identifies the `54.240.143.*` range of allowed IPv4 addresses by using IP address condition operators. Since all IPv6 addresses are outside the allowed range, this policy prevents communication using IPv6 addresses.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "iotfleetwise:*",
      "Resource": "arn:aws:iotfleetwise:us-east-1:111122223333:*",
      "Condition": {
         "IpAddress": {"aws:SourceIp": "54.240.143.0/24"}
      }
    }
  ]
}
```

To include IPv6 addresses, you can modify the policy's Condition element to allow both IPv4 (54.240.143.0/24) and IPv6 (2001:DB8:1234:5678::/64) address ranges as shown in the following example.

```
"Condition": {
  "IpAddress": {
    "aws:SourceIp": [
      "54.240.143.0/24",
      "2001:DB8:1234:5678::/64"
    ]
  }
}
```

Using dual-stack endpoints

AWS IoT FleetWise dual-stack endpoints support requests to AWS IoT FleetWise APIs over IPv6 and IPv4. When you make a request to a dual-stack endpoint, it automatically resolves to an IPv4 or an IPv6 address. In the dual-stack mode, both IPv4 and IPv6 client connections are accepted.

If you're using the REST API, you can directly access an AWS IoT FleetWise endpoint by using the endpoint name (URI). AWS IoT FleetWise supports only regional dual-stack endpoint names, which means that you must specify the AWS Region as part of the name.

The following table shows the format of control plane endpoints for AWS IoT FleetWise when using IPv4 and the dual-stack modes. For more information about these endpoints, see AWS IoT FleetWise endpoints.

Endpoint	IPv4 address	Dual-stack mode
Control plane	iotfleetwise.*<region>*.amazonaws.com	iotfleetwise.*<region>*.api.aws

When using the AWS CLI and AWS SDKs, you can use a `AWS_USE_DUALSTACK_ENDPOINT` environment variable, or the `use_dualstack_endpoint` parameter, which is a shared config file setting, to change to a dual-stack endpoint. You can also specify the dual-stack endpoint directly as an override of the AWS IoT FleetWise endpoint in the config file. For more information, see Dual-stack and FIPS endpoints.

When you use the AWS CLI, you can set the configuration value `use_dualstack_endpoint` as `true` in a profile in your AWS Config file. This will direct all AWS IoT FleetWise requests made by the commands to the dual-stack endpoint for the specified region. You specify the region in the config file or in a command using the `--region` option.

```
$ aws configure set default.iotfleetwise.use_dualstack_endpoint true
```

Instead of using the dual-stack endpoints for all commands, to use these endpoints for specific commands:

- You can use the dual-stack endpoint for specific commands by setting the `--endpoint-url` parameter for those commands. For example, in the following command, you can replace the *<endpoint-url>* to `iotfleetwise.`*<region>*`.api.aws`.

```
aws iotfleetwise list-fleets \
  --endpoint-url <endpoint-url>
```

- You can set up separate profiles in your AWS Config file. For example, create one profile that sets `use_dualstack_endpoint` to true, and a profile that does not set `use_dualstack_endpoint`. When you run a command, specify which profile you want to use, depending upon whether or not you want to use the dual-stack endpoint.

Tutorial: Get started with AWS IoT FleetWise

With AWS IoT FleetWise, you can collect, transform, and transfer your vehicle data. Use the tutorial in this section to get started with AWS IoT FleetWise.

See the following topics to learn more about AWS IoT FleetWise:

- Ingest AWS IoT FleetWise data to the cloud
- Model AWS IoT FleetWise vehicles
- Manage AWS IoT FleetWise vehicles
- Manage fleets in AWS IoT FleetWise
- Collect AWS IoT FleetWise data with campaigns

Introduction

Use AWS IoT FleetWise to collect, transform, and transfer the unique data format from automated vehicles to the cloud in near real time. You have access to fleet-wide insights. This can help you to efficiently detect and mitigate issues in vehicle health, transfer high-value data signals, and remotely diagnose problems, all while reducing costs.

This tutorial shows you how to get started with AWS IoT FleetWise. You'll learn how to create a vehicle model (model manifest), a decoder manifest, a vehicle, and a campaign.

For more information about the key components and concepts of AWS IoT FleetWise, see Key concepts and features of AWS IoT FleetWise.

Estimated time: About 45 minutes.

> **Important**
>
> You will be charged for the AWS IoT FleetWise resources that this demo creates and consumes. For more information, see AWS IoT FleetWise in the *AWS IoT FleetWise Pricing* page.

Prerequisites

To complete this getting started tutorial, you first need the following:

- An AWS account. If you don't have an AWS account, see Creating an AWS account in the *AWS Account Management Reference Guide*.
- Access to an AWS Region that supports AWS IoT FleetWise. Currently, AWS IoT FleetWise is supported in US East (N. Virginia) and Europe (Frankfurt). You can use the Region selector in the AWS Management Console to switch to one of these Regions. For more information, see AWS IoT FleetWise endpoints and quotas.
- Amazon Timestream resources:
  - An Amazon Timestream database. For more information, see Create a database in the *Amazon Timestream Developer Guide*.
  - An Amazon Timestream table created in Amazon Timestream that will hold your data. For more information, see Create a table in the *Amazon Timestream Developer Guide*.
- The Edge Agent software demo. (Instructions for setting up the demo are in the next step.)
  - You can use the Explore Edge Agent quick start demo to explore AWS IoT FleetWise and learn how to develop Edge Agent software for AWS IoT FleetWise. This demo uses an CloudFormation template. It walks you through reviewing the Edge Agent reference implementation, developing your Edge Agent, and then deploying your Edge Agent software on an Amazon EC2 Graviton and generating sample vehicle data. The demo also provides a script that you can use to create a signal catalog, a vehicle model, a decoder manifest, a vehicle, a fleet, and a campaign — all in the cloud.
  - To download the demo, navigate to the AWS IoT FleetWise console. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.

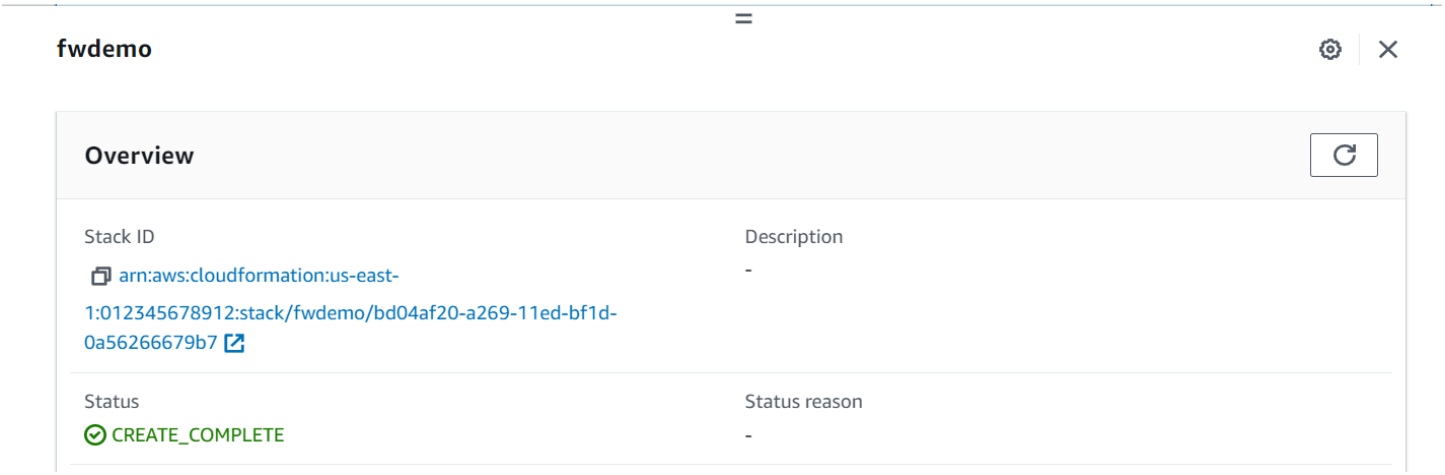
Step 1: Set up the Edge Agent software for AWS IoT FleetWise

> **Note**
>
> The CloudFormation stack in this step uses telemetry data. You can also create a CloudFormation stack using vision system data. For more information, see the *Vision System Data Developer Guide*.
> Vision system data is in preview release and is subject to change.

Your Edge Agent software for AWS IoT FleetWise facilitates communication between vehicles and the cloud. It receives instructions from data collection schemes on how to collect data from cloud-connected vehicles.

To set up your Edge Agent software, in **General information**, do the following:

1. Open the Launch CloudFormation Template.
2. On the **Quick create stack** page, for **Stack name**, enter the name of your stack of AWS IoT FleetWise resources. A stack is a friendly name that appears as a prefix on the names of the resources this CloudFormation template creates.
3. Under **Parameters**, enter your custom values for the parameters related to your stack.
   a. **Fleetsize** - You can increase the number of vehicles in your fleet by updating the Fleetsize parameter.
   b. **IoTCoreRegion** - You can specify the Region where the AWS IoT thing is created by updating the IoTCoreRegion parameter. You must use the same Region that you used to create your AWS IoT FleetWise vehicles. For more information about AWS Regions, see Regions and Zones - Amazon Elastic Compute Cloud.
4. In the **Capabilities** section, select the box to acknowledge that CloudFormation creates IAM resources.
5. Choose **Create stack**, then wait approximately 15 minutes for the status of the stack to display CREATE_COMPLETE.
6. To confirm the stack was created, choose the **Stack info** tab, refresh the view, and look for CREATE_COMPLETE.



> **Important**
>
> You will be charged for the AWS IoT FleetWise resources that this demo creates and consumes. For more information, see AWS IoT FleetWise in the *AWS IoT FleetWise Pricing* page.

Step 2: Create a vehicle model

> **Important**
>
> You can't create a vehicle model with vision system data signals in the AWS IoT FleetWise console. Instead, use the AWS CLI.

You use vehicle models to standardize the format of your vehicles, and to help define the relationship between signals in the vehicles that you create. A *signal catalog* is also created when you create a vehicle model. A signal catalog is a collection of standardized signals that can be reused to create vehicle models. Signals are fundamental structures that you define to contain vehicle data and its metadata. At this time, the AWS IoT FleetWise service supports only one signal catalog per AWS Region per account. This helps to verify that data processed from a fleet of vehicles is consistent.

To create a vehicle model

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models.**
3. On the **Vehicle models** page, choose **Create vehicle model**.
4. In the **General information** section, enter the name of your vehicle model, such as Vehicle1, and an optional description. Then choose **Next**.
5. Choose one or more signals from the signal catalog. You can filter signals by name in the search catalog, or choose them from the list. For example, you can choose signals for tire pressure and brake pressure so that you can collect data related to these signals. Choose **Next**.
6. Choose your .dbc files and upload them from your local device. Choose **Next**.

> **Note**
>
> For this tutorial, you can download a sample .dbc file to upload for this step.

7. Add attributes to your vehicle model and then choose **Next**.

   a. **Name** - Enter the name of the vehicle attribute, such as the manufacturer name or manufacturing date.

   b. **Data Type** - On the **Data type** menu, choose a data type.

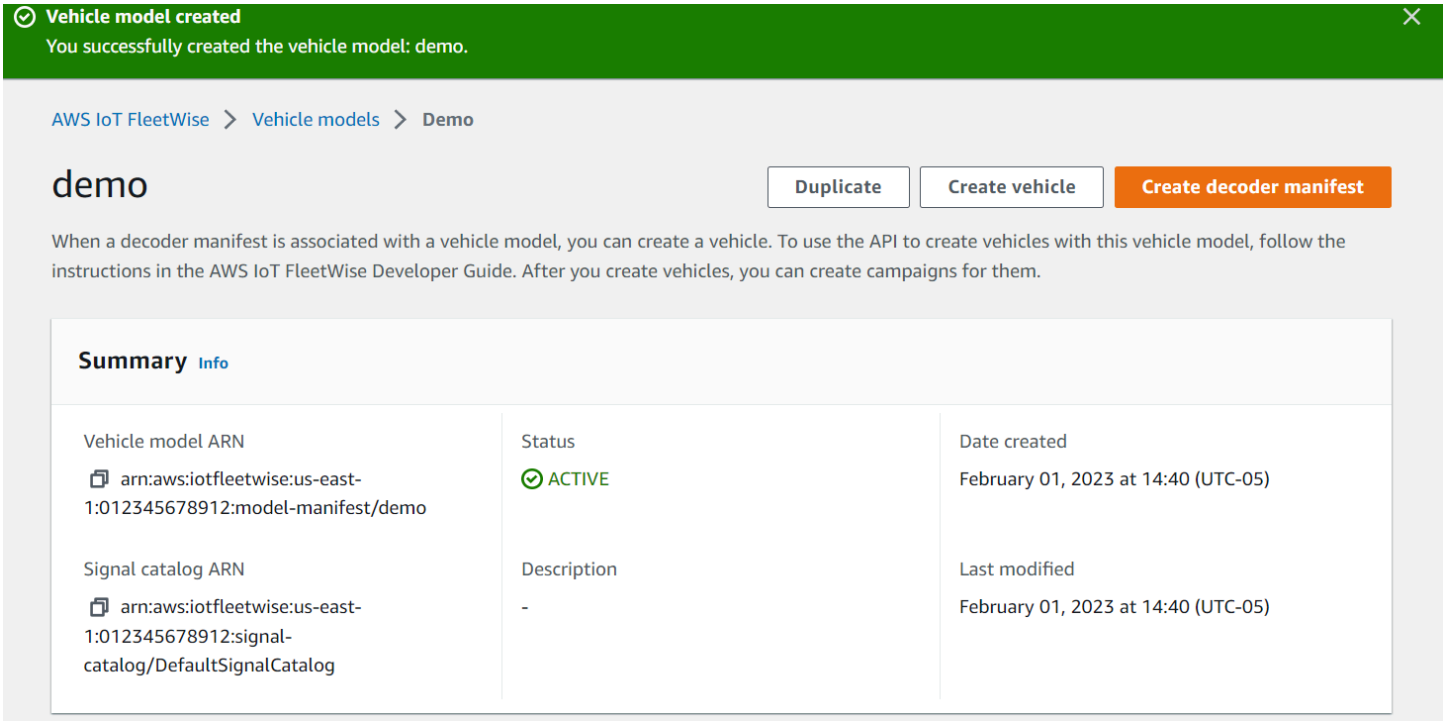
   c. **Unit** - (Optional) Enter a unit value, such as kilometer or Celsius.

   d. **Path** - (Optional) Enter a name for the path to a signal, such as `Vehicle.Engine.Light.` The dot (.) indicates that it is a child signal.

   e. **Default value** - (Optional) Enter a default value.

   f. **Description** - (Optional) Enter a description of the attribute.

8. Review your configurations. When you're ready, choose **Create**. A notification appears saying your vehicle model was successfully created.



Step 3: Create a decoder manifest

Decoder manifests are associated with the vehicle models that you create. They contain information that helps AWS IoT FleetWise decode and transform vehicle data from a binary format into human-readable values that can be analyzed. Network interfaces and decoder signals are components that help configure decoder manifests. A network interface contains information about the CAN or OBD protocol that your vehicle network uses. The decoder signal provides decoding information for a specific signal.

To create a decoder manifest

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. In the **Vehicle models** section, choose the vehicle model that you want to use to create a decoder manifest.
4. Choose **Create decoder manifest**.

Step 4: Configure a decoder manifest

To configure a decoder manifest

> **Important**
>
> You can't configure vision system data signals in decoder manifests using the AWS IoT FleetWise console. Instead, use the AWS CLI. For more information, see Create a decoder manifest (AWS CLI).

1. To help you identify your decoder manifest, enter a name and an optional description for it. Then, choose **Next**.
2. To add one or more network interfaces, choose either the CAN_INTERFACE or the OBD_INTERFACE type.

   - **On-board diagnostic (OBD) interface -** Choose this interface type if you want a protocol that defines how self-diagnostic data is communicated between electronic control units (ECUs). This protocol provides a number of standard diagnostic trouble codes (DTCs) that can help you troubleshoot problems with your vehicle.

- **Controller Area Network (CAN bus) interface -**Choose this interface type if you want a protocol that defines how data is communicated between ECUs. ECUs can be engine control units, airbags, or the audio system.

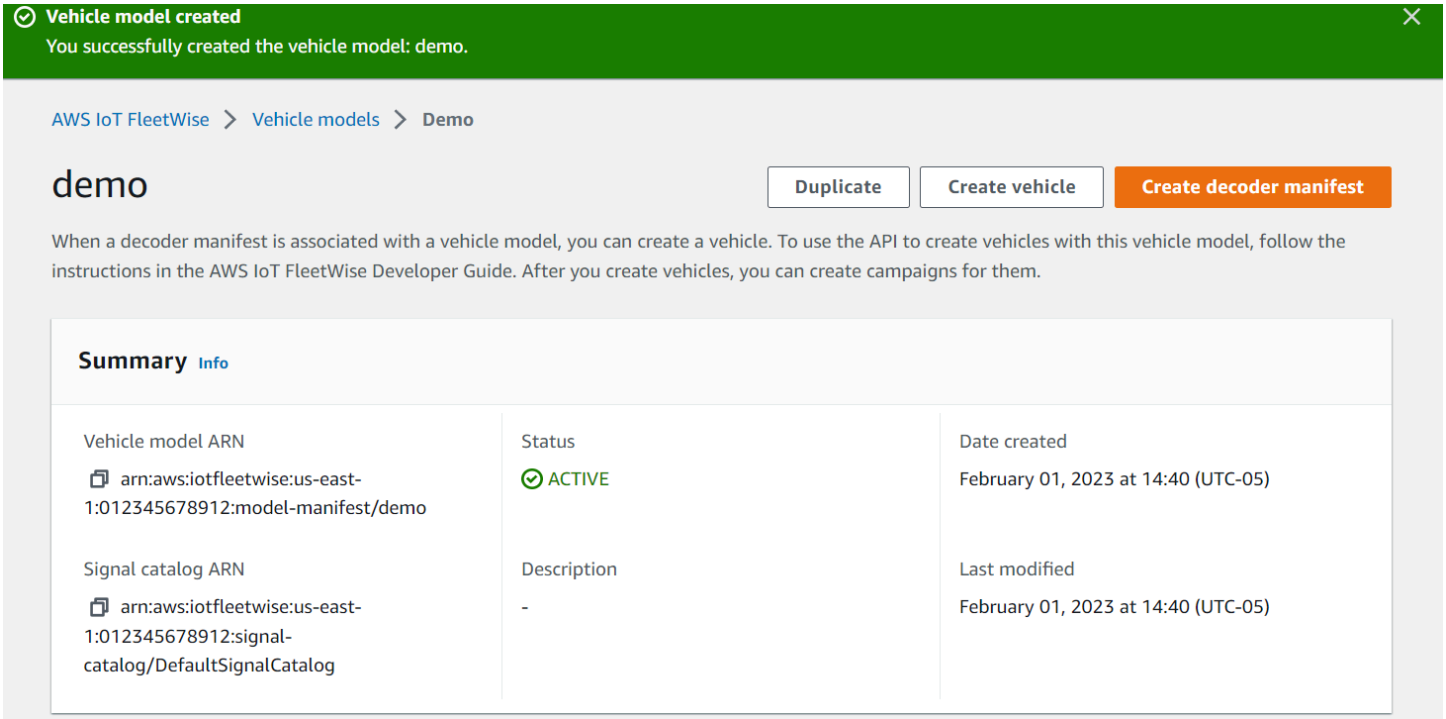
3. Enter a network interface name.
4. To add signals to the network interface, choose one or more signals from the list.
5. Choose a decoder signal for the signal you added in the previous step. To provide decoding information, upload a .dbc file. Each signal in the vehicle model must be paired with a decoder signal that you can choose from the list.
6. To add another network interface, choose **Add network interface**. When you're done adding network interfaces, choose **Next**.
7. Review your configurations and then choose **Create**. A notification appears saying your decoder manifest was successfully created.

Step 5: Create a vehicle

In AWS IoT FleetWise, vehicles are virtual representations of your real-life, physical vehicle. All vehicles created from the same vehicle model inherit the same group of signals, and each vehicle that you create corresponds to a newly created IoT thing. You must associate all vehicles with a decoder manifest.

Prerequisites

1. Verify that you've already created the vehicle model and decoder manifest. Also, verify that the status of the vehicle model is **ACTIVE**.
   a. To verify that the status of the vehicle model is ACTIVE, open the AWS IoT FleetWise console.
   b. On the navigation pane, choose **Vehicle models**.
   c. In the **Summary** section, under **Status**, check the status of your vehicle.



To create a vehicle

1. Open the AWS FleetWise console.
2. On the navigation pane, choose **Vehicles**.
3. Choose **Create vehicle**.
4. To define the vehicle properties, enter the vehicle name, and then choose a model manifest (vehicle model) and a decoder manifest.
5. (Optional) To define the vehicle attributes, enter a key-value pair and then choose **Add attributes**.
6. (Optional) To label your AWS resource, add tags and then choose **Add new tag.**
7. Choose **Next.**
8. To configure the vehicle certificate, you can either upload your own certificate or choose **Auto-generate a new certificate.** We recommend auto-generating your certificate for a quicker setup. If you already have a certificate, you can choose to use it instead.
9. Download the public and private key files and then choose **Next.**
10. To attach a policy to the vehicle certificate, you can either enter an existing policy name or create a new policy. To create a new policy, choose **Create policy** and then choose **Next**.
11. Review your configurations. When you're done, choose **Create vehicle.**

Step 6: Create a campaign

In AWS IoT FleetWise, campaigns are used to facilitate the selection, collection, and transfer of data from vehicles to the cloud. Campaigns contain data collection schemes that give the Edge Agent software instructions on how to collect data with a condition-based collection scheme or a time-based collection scheme.

To create a campaign

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Campaigns**.
3. Choose **Create campaign**.
4. Enter your campaign name and an optional description.
5. To configure your campaign's data collection scheme, you can manually define the data collection scheme or upload a .json file from your local device. Uploading a .json file automatically defines the data collection scheme.

    a. To manually define the data collection scheme, choose **Define Data Collection Scheme** and choose the type of data collection scheme you want to use for your campaign. You can choose either a **Condition-based** collection scheme or **Time-based** collection scheme.

    b. If you choose a **Time-based** collection scheme, you must specify the duration of time that your campaign will collect the vehicle data.

    c. If you choose a condition-based collection scheme, you must specify an expression to recognize what data to collect. Be sure to specify the signal's name as a variable, a comparison operator, and a comparison value.

    d. (Optional) Choose the language version of your expression, or keep it as the default value of 1.

    e. (Optional) Specify the trigger interval between two data collection events.

    f. To collect data, choose the **Trigger** mode condition for the Edge Agent software. By default, the Edge Agent for AWS IoT FleetWise software **Always** collects data whenever the condition is met. Or, it can collect data only when the condition is met for the first time, **On first trigger**.

    g. (Optional) You can choose more advanced scheme options.
6. To specify the signals that the data collection scheme will collect data from, search for the name of the signal from the menu.

7. (Optional) You can choose a maximum sample count or minimum sampling interval. You can also add more signals.
8. Choose **Next**.
9. Define the storage destination that you want the campaign to transfer data to. You can store data in Amazon S3 or Amazon Timestream.

    a. Amazon S3 – Choose the S3 bucket that AWS IoT FleetWise has permissions to.

    b. Amazon Timestream – choose the Timestream database and table name. Enter an IAM role that allows AWS IoT FleetWise to send data to Timestream.
10. Choose **Next**.
11. Choose vehicle attributes or vehicle names from the search box.
12. Enter the value related to the attribute or name that you chose for your vehicle.
13. Choose the vehicles that your campaign will collect data from. Then, choose **Next**.
14. Review the configurations of your campaign and then choose **Create campaign**. You or your team must deploy the campaign to vehicles.

Step 7: Clean up

To avoid further charges for the resources you used during this tutorial, delete the CloudFormation stack and all stack resources.

To delete the CloudFormation stack

1. Open the CloudFormation console.
2. From the list of **Stacks**, choose the stack that you created in step 1.
3. Choose **Delete**.
4. To confirm deletion, choose **Delete**. The stack takes around 15 minutes to delete.

Next steps

1. You can process and visualize the vehicle data that your campaign collects. For more information, see Visualize AWS IoT FleetWise vehicle data.
2. You can troubleshoot and resolve issues with AWS IoT FleetWise. For more information, see Troubleshooting AWS IoT FleetWise.

Ingest AWS IoT FleetWise data to the cloud

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

The Edge Agent for AWS IoT FleetWise software, when installed and running in vehicles, is designed to facilitate secure communication between your vehicles and the cloud.

Note

- AWS IoT FleetWise is not intended for use in, or in association with, the operation of any hazardous environments or critical systems that may lead to serious bodily injury or death or cause environmental or property damage. Vehicle data collected through your use of AWS IoT FleetWise is for informational purposes only, and you may not use AWS IoT FleetWise to control or operate vehicle functions.
- Vehicle data collected through your use of AWS IoT FleetWise should be evaluated for accuracy as appropriate for your use case, including for purposes of meeting any compliance obligations you may have under applicable vehicle safety regulations (such as safety monitoring and reporting obligations). Such evaluation should include collecting and reviewing information through other industry standard means and sources (such as reports from drivers of vehicles).

To ingest data to the cloud, do the following:

1. Develop and install your Edge Agent for AWS IoT FleetWise software in your vehicle. For more information about how to work with the Edge Agent software, do the following to download the *Edge Agent for AWS IoT FleetWise software Developer Guide*.
    1. Navigate to the AWS IoT FleetWise console.
    2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.

2. Create or import a signal catalog containing signals that you'll use to create a vehicle model. For more information, see Create an AWS IoT FleetWise signal catalog and Import a signal catalog (AWS CLI).

   Note

   - If you use the AWS IoT FleetWise console to create the first vehicle model, you don't need to manually create a signal catalog. When you create your first vehicle model, AWS IoT FleetWise automatically creates a signal catalog for you. For more information, see Create an AWS IoT FleetWise vehicle model.
   - AWS IoT FleetWise currently supports a signal catalog for each AWS account per AWS Region.

3. Use signals in the signal catalog to create a vehicle model. For more information, see Create an AWS IoT FleetWise vehicle model.

   Note

   - If you use the AWS IoT FleetWise console to create a vehicle model, you can upload .dbc files to import signals. .dbc is a file format that Controller Area Network (CAN bus) databases support. After the vehicle model is created, new signals are automatically added to the signal catalog. For more information, see Create an AWS IoT FleetWise vehicle model.
   - If you use the `CreateModelManifest` API operation to create a vehicle model, you must use the `UpdateModelManifest` API operation to activate the vehicle model. For more information, see Update an AWS IoT FleetWise vehicle model.
   - If you use the AWS IoT FleetWise console to create a vehicle model, AWS IoT FleetWise automatically activates the vehicle model for you.

4. Create a decoder manifest. The decoder manifest contains decoding information for every signal specified in the vehicle model that you created in the previous step. The decoder manifest is associated with the vehicle model that you created. For more information, see Manage AWS IoT FleetWise decoder manifests.

> **Note**
>
> - If you use the `CreateDecoderManifest` API operation to create a decoder manifest, you must use the `UpdateDecoderManifest` API operation to activate the decoder manifest. For more information, see Update an AWS IoT FleetWise decoder manifest.
> - If you use the AWS IoT FleetWise console to create a decoder manifest, AWS IoT FleetWise automatically activates the decoder manifest for you.

5. Create vehicles from the vehicle model. Vehicles created from the same vehicle model inherit the same group of signals. You must use AWS IoT Core to provision your vehicle before you can ingest data to the cloud. For more information, see Manage AWS IoT FleetWise vehicles.
6. (Optional) Create a fleet to represent a group of vehicles, and then associate individual vehicles with the fleet. This helps you manage multiple vehicles at the same time. For more information, see Manage fleets in AWS IoT FleetWise.
7. (Optional) Create campaigns. Campaigns are deployed to a vehicle or a fleet of vehicles. Campaigns give the Edge Agent software instructions on how to select, collect, and transfer data to the cloud. For more information, see Collect AWS IoT FleetWise data with campaigns. You can create campaigns, state templates (below), or both to collect data.

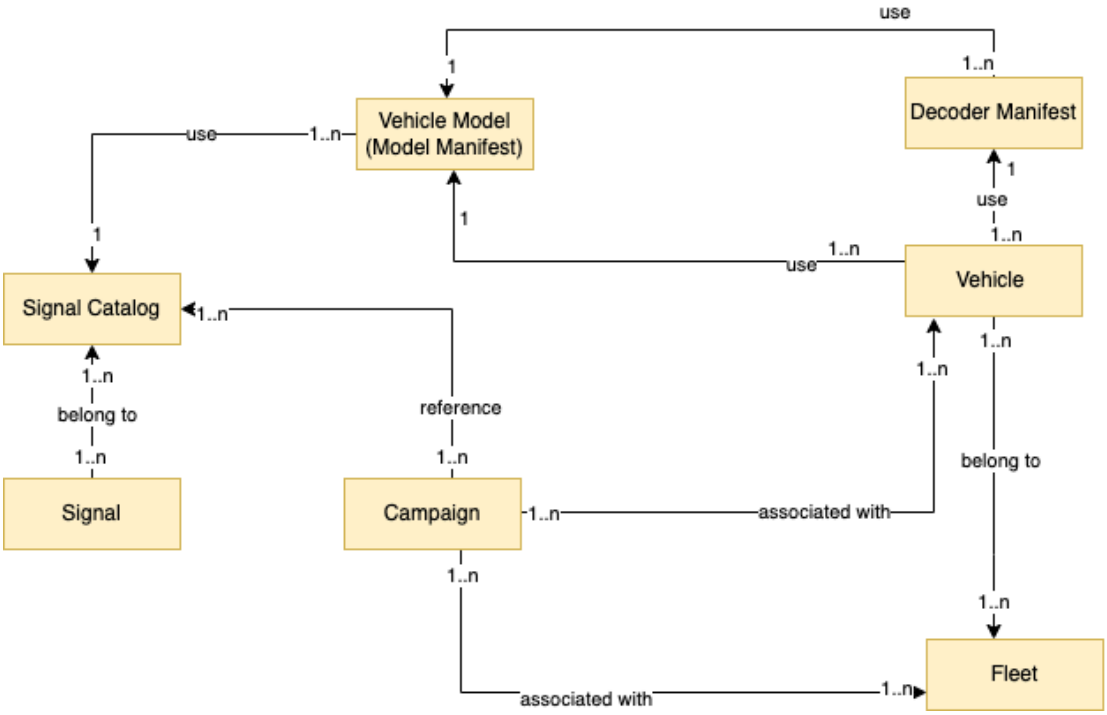
   > **Note**
   >
   > You must use the `UpdateCampaign` API operation to approve the campaign before AWS IoT FleetWise can deploy it to the vehicle or fleet. For more information, see Update an AWS IoT FleetWise campaign.

8. (Optional) Create state templates. State templates are deployed to a vehicle. State templates provide a mechanism for Vehicle owners to track the state of their vehicle. For more information, see Monitor the last known state of your vehicles.

The Edge Agent software transfers vehicle data to AWS IoT Core using an MQTT topic that you choose. To send the data to AWS IoT FleetWise for campaigns, it uses the reserved topic `$aws/iotfleetwise/vehicles/`*`vehicleName`*`/signals`. For Last Known State, the Edge Agent uses the reserved topic `$aws/iotfleetwise/vehicles/`*`vehicleName`*`/last_known_states/data`. For more information about how the ingested data is processed, see Visualize AWS IoT FleetWise vehicle data.

Model AWS IoT FleetWise vehicles

AWS IoT FleetWise provides a vehicle modeling framework that you can use to build virtual representations of your vehicles in the cloud. Signals, signal catalogs, vehicle models, and decoder manifests are the core components that you work with to model your vehicles.



Signal

Signals are fundamental structures that you define to contain vehicle data and its metadata. A signal can be an attribute, a branch, a sensor, or an actuator. For example, you can create a sensor to receive in-vehicle temperature values, and to store its metadata, including a sensor name, a data type, and a unit. For more information, see Manage AWS IoT FleetWise signal catalogs.

Signal catalog

A signal catalog contains a collection of signals. Signals in a signal catalog can be used to model vehicles that use different protocols and data formats. For example, there are two cars made by different automakers: one uses the Control Area Network (CAN bus) protocol; the other one uses the On-board Diagnostics (OBD) protocol. You can define a sensor in the signal catalog to receive in-vehicle temperature values. This sensor can be used to represent the thermocouples in both cars. For more information, see Manage AWS IoT FleetWise signal catalogs.

Vehicle model (model manifest)

Vehicle models are declarative structures that you can use to standardize the format of your vehicles and to define relationships between signals in the vehicles. Vehicle models enforce consistent information across multiple vehicles of the same type. You add signals to create vehicle models. For more information, see Manage AWS IoT FleetWise vehicle models.

Decoder manifest

Decoder manifests contain decoding information for each signal in vehicle models. Sensors and actuators in vehicles transmit low-level messages (binary data). With decoder manifests, AWS IoT FleetWise is able to transform binary data into human-readable values. Every decoder manifest is associated with a vehicle model. For more information, see Manage AWS IoT FleetWise decoder manifests.

You can use the AWS IoT FleetWise console or API to model vehicles in the following way.

1. Create or import a signal catalog containing signals that you'll use to create a vehicle model. For more information, see Create an AWS IoT FleetWise signal catalog and Import a signal catalog (AWS CLI).

   Note

   - If you use the AWS IoT FleetWise console to create the first vehicle model, you don't need to manually create a signal catalog. When you create your first vehicle model, AWS IoT FleetWise automatically creates a signal catalog for you. For more information, see Create an AWS IoT FleetWise vehicle model.
   - AWS IoT FleetWise currently supports a signal catalog for each AWS account per AWS Region.

2. Use signals in the signal catalog to create a vehicle model. For more information, see Create an AWS IoT FleetWise vehicle model.

   Note

   - If you use the AWS IoT FleetWise console to create a vehicle model, you can upload .dbc files to import signals. .dbc is a file format that Controller Area Network (CAN bus) databases support. After the vehicle model is created, new signals are

automatically added to the signal catalog. For more information, see Create an AWS IoT FleetWise vehicle model.

- If you use the `CreateModelManifest` API operation to create a vehicle model, you must use the `UpdateModelManifest` API operation to activate the vehicle model. For more information, see Update an AWS IoT FleetWise vehicle model.
- If you use the AWS IoT FleetWise console to create a vehicle model, AWS IoT FleetWise automatically activates the vehicle model for you.

3. Create a decoder manifest. The decoder manifest contains decoding information for every signal specified in the vehicle model that you created in the previous step. The decoder manifest is associated with the vehicle model that you created. For more information, see Manage AWS IoT FleetWise decoder manifests.

Note

- If you use the `CreateDecoderManifest` API operation to create a decoder manifest, you must use the `UpdateDecoderManifest` API operation to activate the decoder manifest. For more information, see Update an AWS IoT FleetWise decoder manifest.
- If you use the AWS IoT FleetWise console to create a decoder manifest, AWS IoT FleetWise automatically activates the decoder manifest for you.

CAN bus databases support the .dbc file format. You might upload .dbc files to import signals and signal decoders. To get an example .dbc file, do the following.

To get a .dbc file

1. Download the EngineSignals.zip.
2. Navigate to the directory where you downloaded the `EngineSignals.zip` file.
3. Unzip the file and save it locally as `EngineSignals.dbc`.

Topics

- Manage AWS IoT FleetWise signal catalogs
- Manage AWS IoT FleetWise vehicle models
- Manage AWS IoT FleetWise decoder manifests

Manage AWS IoT FleetWise signal catalogs

> **Note**
>
> You can download a demo script to convert ROS 2 messages to VSS .json files that are compatible with the signal catalog. For more information, see the *Vision System Data Developer Guide*.

A signal catalog is a collection of standardized signals that can be reused to create vehicle models. AWS IoT FleetWise supports Vehicle Signal Specification (VSS) that you can follow to define signals. A signal can be any of the following type.

Attribute

Attributes represent static information that generally doesn't change, such as manufacturer and manufacturing date.

Branch

Branches represent signals in a nested structure. Branches demonstrate signal hierarchies. For example, the `Vehicle` branch has a child branch, `Powertrain`. The `Powertrain` branch has a child branch, `combustionEngine`. To locate the `combustionEngine` branch, use the `Vehicle.Powertrain.combustionEngine` expression.

Sensor

Sensor data reports the current state of the vehicle and change over time, as the state of the vehicle changes, such as fluid levels, temperatures, vibrations, or voltage.

Actuator

Actuator data reports the state of a vehicle device, such as motors, heaters, and door locks. Changing the state of a vehicle device can update actuator data. For example, you can define an actuator to represent the heater. The actuator receives new data when you turn on or off the heater.

Custom structure

A custom structure (also known as a struct) represents a complex or higher-order data structure. It facilitates logical binding or grouping of data that originates from the same source. A struct is

used when data is read or written in an atomic operation, such as to represent a complex data type or higher-order shape.

A signal of struct type is defined in the signal catalog using a reference to a struct data type instead of a primitive data type. Structs can be used for all types of signals including sensors, attributes, actuators, and vision system data types. If a signal of struct type is sent or received, AWS IoT FleetWise expects all included items to have valid values, so all items are mandatory. For example, if a struct contains the items Vehicle.Camera.Image.height, Vehicle.Camera.Image.width, and Vehicle.Camera.Image.data – it's expected that the sent signal contains values for all of these items.

> **Note**
>
> Vision system data is in preview release and is subject to change.

Custom property

A custom property represents a member of the complex data structure. The data type of the property can be either primitive or another struct.

When representing a higher-order shape using a struct and custom property, the intended higher-order shape is always defined and visioned as a tree structure. The custom property is used to define all the leaf nodes while the struct is used to define all the non-leaf nodes.

> **Note**
>
> - If you use the AWS IoT FleetWise console to create the first vehicle model, you don't need to manually create a signal catalog. When you create your first vehicle model, AWS IoT FleetWise automatically creates a signal catalog for you. For more information, see Create an AWS IoT FleetWise vehicle model.
> - If you use the AWS IoT FleetWise console to create a vehicle model, you can upload .dbc files to import signals. .dbc is a file format that Controller Area Network (CAN bus) databases support. After the vehicle model is created, new signals are automatically added to the signal catalog. For more information, see Create an AWS IoT FleetWise vehicle model.
> - AWS IoT FleetWise currently supports a signal catalog for each AWS account per Region.

AWS IoT FleetWise provides the following API operations that you can use to create and manage signal catalogs.

- CreateSignalCatalog – Creates a new signal catalog.
- ImportSignalCatalog – Imports signals to create a signal catalog by uploading a .json file. Signals must be defined by following VSS and saved in the JSON format.
- UpdateSignalCatalog – Updates an existing signal catalog by updating, removing, or adding signals.
- DeleteSignalCatalog – Deletes an existing signal catalog.
- ListSignalCatalogs – Retrieves a paginated list of summaries of all signal catalogs.
- ListSignalCatalogNodes – Retrieves a paginated list of summaries of all signals (nodes) in a given signal catalog.
- GetSignalCatalog – Retrieves information about a signal catalog.

Tutorials

- Configure AWS IoT FleetWise signals
- Create an AWS IoT FleetWise signal catalog
- Import an AWS IoT FleetWise signal catalog
- Update an AWS IoT FleetWise signal catalog
- Delete an AWS IoT FleetWise signal catalog
- Get AWS IoT FleetWise signal catalog information

Configure AWS IoT FleetWise signals

This section shows you how to configure branches, attributes, sensors, and actuators.

Topics

- Configure branches
- Configure attributes
- Configure sensors or actuators
- Configure complex data types

Configure branches

To configure a branch, specify the following information.

- `fullyQualifiedName` – The fully qualified name of the branch is the path to the branch plus the branch's name. Use a dot(.) to refer to a child branch. For example, `Vehicle.Chassis.SteeringWheel` is the fully qualified name for the `SteeringWheel` branch. `Vehicle.Chassis.` is the path to this branch.

  The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, colon (:), and underscore (_).
- (Optional) `Description` – The description for the branch.

  The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `deprecationMessage` – The deprecation message for the node or branch being moved or deleted.

  The deprecationMessage can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the branch, such as the rationale for the branch or references to related branches.

  The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

Configure attributes

To configure an attribute, specify the following information.

- `dataType` – The attribute's data type must be one of the following: INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, BOOLEAN, FLOAT, DOUBLE, STRING, UNIX_TIMESTAMP, INT8_ARRAY, UINT8_ARRAY, INT16_ARRAY, UINT16_ARRAY, INT32_ARRAY, UINT32_ARRAY, INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, UNKNOWN, fullyQualifiedName, or a custom struct defined in the data type branch.

- `fullyQualifiedName` – The fully qualified name of the attribute is the path to the attribute plus the attribute's name. Use a dot(.) to refer to a child signal. For example, `Vehicle.Chassis.SteeringWheel.Diameter` is the fully qualified name for the `Diameter` attribute. `Vehicle.Chassis.SteeringWheel.` is the path to this attribute.

  The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).
- (Optional) `Description` – The description for the attribute.

  The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `unit` – The scientific unit for the attribute, such as km or Celsius.
- (Optional) `min` – The minimum value of the attribute.
- (Optional) `max` – The maximum value of the attribute.
- (Optional) `defaultValue` – The default value of the attribute.
- (Optional) `assignedValue` – The value assigned to the attribute.
- (Optional) `allowedValues` – A list of values that the attribute accepts.
- (Optional) `deprecationMessage` – The deprecation message for the node or branch that's being moved or deleted.

  The deprecationMessage can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the attribute, such as the rationale for the attribute or references to related attributes.

  The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

Configure sensors or actuators

To configure a sensor or actuator, specify the following information.

- `dataType` – The signal's data type must be one of the following: INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, BOOLEAN, FLOAT, DOUBLE, STRING, UNIX_TIMESTAMP, INT8_ARRAY, UINT8_ARRAY, INT16_ARRAY, UINT16_ARRAY, INT32_ARRAY, UINT32_ARRAY,

INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, UNKNOWN, fullyQualifiedName, or a custom struct defined in the data type branch.

- `fullyQualifiedName` – The fully qualified name of the signal is the path to the signal plus the signal's name. Use a dot(.) to refer to a child signal. For example, `Vehicle.Chassis.SteeringWheel.HandsOff.HandsOffSteeringState` is the fully qualified name for the `HandsOffSteeringState` actuator. `Vehicle.Chassis.SteeringWheel.HandsOff.` is the path to this actuator.

  The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).
- (Optional) `Description` – The description for the signal.

  The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `unit` – The scientific unit for the signal, such as km or Celsius.
- (Optional) `min` – The minimum value of the signal.
- (Optional) `max` – The maximum value of the signal.
- (Optional) `assignedValue` – The value assigned to the signal.
- (Optional) `allowedValues` – list of values that the signal accepts.
- (Optional) `deprecationMessage` – The deprecation message for the node or branch that's being moved or deleted.

  The deprecationMessage can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the sensor or actuator, such as their rationale or references to related sensors or actuators.

  The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

Configure complex data types

Complex data types are used when modeling vision systems. In addition to branches, these data types are made up of structures (also known as a struct) and properties. A struct is a signal that

is described by multiple values, like an image. A property represents a member of the struct, like a primitive data type (such as UINT8) or another struct (such as timestamp). For example, Vehicle.Cameras.Front represents a branch, Vehicle.Cameras.Front.Image represents a struct, and Vehicle.Cameras.Timestamp represents a property.

The following complex data type example demonstrates how signals and data types are exported to a single .json file.

Example complex data type

```
{
  "Vehicle": {
    "type": "branch"
    // Signal tree
  },
  "ComplexDataTypes": {
    "VehicleDataTypes": {
      // complex data type tree
      "children": {
        "branch": {
          "children": {
            "Struct": {
              "children": {
                "Property": {
                  "type": "property",
                  "datatype": "Data type",
                  "description": "Description",
                  //                  ...
                }
              },
              "description": "Description",
              "type": "struct"
            }
          }
          "description": "Description",
          "type": "branch"
        }
      }
    }
  }
}
```

Note

You can download a demo script to convert ROS 2 messages to VSS .json files that are compatible with the signal catalog. For more information, see the *Vision System Data Developer Guide*.
Vision system data is in preview release and is subject to change.

Configure struct

To configure a custom structure (or struct), specify the following information.

- `fullyQualifiedName` – The fully qualified name of the custom structure.
  For example, the fully qualified name of a custom structure might be `ComplexDataTypes.VehicleDataTypes.SVMCamera`.

  The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).
- (Optional) `Description` – The description for the signal.

  The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `deprecationMessage` – The deprecation message for the node or branch that's being moved or deleted.

  The deprecationMessage can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the sensor or actuator, such as their rationale or references to related sensors or actuators.

  The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

Configure property

To configure a custom property, specify the following information.

- `dataType` – The signal's data type must be one of the following: INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, BOOLEAN, FLOAT, DOUBLE, STRING, UNIX_TIMESTAMP, INT8_ARRAY, UINT8_ARRAY, INT16_ARRAY, UINT16_ARRAY, INT32_ARRAY, UINT32_ARRAY, INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, STRUCT, STRUCT_ARRAY, or UNKNOWN.
- `fullyQualifiedName` – The fully qualified name of the custom property.
  For example, the fully qualified name of a custom property might be `ComplexDataTypes.VehicleDataTypes.SVMCamera.FPS`.

  The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore)
- (Optional) `Description` – The description for the signal.

  The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `deprecationMessage` – The deprecation message for the node or branch that's being moved or deleted.

  The deprecationMessage can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the sensor or actuator, such as their rationale or references to related sensors or actuators.

  The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).
- (Optional) `dataEncoding` – Indicates whether the property is binary data. The custom property's data encoding must be one of the following: BINARY or TYPED.
- (Optional) `structFullyQualifiedName` – The fully qualified name of the structure (struct) node for the custom property if the data type of the custom property is Struct or StructArray.

  The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).

Create an AWS IoT FleetWise signal catalog

You can use the CreateSignalCatalog API operation to create a signal catalog. The following example uses AWS CLI.

To create a signal catalog, run the following command.

Replace *signal-catalog-configuration* with the name of the .json file that contains the configuration.

```
aws iotfleetwise create-signal-catalog --cli-input-json file://signal-catalog-configuration.json
```

- Replace *signal-catalog-name* with the name of the signal catalog that you're creating.
- (Optional) Replace *description* with a description to help you identify the signal catalog.

For more information about how to configure branches, attributes, sensors, and actuators, see Configure AWS IoT FleetWise signals.

```
{
    "name": "signal-catalog-name",
    "description": "description",
    "nodes": [
  {
    "branch": {
      "fullyQualifiedName": "Types"
    }
  },
  {
    "struct": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage"
    }
  },
  {
    "struct": {
      "fullyQualifiedName": "Types.std_msgs_Header"
    }
  },
  {
    "struct": {
      "fullyQualifiedName": "Types.builtin_interfaces_Time"
```

```
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.builtin_interfaces_Time.sec",
      "dataType": "INT32",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.builtin_interfaces_Time.nanosec",
      "dataType": "UINT32",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.std_msgs_Header.stamp",
      "dataType": "STRUCT",
      "structFullyQualifiedName": "Types.builtin_interfaces_Time"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.std_msgs_Header.frame_id",
      "dataType": "STRING",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage.header",
      "dataType": "STRUCT",
      "structFullyQualifiedName": "Types.std_msgs_Header"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage.format",
      "dataType": "STRING",
      "dataEncoding": "TYPED"
    }
  },
```

```
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage.data",
      "dataType": "UINT8_ARRAY",
      "dataEncoding": "BINARY"
    }
  },
  {
    "branch": {
      "fullyQualifiedName": "Vehicle",
      "description": "Vehicle"
    }
  },
  {
    "branch": {
      "fullyQualifiedName": "Vehicle.Cameras"
    }
  },
  {
    "branch": {
      "fullyQualifiedName": "Vehicle.Cameras.Front"
    }
  },
  {
    "sensor": {
      "fullyQualifiedName": "Vehicle.Cameras.Front.Image",
      "dataType": "STRUCT",
      "structFullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage"
    }
  },
  {
    "struct": {
      "fullyQualifiedName": "Types.std_msgs_msg_Float64"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.std_msgs_msg_Float64.data",
      "dataType": "DOUBLE",
      "dataEncoding": "TYPED"
    }
  },
  {
    "sensor": {
```

```
      "fullyQualifiedName": "Vehicle.Velocity",
      "dataType": "STRUCT",
      "structFullyQualifiedName": "Types.std_msgs_msg_Float64"
    }
  },
  {
    "struct": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.x_offset",
      "dataType": "UINT32",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.y_offset",
      "dataType": "UINT32",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.height",
      "dataType": "UINT32",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.width",
      "dataType": "UINT32",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.do_rectify",
      "dataType": "BOOLEAN",
      "dataEncoding": "TYPED"
    }
```

```
  },
  {
    "branch": {
      "fullyQualifiedName": "Vehicle.Perception"
    }
  },
  {
    "sensor": {
      "fullyQualifiedName": "Vehicle.Perception.Obstacle",
      "dataType": "STRUCT",
      "structFullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest"
    }
  }
]
}
```

Note

You can download a demo script to convert ROS 2 messages to VSS .json files that are compatible with the signal catalog. For more information, see the *Vision System Data Developer Guide*.
Vision system data is in preview release and is subject to change.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `CreateSignalCatalog` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
```

```
        }
    ]
}
```

Import an AWS IoT FleetWise signal catalog

You can use the AWS IoT FleetWise console or API to import a signal catalog.

Topics

Import a signal catalog (console)

You can use the AWS IoT FleetWise console to import a signal catalog.

> **Important**
>
> You can have a maximum of one signal catalog. If you already have a signal catalog, you won't see the option to import a signal catalog in the console.

To import a signal catalog

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Signal catalog**.
3. On the signal catalog summary page, choose **Import signal catalog**.
4. Import the file containing the signals.
   - To upload a file from an S3 bucket:
     a. Choose **Import from S3**.
     b. Choose **Browse S3**.
     c. For **Buckets**, enter the bucket name or object, choose it from the list, and then choose the file from the list. Choose the **Choose file** button.

Or, for **S3 URI**, enter an Amazon Simple Storage Service URI. For more information, see Methods for accessing a bucket in the *Amazon S3 User Guide*.

- To upload a file from your computer:

  a. Choose **Import from file**.

  b. Upload a .json file in a Vehicle Signal Specification (VSS) format.

5. Verify the signal catalog, and then choose **Import file**.

Import a signal catalog (AWS CLI)

You can use the ImportSignalCatalog API operation to upload a JSON file that helps create a signal catalog. You must follow the Vehicle Signal Specification (VSS) to save signals in the JSON file. The following example uses AWS CLI.

To import a signal catalog, run the following command.

- Replace *signal-catalog-name* with the name of the signal catalog that you're creating.
- (Optional) Replace description with a *description* to help you identify the signal catalog.
- Replace *signal-catalog-configuration-vss* with the name of the JSON string file that contains signals defined in VSS.

For more information about how to configure branches, attributes, sensors, and actuators, see Configure AWS IoT FleetWise signals.

```
aws iotfleetwise import-signal-catalog \
                --name signal-catalog-name \
                --description  description \
                --vss file://signal-catalog-configuration-vss.json
```

The JSON must be stringified and passed through the `vssJson` field. The following is an example of signals defined in VSS.

```
{
 "Vehicle": {
  "type": "branch",
  "children": {
```

```
   "Chassis": {
    "type": "branch",
    "description": "All data concerning steering, suspension, wheels, and brakes.",
    "children": {
     "SteeringWheel": {
      "type": "branch",
      "description": "Steering wheel signals",
      "children": {
       "Diameter": {
        "type": "attribute",
        "description": "The diameter of the steering wheel",
        "datatype": "float",
        "unit": "cm",
        "min": 1,
        "max": 50
       },
       "HandsOff": {
        "type": "branch",
        "children": {
         "HandsOffSteeringState": {
          "type": "actuator",
          "description": "HndsOffStrWhlDtSt. Hands Off Steering State",
          "datatype": "boolean"
         },
         "HandsOffSteeringMode": {
          "type": "actuator",
          "description": "HndsOffStrWhlDtMd. Hands Off Steering Mode",
          "datatype": "int8",
          "min": 0,
          "max": 2
         }
        }
       }
      }
     },
     "Accelerator": {
      "type": "branch",
      "description": "",
      "children": {
       "AcceleratorPedalPosition": {
        "type": "sensor",
        "description": "Throttle__Position. Accelerator pedal position as percent. 0 =
 Not depressed. 100 = Fully depressed.",
        "datatype": "uint8",
```

```
        "unit": "%",
        "min": 0,
        "max": 100.000035
       }
      }
     }
    }
   },
   "Powertrain": {
    "type": "branch",
    "description": "Powertrain data for battery management, etc.",
    "children": {
     "Transmission": {
      "type": "branch",
      "description": "Transmission-specific data, stopping at the drive shafts.",
      "children": {
       "VehicleOdometer": {
        "type": "sensor",
        "description": "Vehicle_Odometer",
        "datatype": "float",
        "unit": "km",
        "min": 0,
        "max": 67108863.984375
       }
      }
     },
     "CombustionEngine": {
      "type": "branch",
      "description": "Engine-specific data, stopping at the bell housing.",
      "children": {
       "Engine": {
        "type": "branch",
        "description": "Engine description",
        "children": {
         "timing": {
          "type": "branch",
          "description": "timing description",
          "children": {
           "run_time": {
            "type": "sensor",
            "description": "Engine run time",
            "datatype": "int16",
            "unit": "ms",
            "min": 0,
```

```
            "max": 10000
           },
           "idle_time": {
            "type": "sensor",
            "description": "Engine idle time",
            "datatype": "int16",
            "min": 0,
            "unit": "ms",
            "max": 10000
           }
          }
         }
        }
       }
      }
     }
    },
    "Axle": {
     "type": "branch",
     "description": "Axle signals",
     "children": {
      "TireRRPrs": {
       "type": "sensor",
       "description": "TireRRPrs. Right rear Tire pressure in kilo-Pascal",
       "datatype": "float",
       "unit": "kPaG",
       "min": 0,
       "max": 1020
      }
     }
    }
   }
  },
  "Cameras": {
   "type": "branch",
   "description": "Branch to aggregate all cameras in the vehicle",
   "children": {
    "FrontViewCamera": {
     "type": "sensor",
     "datatype": "VehicleDataTypes.SVMCamera",
     "description": "Front view camera"
    },
    "RearViewCamera": {
```

```
    "type": "sensor",
    "datatype": "VehicleDataTypes.SVMCamera",
    "description": "Rear view camera"
   },
   "LeftSideViewCamera": {
    "type": "sensor",
    "datatype": "VehicleDataTypes.SVMCamera",
    "description": "Left side view camera"
   },
   "RightSideViewCamera": {
    "type": "sensor",
    "datatype": "VehicleDataTypes.SVMCamera",
    "description": "Right side view camera"
   }
  }
 },
 "ComplexDataTypes": {
  "VehicleDataTypes": {
   "type": "branch",
   "description": "Branch to aggregate all camera related higher order data types",
   "children": {
    "SVMCamera": {
     "type": "struct",
     "description": "This data type represents Surround View Monitor (SVM) camera
 system in a vehicle",
     "comment": "Test comment",
     "deprecation": "Test deprecation message",
     "children": {
      "Make": {
       "type": "property",
       "description": "Make of the SVM camera",
       "datatype": "string",
       "comment": "Test comment",
       "deprecation": "Test deprecation message"
      },
      "Description": {
       "type": "property",
       "description": "Description of the SVM camera",
       "datatype": "string",
       "comment": "Test comment",
       "deprecation": "Test deprecation message"
      },
      "FPS": {
       "type": "property",
```

```
        "description": "FPS of the SVM camera",
        "datatype": "double",
        "comment": "Test comment",
        "deprecation": "Test deprecation message"
       },
       "Orientation": {
        "type": "property",
        "description": "Orientation of the SVM camera",
        "datatype": "VehicleDataTypes.Orientation",
        "comment": "Test comment",
        "deprecation": "Test deprecation message"
       },
       "Range": {
        "type": "property",
        "description": "Range of the SVM camera",
        "datatype": "VehicleDataTypes.Range",
        "comment": "Test comment",
        "deprecation": "Test deprecation message"
       },
       "RawData": {
        "type": "property",
        "description": "Represents binary data of the SVM camera",
        "datatype": "uint8[]",
        "dataencoding": "binary",
        "comment": "Test comment",
        "deprecation": "Test deprecation message"
       },
       "CapturedFrames": {
        "type": "property",
        "description": "Represents selected frames captured by the SVM camera",
        "datatype": "VehicleDataTypes.Frame[]",
        "dataencoding": "typed",
        "comment": "Test comment",
        "deprecation": "Test deprecation message"
       }
      }
     },
     "Range": {
      "type": "struct",
      "description": "Range of a camera in centimeters",
      "comment": "Test comment",
      "deprecation": "Test deprecation message",
      "children": {
       "Min": {
```

```
      "type": "property",
      "description": "Minimum range of a camera in centimeters",
      "datatype": "uint32",
      "comment": "Test comment",
      "deprecation": "Test deprecation message"
     },
     "Max": {
      "type": "property",
      "description": "Maximum range of a camera in centimeters",
      "datatype": "uint32",
      "comment": "Test comment",
      "deprecation": "Test deprecation message"
     }
    }
   },
   "Orientation": {
    "type": "struct",
    "description": "Orientation of a camera",
    "comment": "Test comment",
    "deprecation": "Test deprecation message",
    "children": {
     "Front": {
      "type": "property",
      "description": "Indicates whether the camera is oriented to the front of the
vehicle",
      "datatype": "boolean",
      "comment": "Test comment",
      "deprecation": "Test deprecation message"
     },
     "Rear": {
      "type": "property",
      "description": "Indicates whether the camera is oriented to the rear of the
vehicle",
      "datatype": "boolean",
      "comment": "Test comment",
      "deprecation": "Test deprecation message"
     },
     "Side": {
      "type": "property",
      "description": "Indicates whether the camera is oriented to the side of the
vehicle",
      "datatype": "boolean",
      "comment": "Test comment",
      "deprecation": "Test deprecation message"
```

```
      }
     }
    },
    "Frame": {
     "type": "struct",
     "description": "Represents a camera frame",
     "comment": "Test comment",
     "deprecation": "Test deprecation message",
     "children": {
      "Data": {
       "type": "property",
       "datatype": "string",
       "dataencoding": "binary",
       "comment": "Test comment",
       "deprecation": "Test deprecation message"
      }
     }
    }
   }
  }
 }

}
```

The following example shows the same signals defined in VSS in a JSON string.

```
{
   "vssJson": "{\"Vehicle\":{\"type\":\"branch\",\"children\":{\"Chassis\":{\"type
\":\"branch\",\"description\":\"All data concerning steering, suspension, wheels,
 and brakes.\",\"children\":{\"SteeringWheel\":{\"type\":\"branch\",\"description
\":\"Steering wheel signals\",\"children\":{\"Diameter\":{\"type\":\"attribute\",
\"description\":\"The diameter of the steering wheel\",\"datatype\":\"float\",\"unit
\":\"cm\",\"min\":1,\"max\":50},\"HandsOff\":{\"type\":\"branch\",\"children\":
{\"HandsOffSteeringState\":{\"type\":\"actuator\",\"description\":\"HndsOffStrWhlDtSt.
 Hands Off Steering State\",\"datatype\":\"boolean\"},\"HandsOffSteeringMode\":
{\"type\":\"actuator\",\"description\":\"HndsOffStrWhlDtMd. Hands Off Steering Mode
\",\"datatype\":\"int8\",\"min\":0,\"max\":2}}}}},\"Accelerator\":{\"type\":\"branch
\",\"description\":\"\",\"children\":{\"AcceleratorPedalPosition\":{\"type\":\"sensor
\",\"description\":\"Throttle__Position. Accelerator pedal position as percent. 0
 = Not depressed. 100 = Fully depressed.\",\"datatype\":\"uint8\",\"unit\":\"%\",
\"min\":0,\"max\":100.000035}}}}},\"Powertrain\":{\"type\":\"branch\",\"description
\":\"Powertrain data for battery management, etc.\",\"children\":{\"Transmission\":
{\"type\":\"branch\",\"description\":\"Transmission-specific data, stopping at the
```

```
 drive shafts.\",\"children\":{\"VehicleOdometer\":{\"type\":\"sensor\",\"description
\":\"Vehicle_Odometer\",\"datatype\":\"float\",\"unit\":\"km\",\"min\":0,\"max
\":67108863.984375}}},\"CombustionEngine\":{\"type\":\"branch\",\"description\":
\"Engine-specific data, stopping at the bell housing.\",\"children\":{\"Engine\":
{\"type\":\"branch\",\"description\":\"Engine description\",\"children\":{\"timing\":
{\"type\":\"branch\",\"description\":\"timing description\",\"children\":{\"run_time\":
{\"type\":\"sensor\",\"description\":\"Engine run time\",\"datatype\":\"int16\",\"unit
\":\"ms\",\"min\":0,\"max\":10000},\"idle_time\":{\"type\":\"sensor\",\"description
\":\"Engine idle time\",\"datatype\":\"int16\",\"min\":0,\"unit\":\"ms\",\"max
\":10000}}}}}}}},\"Axle\":{\"type\":\"branch\",\"description\":\"Axle signals\",
\"children\":{\"TireRRPrs\":{\"type\":\"sensor\",\"description\":\"TireRRPrs. Right
 rear Tire pressure in kilo-Pascal\",\"datatype\":\"float\",\"unit\":\"kPaG\",\"min
\":0,\"max\":1020}}}}}}"
}
```


> **Note**
>
> You can download a demo script to convert ROS 2 messages to VSS JSON files that are compatible with the signal catalog. For more information, see the *Vision System Data Developer Guide*.
> Vision system data is in preview release and is subject to change.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ImportSignalCatalog` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
```

```
    ]
}
```

Update an AWS IoT FleetWise signal catalog

You can use the UpdateSignalCatalog API operation to update an existing signal catalog. The following example uses AWS CLI.

To update an existing signal catalog, run the following command.

Replace *signal-catalog-configuration* with the name of the .json file that contains the configuration.

```
aws iotfleetwise update-signal-catalog --cli-input-json file://signal-catalog-configuration.json
```

Replace *signal-catalog-name* with the name of the signal catalog that you're updating.

For more information about how to configure branches, attributes, sensors, and actuators, see Configure AWS IoT FleetWise signals.

> **Important**
>
> Custom structures are immutable. If you need to re-order or insert properties to an existing custom structure (struct), delete the structure and create a brand-new structure with the desired order of properties.
> To delete a custom structure, add the structure's fully qualified name in `nodesToRemove`. A structure can't be deleted if it's referred to by any signals. Any signals that refer to the structure (their data type is defined as the target structure) must be updated or deleted before the request to update the signal catalog.

```
{
     "name": "signal-catalog-name",
     "nodesToAdd": [{
       "branch": {
        "description": "Front left of vehicle specific data.",
        "fullyQualifiedName": "Vehicle.Front.Left"
       }
```

```
      },
      {
       "branch": {
        "description": "Door-specific data for the front left of vehicle.",
        "fullyQualifiedName": "Vehicle.Front.Left.Door"
       }
      },
      {
       "actuator": {
        "fullyQualifiedName": "Vehicle.Front.Left.Door.Lock",
        "description": "Whether the front left door is locked.",
        "dataType": "BOOLEAN"
       }
      },
      {
       "branch": {
        "fullyQualifiedName": "Vehicle.Camera"
       }
      },
      {
       "struct": {
        "fullyQualifiedName": "Vehicle.Camera.SVMCamera"
       }
      },
      {
       "property": {
        "fullyQualifiedName": "Vehicle.Camera.SVMCamera.ISO",
        "dataType": "STRING"
       }
      }
     ],
     "nodesToRemove": ["Vehicle.Chassis.SteeringWheel.HandsOffSteeringState"],
     "nodesToUpdate": [{
      "attribute": {
       "dataType": "FLOAT",
       "fullyQualifiedName": "Vehicle.Chassis.SteeringWheel.Diameter",
       "max": 55
      }
     }]
    }
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `UpdateSignalCatalog` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Verify signal catalog update

You can use the ListSignalCatalogNodes API operation to verify if a signal catalog was updated. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all signals (nodes) in a given signal catalog, run the following command.

Replace *signal-catalog-name* with the name of the signal catalog that you're checking.

```
aws iotfleetwise list-signal-catalog-nodes --name signal-catalog-name
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListSignalCatalogNodes` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Delete an AWS IoT FleetWise signal catalog

You can use the DeleteSignalCatalog API operation to delete a signal catalog. The following example uses AWS CLI.

> **Important**
>
> Before deleting a signal catalog, make sure it has no associated vehicle models, decoder manifests, vehicles, fleets, or campaigns. For instructions, see the following:
>
> - Delete an AWS IoT FleetWise vehicle model
> - Delete an AWS IoT FleetWise decoder manifest
> - Delete an AWS IoT FleetWise vehicle
> - Delete an AWS IoT FleetWise fleet
> - Delete an AWS IoT FleetWise campaign

To delete an existing signal catalog, run the following command. Replace *signal-catalog-name* with the name of the signal catalog that you're deleting.

```
aws iotfleetwise delete-signal-catalog --name signal-catalog-name
```

Verify signal catalog deletion

You can use the ListSignalCatalogs API operation to verify if a signal catalog has been deleted. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all signal catalogs, run the following command.

```
aws iotfleetwise list-signal-catalogs
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListSignalCatalogs` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Get AWS IoT FleetWise signal catalog information

You can use the GetSignalCatalog API operation to retrieve signal catalog information. The following example uses AWS CLI.

To retrieve information about a signal catalog, run the following command.

Replace *signal-catalog-name* with the name of the signal catalog that you want to retrieve.

```
aws iotfleetwise get-signal-catalog --name signal-catalog-name
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `GetSignalCatalog` API operation.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Note

This operation is eventually consistent. In other words, changes to the signal catalog might not be reflected immediately.

Manage AWS IoT FleetWise vehicle models

You use signals to create vehicle models that help standardize the format of your vehicles. Vehicle models enforce consistent information across multiple vehicles of the same type, so that you can process data from fleets of vehicles. Vehicles created from the same vehicle model inherit the same group of signals. For more information, see Manage AWS IoT FleetWise vehicles.

Each vehicle model has a status field that contains the state of the vehicle model. The state can be one of the following values:

- `ACTIVE` – The vehicle model is active.
- `DRAFT` – The configuration of the vehicle model is saved.

Important

- You must have a signal catalog before you can create a vehicle model using the `CreateModelManifest` API operation. For more information, see Create an AWS IoT FleetWise signal catalog.

- If you use the AWS IoT FleetWise console to create a vehicle model, AWS IoT FleetWise automatically activates the vehicle model for you.
- If you use the `CreateModelManifest` API operation to create a vehicle model, the vehicle model stays in the `DRAFT` state.
- You can't create vehicles from vehicle models that are in the `DRAFT` state. Use the `UpdateModelManifest` API operation to change vehicle models to the `ACTIVE` state.
- You can't edit vehicle models that are in the `ACTIVE` state.

Topics

Create an AWS IoT FleetWise vehicle model

You can use the AWS IoT FleetWise console or API to create vehicle models.

Topics

Create a vehicle model (console)

In the AWS IoT FleetWise console, you can create a vehicle model in the following ways:

Use a template provided by AWS

AWS IoT FleetWise provides an On-board Diagnostic (OBD) II, J1979 template that automatically creates a signal catalog, a vehicle model, and a decoder manifest for you. The template also adds

OBD network interfaces to the decoder manifest. For more information, see Manage AWS IoT FleetWise decoder manifests.

To create a vehicle model by using a template

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. On the **Vehicle models** page, choose **Add provided template**.
4. Choose **On-board diagnostics (OBD) II**.
5. Enter a name for the OBD network interface that AWS IoT FleetWise is creating.
6. Choose **Add**.

Manually create a vehicle model

You can add signals from the signal catalog or import signals by uploading one or more .dbc files. A .dbc file is a file format that Controller Area Network (CAN bus) databases support.

> **Important**
>
> You can't create a vehicle model with vision system data signals using the AWS IoT FleetWise console. Instead, use the AWS CLI to create a vehicle model.
> Vision system data is in preview release and is subject to change.

To manually create a vehicle model

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. On the **Vehicle models** page, choose **Create vehicle model**, and then do the following.

Topics

- Step 1: Configure vehicle model
- Step 2: Add signals
- Step 3: Import signals
- (Optional) Step 4: Add attributes

- Step 5: Review and create

Step 1: Configure vehicle model

In **General information**, do the following.

1. Enter a name for the vehicle model.
2. (Optional) Enter a description.
3. Choose **Next**.

Step 2: Add signals

> **Note**
>
> - If this is the first time you've used AWS IoT FleetWise, this step isn't available until you have a signal catalog. When the first vehicle model is created, AWS IoT FleetWise automatically creates a signal catalog with signals added to the first vehicle model.
> - If you're experienced with AWS IoT FleetWise, you can add signals to your vehicle model by selecting signals from the signal catalog or uploading .dbc files to import signals.
> - You must have at least one signal to create a vehicle model.

To add signals

1. Choose one or more signals from the signal catalog that you're adding to the vehicle model. You can review selected signals in the right pane.

   > **Note**
   >
   > Only selected signals will be added to the vehicle model.

2. Choose **Next**.

Step 3: Import signals

> **Note**
>
> - If this is the first time you've used AWS IoT FleetWise, you must upload at least one .dbc file to import signals.
> - If you're experienced with AWS IoT FleetWise, you can add signals to your vehicle model by selecting signals from the signal catalog or uploading .dbc files to import signals.
> - You must have at least one signal to create a vehicle model.

To import signals

1. Choose **Choose files**.
2. In the dialog box, choose the .dbc file that contains signals. You can upload multiple .dbc files.
3. AWS IoT FleetWise parses your .dbc files to retrieve signals.

   In the **Signals** section, specify the following metadata for each signal.

   - **Name** – The signal's name.

     The signal name must be unique. The signal name plus the path can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).
   - **Data type** – The signal's data type must be one of the following: INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, BOOLEAN, FLOAT, DOUBLE, STRING, UNIX_TIMESTAMP, INT8_ARRAY, UINT8_ARRAY, INT16_ARRAY, UINT16_ARRAY, INT32_ARRAY, UINT32_ARRAY, INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, or UNKNOWN.
   - **Signal type** – The type of the signal, which can be **Sensor** or **Actuator**.
   - (Optional) **Unit** – The scientific unit for the signal, such as km or Celsius.
   - (Optional) **Path** – The path to the signal. Similar to JSONPath, use a dot(.) to refer to a child signal. For example, **`Vehicle.Engine.Light`**.

     The signal name plus the path can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).
   - (Optional) **Min** – The minimum value of the signal.

- (Optional) **Max** – The maximum value of the signal.
- (Optional) **Description** – The description for the signal.

  The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

4. Choose **Next**.

(Optional) Step 4: Add attributes

You can add up to 100 attributes, including the existing attributes in the signal catalog.

To add attributes

1. In **Add attributes**, specify the following metadata for each attribute.

   - **Name** – The attribute's name.

     The signal name must be unique. The signal name and path can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore)
   - **Data type** – The attribute's data type must be one of the following: INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, BOOLEAN, FLOAT, DOUBLE, STRING, UNIX_TIMESTAMP, INT8_ARRAY, UINT8_ARRAY, INT16_ARRAY, UINT16_ARRAY, INT32_ARRAY, UINT32_ARRAY, INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, or UNKNOWN
   - (Optional) **Unit** – The scientific unit for the attribute, such as km or Celsius.
   - (Optional) **Path** – The path to the signal. Similar to JSONPath, use a dot(.) to refer to a child signal. For example, **`Vehicle.Engine.Light`**.

     The signal name plus the path can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore)
   - (Optional) **Min** – The minimum value of the attribute.
   - (Optional) **Max** – The maximum value of the attribute.
   - (Optional) **Description** – The description for the attribute.

     The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

2. Choose **Next**.

Step 5: Review and create

Verify the configurations for the vehicle model, and then choose **Create**.

Duplicate a vehicle model

AWS IoT FleetWise can copy the configurations of an existing vehicle model to create a new model. Signals specified in the selected vehicle model are copied to the new vehicle model.

To duplicate a vehicle model

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. Choose a model from the vehicle model list, and then choose **Duplicate model**.

To configure the vehicle model, follow the Manually create a vehicle model tutorial.

It can take a few minutes for AWS IoT FleetWise to process your request to create the vehicle model. After the vehicle model is successfully created, on the **Vehicle models** page, the **Status** column shows **ACTIVE**. When the vehicle model becomes active, you can't edit it.

Create a vehicle model (AWS CLI)

You can use the CreateModelManifest API operation to create vehicle models (model manifests). The following example uses the AWS CLI.

> **Important**
>
> You must have a signal catalog before you can create a vehicle model using the `CreateModelManifest` API operation. For more information about how to create a signal catalog, see Create an AWS IoT FleetWise signal catalog.

To create a vehicle model, run the following command.

Replace *`vehicle-model-configuration`* with the name of the .json file that contains the configuration.

```
aws iotfleetwise create-model-manifest --cli-input-json file://vehicle-model-configuration.json
```

- Replace *vehicle-model-name* with the name of the vehicle model that you're creating.
- Replace *signal-catalog-ARN* with the Amazon Resource Name (ARN) of the signal catalog.
- (Optional) Replace *description* with a description to help you identify the vehicle model.

For more information about how to configure branches, attributes, sensors, and actuators, see Configure AWS IoT FleetWise signals.

```
{
    "name": "vehicle-model-name",
    "signalCatalogArn": "signal-catalog-ARN",
    "description": "description",
    "nodes": ["Vehicle.Chassis"]
}
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `CreateModelManifest` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Update an AWS IoT FleetWise vehicle model

You can use the UpdateModelManifest API operation to update an existing vehicle model (model manifests). The following example uses the AWS CLI.

To update an existing vehicle model, run the following command.

Replace *update-vehicle-model-configuration* with the name of the .json file that contains the configuration.

```
aws iotfleetwise update-model-manifest --cli-input-json file://update-vehicle-model-configuration.json
```

- Replace *vehicle-model-name* with the name of the vehicle model that you're updating.
- (Optional) To activate the vehicle model, replace *vehicle-model-status* with `ACTIVE`.

> **Important**
>
> After the vehicle model is activated, you can't change the vehicle model.

- (Optional) Replace *description* with an updated description to help you identify the vehicle model.

```
{
    "name": "vehicle-model-name",
    "status": "vehicle-model-status",
    "description": "description",
    "nodesToAdd": ["Vehicle.Front.Left"],
    "nodesToRemove": ["Vehicle.Chassis.SteeringWheel"],
}
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `UpdateModelManifest` API operation.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Verify vehicle model update

You can use the ListModelManifestNodes API operation to verify if a vehicle model was updated. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all signals (nodes) in a given vehicle model, run the following command.

Replace *vehicle-model-name* with the name of the vehicle model that you're checking.

```
aws iotfleetwise list-model-manifest-nodes /
                --name vehicle-model-name
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListModelManifestNodes` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
```

```
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Delete an AWS IoT FleetWise vehicle model

You can use the AWS IoT FleetWise console or API to delete vehicle models.

> **Important**
>
> Vehicles and decoder manifests associated with the vehicle model must be deleted first. For more information, see Delete an AWS IoT FleetWise vehicle and Delete an AWS IoT FleetWise decoder manifest.

Delete a vehicle model (console)

To delete a vehicle model, use the AWS IoT FleetWise console.

To delete a vehicle model

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. On the **Vehicle models** page, choose the target vehicle model.
4. Choose **Delete**.
5. In **Delete `vehicle-model-name`?**, enter the name of the vehicle model to delete, and then choose **Confirm**.

Delete a vehicle model (AWS CLI)

You can use the DeleteModelManifest API operation to delete an existing vehicle model (model manifests). The following example uses the AWS CLI.

To delete a vehicle model, run the following command.

Replace *`model-manifest-name`* with the name of the vehicle model that you're deleting.

```
aws iotfleetwise delete-model-manifest --name model-manifest-name
```

Verify vehicle model deletion

You can use the ListModelManifests API operation to verify if a vehicle model was deleted. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all vehicle models, run the following command.

```
aws iotfleetwise list-model-manifests
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListModelManifests` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Get AWS IoT FleetWise vehicle model information

You can use the GetModelManifest API operation to retrieve information about a vehicle model. The following example uses AWS CLI.

To retrieve information about a vehicle model, run the following command.

Replace *vehicle-model* with the name of the vehicle model that you want to retrieve.

```
aws iotfleetwise get-model-manifest --name vehicle-model
```

> **Note**
>
> This operation is eventually consistent. In other words, changes to the vehicle model might not be reflected immediately.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `GetModelManifest` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Manage AWS IoT FleetWise decoder manifests

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Decoder manifests contain decoding information that AWS IoT FleetWise uses to transform vehicle data (binary data) into human-readable values and to prepare your data for data analyses. Network

interface and signal decoders are the core components that you work with to configure decoder manifests.

Network interface

Contains information about the protocol that the in-vehicle network uses. AWS IoT FleetWise supports the following protocols.

Controller Area Network (CAN bus)

A protocol that defines how data is communicated between electronic control units (ECUs). ECUs can be the engine control unit, airbags, or the audio system.

On-board diagnostic (OBD) II

A further developed protocol that defines how self-diagnostic data is communicated between ECUs. It provides a number of standard diagnostic trouble codes (DTCs) that help identify what is wrong with your vehicle.

Vehicle middleware

The vehicle middleware defined as a type of network interface. Examples of vehicle middleware include Robot Operating System (ROS 2) and Scalable service-Oriented MiddlewarE over IP (SOME/IP).

> **Note**
>
> AWS IoT FleetWise supports ROS 2 middleware for vision system data.

Custom interfaces

You can also use your own interface to decode signals at the Edge. This can save you time since you don't need to create decoding rules in the cloud.

Signal decoder

Provides detailed decoding information for a specific signal. Every signal specified in the vehicle model must be paired with a signal decoder. If the decoder manifest contains CAN network interfaces, it must contain CAN decoder signals. If the decoder manifest contains OBD network interfaces, it must contain OBD signal decoders.

The decoder manifest must contain message signal decoders if it also contains vehicle middleware interfaces. Or, if the decoder manifest contains custom decoding interfaces, it must also contain custom decoding signals.

Each decoder manifest must be associated with a vehicle model. AWS IoT FleetWise uses the associated decoder manifest to decode data from vehicles created based on the vehicle model.

Each decoder manifest has a status field that contains the state of the decoder manifest. The state can be one of the following values:

- `ACTIVE` – The decoder manifest is active.
- `DRAFT` – The configuration of the decoder manifest isn't saved.
- `VALIDATING` – The decoder manifest is under validation for its eligibility. This only applies to decoder manifests that contain at least one vision system data signal.
- `INVALID` – The decoder manifest failed validation and can't be activated yet. This only applies to decoder manifests that contain at least one vision system data signal. You can use the ListDecoderManifests and GetDecoderManifest APIs to check the reason for a failed validation.

Important

- If you use the AWS IoT FleetWise console to create a decoder manifest, AWS IoT FleetWise automatically activates the decoder manifest for you.
- If you use the `CreateDecoderManifest` API operation to create a decoder manifest, the decoder manifest stays in the `DRAFT` state.
- You can't create vehicles from vehicle models that are associated with a `DRAFT` decoder manifest. Use the `UpdateDecoderManifest` API operation to change the decoder manifest to the `ACTIVE` state.
- You can't edit decoder manifests that are in the `ACTIVE` state.

Topics

- Delete an AWS IoT FleetWise decoder manifest
- Get AWS IoT FleetWise decoder manifest information

Configure AWS IoT FleetWise network interfaces and decoder signals

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Every decoder manifest has at least a network interface and signal decoders paired with signals specified in the associated vehicle model.

If the decoder manifest contains CAN network interfaces, it must contain CAN signal decoders. If the decoder manifest contains OBD network interfaces, it must contain OBD signal decoders.

Topics

- Configure network interfaces
- Configure signal decoders

Configure network interfaces

To configure a CAN network interface, specify the following information.

- `name` – The CAN interface's name.

  The interface name must be unique and can have 1–100 characters.
- (Optional) `protocolName` – The protocol's name.

  Valid values: `CAN-FD` and `CAN`
- (Optional) `protocolVersion` – AWS IoT FleetWise currently supports CAN-FD and CAN 2.0b.

  Valid values: `1.0` and `2.0b`

To configure an OBD network interface, specify the following information.

- `name` – The OBD interface's name.

  The interface name must be unique and can have 1–100 characters.
- `requestMessageId` – The ID of the message that is requesting data.
- (Optional) `dtcRequestIntervalSeconds` – How often to request diagnostic trouble codes (DTCs) from the vehicle in seconds. For example, if the specified value is 120, the Edge Agent software collects stored DTCs once every 2 minutes.
- (Optional) `hasTransmissionEcu` – Whether the vehicle has a transmission control module (TCM).

  Valid values: `true` and `false`
- (Optional) `obdStandard` – The OBD standard that AWS IoT FleetWise supports. AWS IoT FleetWise currently supports the World Wide Harmonization On-Board Diagnostics (WWH-OBD) ISO15765-4 standard.
- (Optional) `pidRequestIntervalSeconds` – How often to request OBD II PIDs from the vehicle. For example, if the specified value is 120, the Edge Agent software collects OBD II PIDs once every 2 minutes.
- (Optional) `useExtendedIds` – Whether to use extended IDs in the message.

  Valid values: `true` and `false`

To configure a vehicle middleware network interface, specify the following information.

- `name` – The vehicle middleware interface's name.

  The interface name must be unique and can have 1–100 characters.
- `protocolName` – The protocol's name.

  Valid values: `ROS_2`

To configure a custom decoding interface, specify the following information.

- `name` – The name of your decoder that you use to decode signals at the Edge.

  The decoder interface name can have 1–100 characters.

Configure signal decoders

To configure a CAN signal decoder, specify the following information.

- `factor` – The multiplier used to decode the message.
- `isBigEndian` – Whether the byte ordering of the message is big-endian. If it's big-endian, the most significant value in the sequence is stored first, at the lowest storage address.
- `isSigned` – Whether the message is signed. If it's signed, the message can represent both positive and negative numbers.
- `length` – The length of the message in bits.
- `messageId` – The ID of the message.
- `offset` – The offset used to calculate the signal value. Combined with factor, the calculation is `value = raw_value * factor + offset`.
- `startBit` – Indicates the location of the first bit of the message.
- (Optional) `name` – The name of the signal.
- (Optional) `signalValueType` – The value type of the signal. Integer is the default value type.

To configure an OBD signal decoder, specify the following information.

- `byteLength` – The length of the message in bytes.
- `offset` – The offset used to calculate the signal value. Combined with scaling, the calculation is `value = raw_value * scaling + offset`.
- `pid` – The diagnostic code used to request a message from a vehicle for this signal.
- `pidResponseLength` – The length of the requested message.
- `scaling` – The multiplier used to decode the message.
- `serviceMode` – The mode of operation (diagnostic service) in a message.
- `startByte` – Indicates the beginning of the message.
- (Optional) `bitMaskLength` – The number of bits that are masked in a message.
- (Optional) `bitRightShift` – The number of positions shifted to the right.
- (Optional) `isSigned` – Whether the message is signed. If it's signed, the message can represent both positive and negative numbers. The message is not signed by default (`false`).
- (Optional) `signalValueType` – The value type of the signal. Integer is the default value type.

To configure a message signal decoder, specify the following information.

- `topicName` – The topic name for the message signal. It corresponds to topics in ROS 2. For more information about the structured message object, see StructuredMessage.
- `structuredMessage` – The structured message for the message signal. It can be defined with either a primitiveMessageDefinition, structuredMessageListDefinition, or structuredMessageDefinition recursively.

To configure a custom decoding signal, specify the following information.

- (Optional) `id` – The ID of the signal that you decode yourself using your decoder interface. The signal ID can have 1–150 characters. If not specified, the `id` defaults to the `fullyQualifiedName` of the signal.

Create an AWS IoT FleetWise decoder manifest

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the AWS IoT FleetWise console or API to create a decoder manifest for your vehicle model.

Topics

- Create a decoder manifest (console)
- Create a decoder manifest (AWS CLI)

Create a decoder manifest (console)

You can use the AWS IoT FleetWise console to create a decoder manifest that's associated with your vehicle model.

> **Important**
>
> You can't configure vision system data signals in decoder manifests using the AWS IoT FleetWise console. Instead, use the AWS CLI. Vision system data is in preview release and is subject to change.

To create a decoder manifest

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. Choose the target vehicle model.
4. On the vehicle model summary page, choose **Create decoder manifest**, and then do the following.

Topics

Step 1: Configure decoder manifest

In **General information**, do the following.

1. Enter a unique name for the decoder manifest.
2. (Optional) Enter a description.
3. Choose **Next**.

Add network interfaces

Each decoder manifest must have at least one network interface. You can add multiple network interfaces to a decoder manifest.

To add a network interface

1. Upload a network interface file. You can upload a .dbc file for CAN protocols, or a .json file for ROS 2 or custom interfaces.
2. Enter a name for your network interface. If you uploaded a custom interface, the name is already provided.

Map missing signals

If there are signals in the vehicle model that are missing paired signal decoders in the uploaded network interfaces, you can create a default custom decoder that will map the missing signals. This is optional since you can manually map the signals in the next step.

To create a default custom decoder

1. Select **Create default custom decoder for missing signals**.
2. Choose **Next**.

Step 2: Map CAN interface

You can map the CAN signals with CAN signal decoders. If you selected the **Create default custom decoder for missing signals** checkbox, any signals that are missing a decoder signal are automatically mapped to default custom signal decoders.

To map CAN signals

1. In **CAN signal mapping**, select a signal decoder.
2. Choose **Next**.

> **Note**
>
> If you added a ROS 2 or a custom interface, you can verify the mappings before creating the decoder manifest.

Step 3: Review and create

Verify the configurations for the decoder manifest, and then choose **Create**.

Create a decoder manifest (AWS CLI)

You can use the CreateDecoderManifest API operation to create decoder manifests. The following example uses the AWS CLI.

> **Important**
>
> You must have a vehicle model before you can create a decoder manifest. Every decoder manifest must be associated with a vehicle model. For more information, see Create an AWS IoT FleetWise vehicle model.

To create a decoder manifest, run the following command.

Replace *decoder-manifest-configuration* with the name of the .json file that contains the configuration.

```
aws iotfleetwise create-decoder-manifest --cli-input-json file://decoder-manifest-configuration.json
```

- Replace *decoder-manifest-name* with the name of the decoder manifest that you're creating.
- Replace *vehicle-model-ARN* with the Amazon Resource Name (ARN) of the vehicle-model.
- (Optional) Replace *description* with a description to help you identify the decoder manifest.

For more information about how to configure branches, attributes, sensors, and actuators, see Configure AWS IoT FleetWise network interfaces and decoder signals.

```
{
    "name": "decoder-manifest-name",
    "modelManifestArn": "vehicle-model-arn",
    "description": "description",
    "networkInterfaces": [
        {
            "canInterface": {
                "name": "myNetworkInterface",
                "protocolName": "CAN",
                "protocolVersion": "2.0b"
            },
```

```
            "interfaceId": "Qq1acaenByOB3sSM39SYm",
            "type": "CAN_INTERFACE"
        }
    ],
    "signalDecoders": [
        {
            "canSignal": {
                "name": "Engine_Idle_Time",
                "factor": 1,
                "isBigEndian": true,
                "isSigned": false,
                "length": 24,
                "messageId": 271343712,
                "offset": 0,
                "startBit": 16
            },
            "fullyQualifiedName": "Vehicle.EngineIdleTime",
            "interfaceId": "Qq1acaenByOB3sSM39SYm",
            "type": "CAN_SIGNAL"
        },
        {
            "canSignal": {
                "name": "Engine_Run_Time",
                "factor": 1,
                "isBigEndian": true,
                "isSigned": false,
                "length": 24,
                "messageId": 271343712,
                "offset": 0,
                "startBit": 40
            },
            "fullyQualifiedName": "Vehicle.EngineRunTime",
            "interfaceId": "Qq1acaenByOB3sSM39SYm",
            "type": "CAN_SIGNAL"
        }
    ]
}
```

- Replace *`decoder-manifest-name`* with the name of the decoder manifest that you're creating.
- Replace *`vehicle-model-ARN`* with the Amazon Resource Name (ARN) of the vehicle-model.
- (Optional) Replace *`description`* with a description to help you identify the decoder manifest.

The order of property nodes within a structure (struct) must remain consistent as defined in the signal catalog and vehicle model (model manifest). For more information about how to configure branches, attributes, sensors, and actuators, see Configure AWS IoT FleetWise network interfaces and decoder signals.

```
{
 "name": "decoder-manifest-name",
 "modelManifestArn": "vehicle-model-arn",
 "description": "description",
 "networkInterfaces": [{
  "canInterface": {
   "name": "myNetworkInterface",
   "protocolName": "CAN",
   "protocolVersion": "2.0b"
  },
  "interfaceId": "Qq1acaenByOB3sSM39SYm",
  "type": "CAN_INTERFACE"
 }, {
  "type": "VEHICLE_MIDDLEWARE",
  "interfaceId": "G1KzxkdnmV5Hn7wkV3ZL9",
  "vehicleMiddleware": {
   "name": "ROS2_test",
   "protocolName": "ROS_2"
  }
 }],
 "signalDecoders": [{
   "canSignal": {
    "name": "Engine_Idle_Time",
    "factor": 1,
    "isBigEndian": true,
    "isSigned": false,
    "length": 24,
    "messageId": 271343712,
    "offset": 0,
    "startBit": 16
   },
   "fullyQualifiedName": "Vehicle.EngineIdleTime",
   "interfaceId": "Qq1acaenByOB3sSM39SYm",
   "type": "CAN_SIGNAL"
  },
  {
   "canSignal": {
    "name": "Engine_Run_Time",
```

```
    "factor": 1,
    "isBigEndian": true,
    "isSigned": false,
    "length": 24,
    "messageId": 271343712,
    "offset": 0,
    "startBit": 40
   },
   "fullyQualifiedName": "Vehicle.EngineRunTime",
   "interfaceId": "Qq1acaenByOB3sSM39SYm",
   "type": "CAN_SIGNAL"
  },
  {
   "fullyQualifiedName": "Vehicle.CompressedImageTopic",
   "type": "MESSAGE_SIGNAL",
   "interfaceId": "G1KzxkdnmV5Hn7wkV3ZL9",
   "messageSignal": {
    "topicName": "CompressedImageTopic:sensor_msgs/msg/CompressedImage",
    "structuredMessage": {
     "structuredMessageDefinition": [{
       "fieldName": "header",
       "dataType": {
        "structuredMessageDefinition": [{
          "fieldName": "stamp",
          "dataType": {
           "structuredMessageDefinition": [{
             "fieldName": "sec",
             "dataType": {
              "primitiveMessageDefinition": {
               "ros2PrimitiveMessageDefinition": {
                "primitiveType": "INT32"
               }
              }
             }
            },
            {
             "fieldName": "nanosec",
             "dataType": {
              "primitiveMessageDefinition": {
               "ros2PrimitiveMessageDefinition": {
                "primitiveType": "UINT32"
               }
              }
             }
```

```
            }
           ]
          }
         },
         {
          "fieldName": "frame_id",
          "dataType": {
           "primitiveMessageDefinition": {
            "ros2PrimitiveMessageDefinition": {
             "primitiveType": "STRING"
            }
           }
          }
         }
        ]
       }
      },
      {
       "fieldName": "format",
       "dataType": {
        "primitiveMessageDefinition": {
         "ros2PrimitiveMessageDefinition": {
          "primitiveType": "STRING"
         }
        }
       }
      },
      {
       "fieldName": "data",
       "dataType": {
        "structuredMessageListDefinition": {
         "name": "listType",
         "memberType": {
          "primitiveMessageDefinition": {
           "ros2PrimitiveMessageDefinition": {
            "primitiveType": "UINT8"
           }
          }
         },
         "capacity": 0,
         "listType": "DYNAMIC_UNBOUNDED_CAPACITY"
        }
       }
      }
```

```
      ]
     }
    }
   }
  ]
}
```

- Replace *decoder-manifest-name* with the name of the decoder manifest that you're creating.
- Replace *vehicle-model-ARN* with the Amazon Resource Name (ARN) of the vehicle-model.
- (Optional) Replace *description* with a description to help you identify the decoder manifest.

For more information about how to configure branches, attributes, sensors, and actuators, see Configure AWS IoT FleetWise network interfaces and decoder signals.

```
{
 "name": "decoder-manifest-name",
 "modelManifestArn": "vehicle-model-arn",
 "description": "description",
 "networkInterfaces": [
        {
      "interfaceId": "myCustomInterfaceId",
      "type": "CUSTOM_DECODING_INTERFACE",
            "customDecodingInterface": {
                "name": "myCustomInterface"
            }
        }
    ],
    "signalDecoders": [
        {
            "customDecodingSignal": {
                "fullyQualifiedName": "Vehicle.actuator1",
                "interfaceId": "myCustomInterfaceId",
                "type": "CUSTOM_DECODING_SIGNAL",
                "customDecodingSignal": {
                    "id": "Vehicle.actuator1"
                }
            }
        },
        {
            "customDecodingSignal": {
                "fullyQualifiedName": "Vehicle.actuator2",
```

```
                "interfaceId": "myCustomInterfaceId",
                "type": "CUSTOM_DECODING_SIGNAL",
                "customDecodingSignal": {
                    "id": "Vehicle.actuator2"
                }
            }
        }
    ]
}
```

> **Note**
>
> You can download a demo script to create a decoder manifest with vision system signals. For more information, see the *Vision System Data Developer Guide*.
> Vision system data is in preview release and is subject to change.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `CreateDecoderManifest` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Update an AWS IoT FleetWise decoder manifest

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the UpdateDecoderManifest API operation to update a decoder manifest. You can add, remove, and update network interfaces and signal decoders. You can also change the status of the decoder manifest. The following example uses the AWS CLI.

To update a decoder manifest, run the following command.

Replace *decoder-manifest-name* with the name of the decoder manifest that you're updating.

```
aws iotfleetwise update-decoder-manifest /
                --name decoder-manifest-name /
                --status ACTIVE
```

If the signals don't have specified decoding rules, you can create default decoding rules. The signals are added to a custom decoded interface with the `CustomDecodingSignal$id` set to the fully qualified name of the signal. To update a decoder manifest with default decoding rules, run the following command.

Replace *decoder-manifest-name* with the name of the decoder manifest that you're updating.

```
aws iotfleetwise update-decoder-manifest /
                --name decoder-manifest-name /
                --status ACTIVE
                --default-for-unmapped-signals CUSTOM_DECODING
```

> **Important**
>
> After you activate the decoder manifest, you can't edit it.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `UpdateDecoderManifest` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Verify decoder manifest update

You can use the ListDecoderManifestSignals API operation to verify if decoder signals in the decoder manifest were updated. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all decoder signals (nodes) in a given decoder manifest, run the following command.

Replace *decoder-manifest-name* with the name of the decoder manifest that you're checking.

```
aws iotfleetwise list-decoder-manifest-signals /
                --name decoder-manifest-name
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListDecoderManifestSignals` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

You can use the ListDecoderManifestNetworkInterfaces API operation to verify if network interfaces in the decoder manifest were updated. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all network interfaces in a given decoder manifest, run the following command.

Replace *decoder-manifest-name* with the name of the decoder manifest that you're checking.

```
aws iotfleetwise list-decoder-manifest-network-interfaces /
                --name decoder-manifest-name
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListDecoderManifestNetworkInterfaces` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
```

```
        }
    ]
}
```

Delete an AWS IoT FleetWise decoder manifest

You can use the AWS IoT FleetWise console or API to delete a decoder manifest.

> **Important**
>
> Vehicles associated with the decoder manifest must be deleted first. For more information, see Delete an AWS IoT FleetWise vehicle.

Topics

Delete a decoder manifest (console)

You can use the AWS IoT FleetWise console to delete a decoder manifest.

To delete a decoder manifest

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. Choose the target vehicle model.
4. On the vehicle model summary page, choose the **Decoder manifests** tab.
5. Choose the target decoder manifest, and then choose **Delete**.
6. In **Delete `decoder-manifest-name`?**, enter the name of the decoder manifest to delete, and then choose **Confirm**.

Delete a decoder manifest (AWS CLI)

You can use the DeleteDecoderManifest API operation to delete a decoder manifest. The following example uses AWS CLI.

> **Important**
>
> Before you delete the decoder manifest, delete the associated vehicles first. For more information, see Delete an AWS IoT FleetWise vehicle.

To delete a decoder manifest, run the following command.

Replace *decoder-manifest-name* with the name of the decoder manifest that you're deleting.

```
aws iotfleetwise delete-decoder-manifest --name decoder-manifest-name
```

Verify decoder manifest deletion

You can use the ListDecoderManifests API operation to verify if a decoder manifest has been deleted. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all decoder manifests, run the following command.

```
aws iotfleetwise list-decoder-manifests
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListDecoderManifests` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Get AWS IoT FleetWise decoder manifest information

You can use the GetDecoderManifest API operation to verify if network interfaces and signal decoders in the decoder manifest have been updated. The following example uses AWS CLI.

To retrieve information about a decoder manifest, run the following command.

Replace *decoder-manifest* with the name of the decoder manifest that you want to retrieve.

```
aws iotfleetwise get-decoder-manifest --name decoder-manifest
```

> **Note**
>
> This operation is eventually consistent. In other words, changes to the decoder manifest might not be reflected immediately.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `GetDecoderManifest` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Manage AWS IoT FleetWise vehicles

Vehicles are instances of vehicle models. Vehicles must be created from a vehicle model and associated with a decoder manifest. Vehicles uploads one or more data streams to the cloud. For example, a vehicle can send mileage, engine temperature, and state of heater data to the cloud. Every vehicle contains the following information:

`vehicleName`

An ID that identifies the vehicle.

Do not add personally identifiable information (PII) or other confidential or sensitive information in your vehicle name. Vehicle names are accessible by other AWS services, including Amazon CloudWatch. Vehicle names aren't intended to be used for private or sensitive data.

`modelManifestARN`

The Amazon Resource Name (ARN) of a vehicle model (model manifest). Every vehicle is created from a vehicle model. Vehicles created from the same vehicle model consist of the same group of signals inherited from the vehicle model. These signals are defined and standardized in the signal catalog.

`decoderManifestArn`

The ARN of the decoder manifest. A decoder manifest provides decoding information that AWS IoT FleetWise can use to transform raw signal data (binary data) into human-readable values. A decoder manifest must be associated with a vehicle model. AWS IoT FleetWise uses the same decoder manifest to decode raw data from vehicles created based on the same vehicle model.

`attributes`

Attributes are key-value pairs that contain static information. Vehicles can contain attributes inherited from the vehicle model. You can add additional attributes to distinguish an individual vehicle from other vehicles created from the same vehicle model. For example, if you have a black car, you can specify the following value for an attribute: `{"color": "black"}`.

> **Important**
>
> Attributes must be defined in the associated vehicle model before you can add them to individual vehicles.

For more information about vehicle models, decoder manifests, and attributes, see Model AWS IoT FleetWise vehicles.

AWS IoT FleetWise provides the following API operations that you can use to create and manage vehicles.

- CreateVehicle – Creates a new vehicle.
- BatchCreateVehicle – Creates one or more new vehicles.
- UpdateVehicle – Updates an existing vehicle.
- BatchUpdateVehicle – Updates one or more existing vehicles.
- DeleteVehicle – Deletes an existing vehicle.
- ListVehicles – Retrieves a paginated list of summaries of all vehicles.
- GetVehicle – Retrieves information about a vehicle.

Tutorials

- Provision AWS IoT FleetWise vehicles
- Reserved topics in AWS IoT FleetWise
- Create an AWS IoT FleetWise vehicle
- Create multiple AWS IoT FleetWise vehicles
- Update an AWS IoT FleetWise vehicle
- Update multiple AWS IoT FleetWise vehicles
- Delete an AWS IoT FleetWise vehicle
- Get AWS IoT FleetWise vehicle information

Provision AWS IoT FleetWise vehicles

The Edge Agent for AWS IoT FleetWise software running in your vehicle collects and transfers data to the cloud. AWS IoT FleetWise integrates with AWS IoT Core to support secure communication between the Edge Agent software and the cloud through MQTT. Each vehicle corresponds to an AWS IoT thing. You can use an existing AWS IoT thing to create a vehicle or set AWS IoT FleetWise to automatically create an AWS IoT thing for your vehicle. For more information, see Create an AWS IoT FleetWise vehicle.

AWS IoT Core supports authentication and authorization that help securely control access to AWS IoT FleetWise resources. Vehicles can use X.509 certificates to get authenticated (signed in) to use AWS IoT FleetWise and AWS IoT Core policies to get authorized (have permissions) to perform specified actions.

Authenticate vehicles

You can create AWS IoT Core policies to authenticate your vehicles.

To authenticate your vehicle

- To create an AWS IoT Core policy, run the following command.
  - Replace *policy-name* with the name of the policy that you want to create.
  - Replace *file-name* with the name of the JSON file that contains the AWS IoT Core policy.

```
aws iot create-policy --policy-name policy-name --policy-document file://file-name.json
```

Before you use the example policy, do the following:

- Replace *us-east-1* with the AWS Region where you created AWS IoT FleetWise resources.
- Replace *111122223333* with your AWS account ID.

This example includes topics reserved by AWS IoT FleetWise. You must add the topics to the policy. For more information, see Reserved topics in AWS IoT FleetWise.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:111122223333:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:topic/$aws/iotfleetwise/
vehicles/${iot:Connection.Thing.ThingName}/checkins",
                "arn:aws:iot:us-east-1:111122223333:topic/$aws/iotfleetwise/
vehicles/${iot:Connection.Thing.ThingName}/signals"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:topicfilter/$aws/
iotfleetwise/vehicles/${iot:Connection.Thing.ThingName}/collection_schemes",
                "arn:aws:iot:us-east-1:111122223333:topicfilter/$aws/
iotfleetwise/vehicles/${iot:Connection.Thing.ThingName}/decoder_manifests"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:topic/$aws/iotfleetwise/
vehicles/${iot:Connection.Thing.ThingName}/collection_schemes",
                "arn:aws:iot:us-east-1:111122223333:topic/$aws/iotfleetwise/
vehicles/${iot:Connection.Thing.ThingName}/decoder_manifests"
            ]
        }
    ]
}
```

Authorize vehicles

You can create X.509 certificates to authorize your vehicles.

To authorize your vehicle

> **Important**
>
> We recommend that you create a new certificate for each vehicle.

1. To create an RSA key pair and issue an X.509 certificate, run the following command.

   - Replace *cert* with the name of the file that saves the command output contents of certificatePem.
   - Replace *public-key* with the name of the file that saves the command output contents of keyPair.PublicKey.
   - Replace *private-key* with the name of the file that saves the command output contents of keyPair.PrivateKey.

   ```
   aws iot create-keys-and-certificate \
       --set-as-active \
       --certificate-pem-outfile cert.pem \
       --public-key-outfile public-key.key" \
       --private-key-outfile  private-key.key"
   ```

2. Copy the Amazon Resource Name (ARN) of the certificate from the output.
3. To attach the policy to the certificate, run the following command.

   - Replace *policy-name* with the name of the AWS IoT Core policy that you created.
   - Replace *certificate-arn* with the ARN of the certificate that you copied.

   ```
   aws iot attach-policy \
       --policy-name policy-name\
       --target "certificate-arn"
   ```

4. To attach the certificate to the thing, run the following command.

- Replace *thing-name* with the name of your AWS IoT thing or the ID of your vehicle.
- Replace *certificate-arn* with the ARN of the certificate that you copied.

```
aws iot attach-thing-principal \
    --thing-name thing-name \
    --principal "certificate-arn"
```

Reserved topics in AWS IoT FleetWise

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

AWS IoT FleetWise reserves the use of the following topics. If the reserved topic allows, you can subscribe or publish to it. However, you can't create new topics that begin with a dollar sign ($). If you use unsupported publish or subscribe operations with reserved topics, it can result in the connection ending.

Topic	Client operation allowed	Description	
`$aws/iotfleetwise/vehicles/` *vehicleName* `/checkins`	Publish	The Edge Agent software publishes vehicle status information to this topic. Vehicle status information is exchanged in protocol buffers (Protobuf) format. For more informati	

Topic	Client operation allowed	Description
		on, see the *Edge Agent for AWS IoT FleetWise software Developer Guide*.
`$aws/iotfleetwise/vehicles/vehicleName/signals`	Publish	The Edge Agent software publishes signals to this topic. Signal information is exchanged in protocol buffers (Protobuf) format. For more information, see the *Edge Agent for AWS IoT FleetWise software Developer Guide*.
`$aws/iotfleetwise/vehicles/vehicleName/collection_schemes`	Subscribe	AWS IoT FleetWise publishes data collection schemes to this topic. Vehicles consume these data collection schemes.
`$aws/iotfleetwise/vehicles/vehicleName/decoder_manifests`	Subscribe	AWS IoT FleetWise publishes decoder manifests to this topic. Vehicles consume these decoder manifests.

Topic	Client operation allowed	Description
`$aws/iotfleetwise/vehicles/vehicleName/command/request`	Subscribe	AWS IoT FleetWise publishes requests to execute commands to this topic. Vehicles then consume these command requests.
`$aws/iotfleetwise/vehicles/vehicleName/command/response`	Publish	The Edge Agent software publishes command responses from the vehicle to this topic. Command responses are exchanged in protocol buffers (Protobuf) format. For more information, see the *Edge Agent for AWS IoT FleetWise software Developer Guide*.
`$aws/iotfleetwise/vehicles/vehicleName/command/notification`	Subscribe	AWS IoT FleetWise publishes command status updates to this topic. The notifications are sent in a JSON format.

Topic	Client operation allowed	Description
$aws/iotfleetwise/vehicles/*$vehicle_name*/last_known_states/config	Subscribe	AWS IoT FleetWise publishes state template configurations to this topic. Vehicles consume these state template configurations.
$aws/iotfleetwise/vehicles/*$vehicle_name*/last_known_states/data	Publish	The Edge Agent software publishes data collected from the signals to this topic.

Topic	Client operation allowed	Description	
$aws/iotfleetwise/vehicles/*$vehicle_name*/last_known_state/*$state_template_name*/data	Subscribe	AWS IoT FleetWise publishes data collected from the signals configured in the specified *$state_template_name* to this topic. The updates can be partial. For example, if a state template association contains multiple signals with the on-change update strategy, then only the signals that have changed are contained in a given message. Signal information is exchanged in protocol buffers (Protobuf) format. For more information, see the *Edge Agent for AWS IoT FleetWise software Developer Guide*.	

Create an AWS IoT FleetWise vehicle

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the AWS IoT FleetWise console or API to create a vehicle.

Important

Before you start, check the following:

- You must have a vehicle model and the status of the vehicle model must be `ACTIVE`. For more information, see Manage AWS IoT FleetWise vehicle models.
- Your vehicle model must be associated with a decoder manifest, and the status of the decoder manifest must be `ACTIVE`. For more information, see Manage AWS IoT FleetWise decoder manifests.

Topics

Create a vehicle (console)

You can use the AWS IoT FleetWise console to create a vehicle.

To create a vehicle

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicles**.
3. On the vehicle summary page, choose **Create vehicle**, and then do the following steps.

Topics

Step 1: Define vehicle properties

In this step, you name the vehicle and associate it with the model manifest and decoder manifest.

1. Enter a unique name for the vehicle.

   > **Important**
   >
   > A vehicle corresponds to an AWS IoT thing. If a thing already exists with that name, choose **Associate the vehicle with an IoT thing** to update the thing with the vehicle. Or, choose a different vehicle name and AWS IoT FleetWise will automatically create a new thing for the vehicle.

2. Choose a vehicle model (model manifest) from the list.
3. Choose a decoder manifest from the list. The decoder manifest is associated with the vehicle model.
4. (Optional) To associate vehicle attributes, choose **Add attributes**. If you skip this step, you must add attributes after the vehicle is created before you can deploy it to campaigns.
5. (Optional) To associate tags with the vehicle, choose **Add new tag**. You can also add tags after the vehicle is created.
6. Choose **Next**.

Step 2: Configure vehicle certificate

To use your vehicle as an AWS IoT thing, you must configure a vehicle certificate with an attached policy. If you skip this step, you must configure a certificate after the vehicle is created before you can deploy it to campaigns.

1. Choose **Auto-generate a new certificate (recommended)**.
2. Choose **Next**.

Step 3: Attach policies to certificate

Attach a policy to the certificate you configured in the previous step.

1. For **Policies**, enter an existing policy name. To create a new policy, choose **Create policy**.
2. Choose **Next**.

Step 4: Review and create

Verify the configurations for the vehicle, and then choose **Create vehicle**.

> **Important**
>
> After the vehicle is created, you must download the certificate and keys. You'll use the certificate and private key to connect the vehicle in the Edge Agent for AWS IoT FleetWise software.

Create a vehicle (AWS CLI)

When you create a vehicle, you must use a vehicle model that is associated with a decoder manifest. You can use the CreateVehicle API operation to create a vehicle. The following example uses the AWS CLI.

To create a vehicle, run the following command.

Replace *file-name* with the name of the .json file that contains the vehicle configuration.

```
aws iotfleetwise create-vehicle --cli-input-json file://file-name.json
```

Example– vehicle configuration

- (Optional) The `associationBehavior` value can be one of the following:
  - `CreateIotThing` – When your vehicle is created, AWS IoT FleetWise automatically creates an AWS IoT thing with the name of your vehicle ID for your vehicle.
  - `ValidateIotThingExists` – Use an existing AWS IoT thing to create a vehicle.

    To create an AWS IoT thing, run the following command. Replace *thing-name* with the name of the thing you want to create.

```
aws iot create-thing --thing-name thing-name
```

If it's not specified, AWS IoT FleetWise automatically creates an AWS IoT thing for your vehicle.

> **Important**
>
> Make sure that the AWS IoT thing is provisioned after the vehicle is created. For more information, see Provision AWS IoT FleetWise vehicles.

- Replace *vehicle-name* with one of the following.
  - The name of your AWS IoT thing if `associationBehavior` is configured to `ValidateIotThingExists`.
  - The ID of the vehicle to create if `associationBehavior` is configured to `CreateIotThing`.

    The vehicle ID can have 1–100 characters. Valid characters: a–z, A–Z, 0–9, dash (-), underscore (_), and colon (:).
- Replace *model-manifest-ARN* with the ARN of your vehicle model (model manifest).
- Replace *decoder-manifest-ARN* with the ARN of the decoder manifest associated with the specified vehicle model.
- (Optional) You can add additional attributes to distinguish this vehicle from other vehicles created from the same vehicle model. For example, if you have an electric car, you can specify the following value for an attribute: `{"fuelType": "electric"}`.

> **Important**
>
> Attributes must be defined in the associated vehicle model before you can add them to individual vehicles.

```
{
    "associationBehavior": "associationBehavior",
    "vehicleName": "vehicle-name",
    "modelManifestArn": "model-manifest-ARN",
    "decoderManifestArn": "decoder-manifest-ARN",
    "attributes": {
        "key": "value"
```

```
    }
}
```

Example– associate a state template with the vehicle

You can associate state templates with the vehicle to allow collection of state updates from the vehicle in the cloud by using the `stateTemplates` field.

In this example, *`stateTemplateUpdateStrategy`* can be one of:

- `periodic`: allows you to specify a fixed rate at which Edge Agent software will send signal updates to the cloud (Edge Agent software will send updates even if the signal value hasn't changed between updates).
- onChange: Edge Agent software will send signal updates whenever the signal changes.

```
aws iotfleetwise create-vehicle --cli-input-json file://create-vehicle.json
```

Where the *`create-vehicle.json`* file contains (for example):

```
{
    "associationBehavior": "associationBehavior",
    "vehicleName": "vehicle-name",
    "modelManifestArn": "model-manifest-ARN",
    "decoderManifestArn": "decoder-manifest-ARN",
    "attributes": {
        "key": "value"
    },
    "stateTemplates": [
        {
            "identifier": "state-template-name",
            "stateTemplateUpdateStrategy": {
                "periodic": {
                    "stateTemplateUpdateRate": {
                        "unit": "SECOND",
                        "value": 10
                    }
                }
            }
        }
    ]
}
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `CreateVehicle` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Create multiple AWS IoT FleetWise vehicles

You can use the BatchCreateVehicle API operation to create multiple vehicles at one time. The following example uses the AWS CLI.

To create multiple vehicles, run the following command.

Replace *file-name* with the name of the .json file that contains the configurations of multiple vehicles.

```
aws iotfleetwise batch-create-vehicle --cli-input-json file://file-name.json
```

Example– vehicle configurations

```
{
    "vehicles": [
        {
                "associationBehavior": "associationBehavior",
```

```
                "vehicleName": "vehicle-name",
                "modelManifestArn": "model-manifest-ARN",
                "decoderManifestArn": "decoder-manifest-ARN",
                "attributes": {
                    "key": "value"
                }
        },
        {
                "associationBehavior": "associationBehavior",
                "vehicleName": "vehicle-name",
                "modelManifestArn": "model-manifest-ARN",
                "decoderManifestArn": "decoder-manifest-ARN",
                "attributes": {
                    "key": "value"
                }
        }
    ]
}
```

You can create up to 10 vehicles for each batch operation. For more information about the vehicle configuration, see Create an AWS IoT FleetWise vehicle.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `BatchCreateVehicle` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Update an AWS IoT FleetWise vehicle

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the UpdateVehicle API operation to update an existing vehicle. The following example uses the AWS CLI.

To update a vehicle, run the following command.

Replace *file-name* with the name of the .json file that contains the configuration of your vehicle.

```
aws iotfleetwise update-vehicle --cli-input-json file://file-name.json
```

Example– vehicle configuration

- Replace *vehicle-name* with the ID of the vehicle that you want to update.
- (Optional) Replace *model-manifest-ARN* with the ARN of the vehicle model (model manifest) that you use to replace the vehicle model in use.
- (Optional) Replace *decoder-manifest-ARN* with the ARN of your decoder manifest associated with the new vehicle model that you specified.
- (Optional) Replace *attribute-update-mode* with vehicle attributes.
  - `Merge` – Merge new attributes into existing attributes by updating existing attributes with new values and adding new attributes if they don't exist.

    For example, if a vehicle has the following attributes: `{"color": "black", "fuelType": "electric"}`, and you update the vehicle with the following attributes: `{"color": "", "fuelType": "gasoline", "model": "x"}`, the updated vehicle has the following attributes: `{"fuelType": "gasoline", "model": "x"}`.
  - `Overwrite` – Replace existing attributes with new attributes.

    For example, if a vehicle has the following attributes: `{"color": "black", "fuelType": "electric"}`, and you update the vehicle with the `{"model": "x"}` attribute, the updated vehicle has the `{"model": "x"}` attribute.

This is required if attributes are present in the input.

- (Optional) To add new attributes or update existing ones with new values, configure `attributes`. For example, if you have an electric car, you can specify the following value for an attribute: `{"fuelType": "electric"}`.

  To delete attributes, configure `attributeUpdateMode` to `Merge`.

> **Important**
>
> Attributes must be defined in the associated vehicle model before you can add them to individual vehicles.

```
 {
         "vehicleName": "vehicle-name",
         "modelManifestArn": "model-manifest-arn",
         "decoderManifestArn": "decoder-manifest-arn",
         "attributeUpdateMode": "attribute-update-mode"
         }
}
```

Example– add or remove state templates associated with the vehicle

You can associate additional state templates or remove existing associations from the vehicle using the following fields:

- `stateTemplatesToAdd`
- `stateTemplatesToRemove`

```
aws iotfleetwise update-vehicle --cli-input-json file://update-vehicle.json
```

Where the *update-vehicle.json* file contains (for example):

```
{
    "vehicleName": "vehicle-name",
    "modelManifestArn": "model-manifest-arn",
    "decoderManifestArn": "decoder-manifest-arn",
```

```
    "attributeUpdateMode": "attribute-update-mode",
    "stateTemplatesToAdd": [
        {
            "identifier": "state-template-name",
            "stateTemplateUpdateStrategy": {
                "onChange": {}
            }
        }
    ],
    "stateTemplatesToRemove": ["state-template-name"]
}
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `UpdateVehicle` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Update multiple AWS IoT FleetWise vehicles

You can use the BatchUpdateVehicle API operation to update multiple existing vehicles at one time. The following example uses the AWS CLI.

To update multiple vehicles, run the following command.

Replace *file-name* with the name of the .json file that contains the configurations of multiple vehicles.

```
aws iotfleetwise batch-update-vehicle --cli-input-json file://file-name.json
```

Example– vehicle configurations

```
{
   "vehicles": [
      {
         "vehicleName": "vehicle-name",
         "modelManifestArn": "model-manifest-arn",
         "decoderManifestArn": "decoder-manifest-arn",
         "mergeAttributes": true,
         "attributes": {
         "key": "value"
         }
      },
      {
         "vehicleName": "vehicle-name",
         "modelManifestArn": "model-manifest-arn",
         "decoderManifestArn": "decoder-manifest-arn",
         "mergeAttributes": true,
         "attributes": {
         "key": "value"
         }
      }
   ]
}
```

You can update up to 10 vehicles for each batch operation. For more information about the configuration of each vehicle, see Update an AWS IoT FleetWise vehicle.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `BatchUpdateVehicle` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Delete an AWS IoT FleetWise vehicle

You can use the AWS IoT FleetWise console or API to delete vehicles.

> **Important**
>
> After a vehicle is deleted, AWS IoT FleetWise automatically removes the vehicle from the associated fleets and campaigns. For more information, see Manage fleets in AWS IoT FleetWise and Collect AWS IoT FleetWise data with campaigns. However, the vehicle still exists as a thing or is still associated with a thing in AWS IoT Core. For instructions on deleting a thing, see Delete a thing in the *AWS IoT Core Developer Guide*.

Delete a vehicle (console)

You can use the AWS IoT FleetWise console to delete a vehicle.

To delete a vehicle

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicles**.
3. On the **Vehicles** page, select the button next to the vehicle you want to delete.
4. Choose **Delete**.
5. In **Delete `vehicle-name`**, enter the name of the vehicle, and then choose **Delete**.

Delete a vehicle (AWS CLI)

You can use the DeleteVehicle API operation to delete a vehicle. The following example uses AWS CLI.

To delete a vehicle, run the following command.

Replace *vehicle-name* with the ID of the vehicle that you want to delete.

```
aws iotfleetwise delete-vehicle --vehicle-name vehicle-name
```

Verify vehicle deletion

You can use the ListVehicles API operation to verify if a vehicle was deleted. The following example uses the AWS CLI.

To retrieve a paginated list of summaries of all vehicles, run the following command.

```
aws iotfleetwise list-vehicles
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListVehicles` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Get AWS IoT FleetWise vehicle information

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the GetVehicle API operation to retrieve vehicle information. The following example uses the AWS CLI.

To retrieve the metadata of a vehicle, run the following command.

Replace *vehicle-name* with the ID of the vehicle that you want to retrieve.

```
aws iotfleetwise get-vehicle --vehicle-name vehicle-name
```

Note

This operation is eventually consistent. In other words, changes to the vehicle might not be reflected immediately.

You can use the GetVehicleStatus API operation to retrieve the status of resources associated with a vehicle. The following example uses the AWS CLI.

To retrieve the status of resources associated with a vehicle, run the following command.

- Replace *vehicle-name* with the ID of the vehicle which the resources are associated with.
- Replace *type* with the type of the resource whose status you want to retrieve. Valid values for `type` are `CAMPAIGN`, `STATE_TEMPLATE`, and `DECODER`.

```
aws iotfleetwise get-vehicle-status --vehicle-name vehicle-name --type type
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `GetVehicle` or `GetVehicleStatus` API operations.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Manage fleets in AWS IoT FleetWise

A fleet represents a group of vehicles. A fleet without associated vehicles is an empty entity. Before you can use the fleet to manage multiple vehicles at the same time, you must associate vehicles with the fleet. A vehicle can belong to multiple fleets. You can control what data to collect from a fleet of vehicles and when to collect data by deploying a campaign. For more information, see Collect AWS IoT FleetWise data with campaigns.

A fleet contains the following information.

`fleetId`

The ID of the fleet.

(Optional) `description`

A description that helps you find the fleet.

`signalCatalogArn`

The Amazon Resource Name (ARN) of the signal catalog.

AWS IoT FleetWise provides the following API operations that you can use to create and manage fleets.

- CreateFleet – Creates a group of vehicles that contain the same group of signals.
- AssociateVehicleFleet – Associates a vehicle to a fleet.
- DisassociateVehicleFleet – Disassociates a vehicle from a fleet.
- UpdateFleet – Updates the description for an existing fleet.
- DeleteFleet – Deletes an existing fleet.
- ListFleets – Retrieves a paginated list of summaries of all fleets.
- ListFleetsForVehicle – Retrieves a paginated list of IDs of all fleets that the vehicle belongs to.
- ListVehiclesInFleet – Retrieves a paginated list of summaries of all vehicles in a fleet.
- GetFleet – Retrieves information about a fleet.

Topics

- Create an AWS IoT FleetWise fleet
- Associate an AWS IoT FleetWise vehicle with a fleet
- Disassociate an AWS IoT FleetWise vehicle from a fleet
- Update an AWS IoT FleetWise fleet
- Delete an AWS IoT FleetWise fleet
- Get AWS IoT FleetWise fleet information

Create an AWS IoT FleetWise fleet

You can use the CreateFleet API operation to create a vehicle fleet. The following example uses AWS CLI.

> **Important**
>
> You must have a signal catalog before you can create a fleet. For more information, see Create an AWS IoT FleetWise signal catalog.

To create a fleet, run the following command.

- Replace *fleet-id* with the ID of the fleet that you're creating.

  The fleet ID must be unique and have 1-100 characters. Valid characters: letters (A-Z and a-z), numbers (0-9), colons (:), dashes (-), and underscores (_).
- (Optional) Replace *description* with a description.

  The description can have 1-2048 characters.
- Replace *signal-catalog-arn* with the ARN of the signal catalog.

```
aws iotfleetwise create-fleet \
   --fleet-id fleet-id \
   --description description \
   --signal-catalog-arn signal-catalog-arn
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `CreateFleet` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Associate an AWS IoT FleetWise vehicle with a fleet

You can use the AssociateVehicleFleet API operation to associate a vehicle with a fleet. The following example uses AWS CLI.

Important

- You must have a vehicle and a fleet before you can associate a vehicle with a fleet. For more information, see Manage AWS IoT FleetWise vehicles.
- If you associate a vehicle with a fleet that is targeted by a campaign, AWS IoT FleetWise automatically deploys the campaign to the vehicle.

To associate a vehicle with a fleet, run the following command.

- Replace *fleet-id* with the ID of the fleet.
- Replace *vehicle-name* with the ID of the vehicle.

```
aws iotfleetwise associate-vehicle-fleet --fleet-id fleet-id --vehicle-name vehicle-name
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `AssociateVehicleFleet` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Disassociate an AWS IoT FleetWise vehicle from a fleet

You can use the DisassociateVehicleFleet API operation to disassociate a vehicle from a fleet. The following example uses AWS CLI.

To disassociate a vehicle with a fleet, run the following command.

- Replace *fleet-id* with the ID of the fleet.
- Replace *vehicle-name* with the ID of the vehicle.

```
aws iotfleetwise disassociate-vehicle-fleet --fleet-id fleet-id --vehicle-name vehicle-name
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `DisassociateVehicleFleet` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Update an AWS IoT FleetWise fleet

You can use the UpdateFleet API operation to update the description for a fleet. The following example uses AWS CLI.

To update a fleet, run the following command.

- Replace *fleet-id* with the ID of the fleet that you're updating.
- Replace *description* with a new description.

  The description can have 1-2048 characters.

```
aws iotfleetwise update-fleet --fleet-id fleet-id --description description
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `UpdateFleet` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Delete an AWS IoT FleetWise fleet

You can use the DeleteFleet API operation to delete a fleet. The following example uses AWS CLI.

> **Important**
>
> Before you delete a fleet, make sure it has no associated vehicles. For instructions on how to disassociate a vehicle from a fleet, see Disassociate an AWS IoT FleetWise vehicle from a fleet.

To delete a fleet, run the following command.

Replace *fleet-id* with the ID of the fleet that you're deleting.

```
aws iotfleetwise delete-fleet --fleet-id fleet-id
```

Verify fleet deletion

You can use the ListFleets API operation to verify if a fleet was deleted. The following example uses the AWS CLI.

To retrieve a paginated list of summaries of all fleets, run the following command.

```
aws iotfleetwise list-fleets
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListFleets` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Get AWS IoT FleetWise fleet information

You can use the ListFleetsForVehicle API operation to retrieve a paginated list of IDs of all fleets that the vehicle belongs to. The following example uses the AWS CLI.

To retrieve a paginated list of IDs of all fleets that the vehicle belongs to, run the following command.

Replace *vehicle-name* with the ID of the vehicle.

```
aws iotfleetwise list-fleets-for-vehicle \
           --vehicle-name vehicle-name
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListFleetsForVehicle` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

You can use the ListVehiclesInFleet API operation to retrieve a paginated list of summaries of all vehicles in a fleet. The following example uses the AWS CLI.

To retrieve a paginated list of summaries of all vehicles in a fleet, run the following command.

Replace *fleet-id* with the ID of the fleet.

```
aws iotfleetwise list-vehicles-in-fleet \
           --fleet-id fleet-id
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `ListVehiclesInFleet` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
```

```
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

You can use the GetFleet API operation to retrieve fleet information. The following example uses the AWS CLI.

To retrieve the metadata of a fleet, run the following command.

Replace *fleet-id* with the ID of the fleet.

```
aws iotfleetwise get-fleet \
           --fleet-id fleet-id
```

> **Note**
>
> This operation is eventually consistent. In other words, changes to the fleet might not be reflected immediately.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `GetFleet` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
```

```
    ]
}
```

Collect AWS IoT FleetWise data with campaigns

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

A campaign is an orchestration of data collection rules. Campaigns give the Edge Agent for AWS IoT FleetWise software instructions on how to select, collect, and transfer data to the cloud.

You create campaigns in the cloud. After you or your team has approved a campaign, AWS IoT FleetWise sets the campaign as ready to deploy, and it will be deployed on the next vehicle check-in. You can choose to deploy a campaign to a vehicle or a fleet of vehicles. The Edge Agent software doesn't start collecting data until a running campaign is deployed to the vehicle.

Important

Campaigns won't work until you have the following.

- The Edge Agent software is running in your vehicle. For more information about how to develop, install, and work with the Edge Agent software, do the following.
  1. Open the AWS IoT FleetWise console.
  2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.
- You've set up AWS IoT Core to provision your vehicle. For more information, see Provision AWS IoT FleetWise vehicles.

Note

You can also Monitor the last known state of your vehicles (not fleets) in near real time using state templates that allow you to stream telemetry data with either an "On Change" or "Periodic" update strategy. The capability also provides "On Demand" features to activate or deactivate previously deployed templates or request the current vehicle state one-time (fetch).

Access to last known state is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Each campaign contains the following information.

`signalCatalogArn`

The Amazon Resource Name (ARN) of the signal catalog associated with the campaign.

(Optional) `tags`

Tags are metadata that can be used to manage the campaign. You can assign the same tag to resources from different services to indicate that the resources are related.

`TargetArn`

The ARN of a vehicle or fleet to which the campaign is deployed.

`name`

A unique name that helps identify the campaign.

`collectionScheme`

The data collection schemes give Edge Agent software instructions on what data to collect or when to collect it. AWS IoT FleetWise currently supports the condition-based collection scheme and the time-based collection scheme.

- `conditionBasedCollectionScheme` – the condition-based collection scheme uses a logical expression to recognize what data to collect. The Edge Agent software collects data when the condition is met.
  - `expression` – the logical expression used to recognize what data to collect. For example, if the `$variable.`myVehicle.InVehicleTemperature` > 50.0` expression is specified, the Edge Agent software collects temperature values that are greater than 50.0. For instructions on how to write expressions, see Logical expressions for AWS IoT FleetWise campaigns.
  - (Optional) `conditionLanguageVersion` – the version of the conditional expression language.
  - (Optional) `minimumTriggerIntervalMs` – the minimum duration of time between two data collection events, in milliseconds. If a signal changes often, you might collect data at a slower rate.

- (Optional) `triggerMode` – can be one of the following values:
  - `RISING_EDGE` – the Edge Agent software collects data only when the condition is met for the first time. For example, `$variable.`myVehicle.AirBagDeployed` == true`.
  - `ALWAYS` – Edge Agent software collects data whenever the condition is met.
- `timeBasedCollectionScheme` – when you define a time-based collection scheme, specify a time period in milliseconds. The Edge Agent software uses the time period to decide how often to collect data. For example, if the time period is 120,000 milliseconds, the Edge Agent software collects data once every two minutes.
  - `periodMs` – the time period (in milliseconds) to decide how often to collect data.

(Optional) `compression`

To save wireless bandwidth and reduce network traffic, you can specify SNAPPY to compress data in vehicles.

By default (OFF), the Edge Agent software doesn't compress data.

`dataDestinationConfigs`

Choose the single destination where the campaign will transfer vehicle data. You can send the data to an MQTT topic, or store it in Amazon S3 or Amazon Timestream.

MQTT (Message Queuing Telemetry Transport) is a lightweight and widely adopted messaging protocol. You can publish data to an MQTT topic to stand up your own event-driven architectures using AWS IoT rules. AWS IoT support for MQTT is based on the MQTT v3.1.1 specification and the MQTT v5.0 specification, with some differences. For more information, see MQTT differences.

S3 can be a cost-effective data storage mechanism that offers durable data management capabilities and downstream data services. You can use S3 for data related to driving behaviors or analyzing long-term maintenance.

Timestream is a data persistence mechanism that can help you identify trends and patterns in near real time. You can use Timestream for time-series data, such as to analyze historical trends in vehicle speed or braking.

> **Note**
>
> Amazon Timestream is not available in the Asia Pacific (Mumbai) Region.

(Optional) `dataExtraDimensions`

You can add one or more attributes to provide additional information for a signal.

(Optional) `dataPartitions`

Create a data partition to temporarily store signal data on a vehicle. You configure when and how to forward the data to the cloud.

- Specify how AWS IoT FleetWise stores the data on a vehicle or fleet by defining the maximum storage size, minimum time to live, and storage location.
- The campaign `spoolingMode` must be `TO_DISK`.
- Uploading configurations include defining the version of the condition language and the logical expression.

(Optional) `description`

Add a description to help identify the campaign's purpose.

(Optional) `diagnosticsMode`

When the diagnostics mode is configured to `SEND_ACTIVE_DTCS`, the campaign sends stored, standard diagnostic trouble codes (DTCs) that help identify what is wrong with your vehicle. For example, P0097 indicates the engine control module (ECM) has determined that the intake air temperature sensor 2 (IAT2) input is lower than the normal sensor range.

By default (`OFF`), the Edge Agent software doesn't send diagnostic codes.

(Optional) `expiryTime`

Define the expiration date for your campaign. When the campaign expires, the Edge Agent software stops collecting data as specified in this campaign. If multiple campaigns are deployed to the vehicle, the Edge Agent software uses other campaigns to collect data.

Default value: `253402243200` (December 31, 9999, 00:00:00 UTC)

(Optional) `postTriggerCollectionDuration`

You can define a post-trigger collection duration, so that the Edge Agent software continues collecting data for a specified period after a scheme is invoked. For example, if a condition-based collection scheme with the following expression is invoked: `$variable.`myVehicle.Engine.RPM` > 7000.0`, the Edge Agent software continues to collect revolutions per minute (RPM) values for the engine. Even if the RPM only goes higher

than 7000 once, it might indicate that there's a mechanical issue. In this case, you might want the Edge Agent software to continue collecting data to help monitor the condition.

Default value: `0`

(Optional) `priority`

Specify an integer to indicate the priority level of the campaign. Campaigns with a smaller number are higher priorities. If you deploy multiple campaigns to a vehicle, the campaigns that are higher priorities are initiated first.

Default value: `0`

(Optional) `signalsToCollect`

A list of signals from which data is collected when the data collection scheme is invoked.

- `name` – the name of the signal from which data is collected when the data collection scheme is invoked.
- `dataPartitionId` – the ID of the data partition to use in the signal. The ID must match one of the IDs provided in `dataPartitions`. If you upload a signal as a condition in your data partition, then those same signals must be included in `signalsToCollect`.
- (Optional) `maxSampleCount` – the maximum number of data samples that the Edge Agent software collects and transfers to the cloud when the data collection scheme is invoked.
- (Optional) `minimumSamplingIntervalMs` – the minimum duration of time between two data sample collection events, in milliseconds. If a signal changes often, you can use this parameter to collect data at a slower rate.

  Valid range: 0-4294967295

(Optional) `spoolingMode`

If `spoolingMode` is configured to `TO_DISK`, the Edge Agent software temporarily stores data locally when a vehicle isn't connected to the cloud. After the connection is reestablished, the data stored locally is automatically transferred to the cloud.

Default value: `OFF`

(Optional) `startTime`

An approved campaign is activated at the start time.

Default value: `0`

The status of a campaign can be one of the following values.

- `CREATING` – AWS IoT FleetWise is processing your request to create the campaign.
- `WAITING_FOR_APPROVAL` – After a campaign is created, it enters the `WAITING_FOR_APPROVAL` state. To approve the campaign, use the `UpdateCampaign` API operation. After the campaign is approved, AWS IoT FleetWise automatically deploys the campaign to the target vehicle or fleet. For more information, see Update an AWS IoT FleetWise campaign.
- `RUNNING` – The campaign is active.
- `SUSPENDED` – The campaign is suspended. To resume the campaign, use the `UpdateCampaign` API operation.

AWS IoT FleetWise provides the following API operations that you can use to create and manage campaigns.

- CreateCampaign – Creates a new campaign.
- UpdateCampaign – Updates an existing campaign. After a campaign is created, you must use this API operation to approve the campaign.
- DeleteCampaign – Deletes an existing campaign.
- ListCampaigns – Retrieves a paginated list of summaries for all campaigns.
- GetCampaign – Retrieves information about a campaign.

Tutorials

- Create an AWS IoT FleetWise campaign
- Update an AWS IoT FleetWise campaign
- Delete an AWS IoT FleetWise campaign
- Get AWS IoT FleetWise campaign information
- Store and forward campaign data
- Collect diagnostic trouble code data using AWS IoT FleetWise
- Visualize AWS IoT FleetWise vehicle data

Create an AWS IoT FleetWise campaign

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the AWS IoT FleetWise console or API to create campaigns to collect vehicle data.

> **Important**
>
> For your campaign to work, you must have the following:
>
> - The Edge Agent software is running in your vehicle. For more information about how to develop, install, and work with the Edge Agent software, do the following:
>
> 1. Open the AWS IoT FleetWise console.
> 2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.
>
> - You've set up AWS IoT Core to provision your vehicle. For more information, see Provision AWS IoT FleetWise vehicles.

Topics

Create a campaign (console)

Use the AWS IoT FleetWise console to create a campaign to select, collect, and transfer vehicle data to the cloud.

To create a campaign

1. Open the AWS IoT FleetWise console.

2. On the navigation pane, choose **Campaigns**.
3. On the **Campaigns** page, choose **Create campaign**, and then complete the steps in the following topics.

Topics

> **Important**
>
> - You must have a signal catalog and a vehicle before you create a campaign. For more information, see Manage AWS IoT FleetWise signal catalogs and Manage AWS IoT FleetWise vehicles.
> - After a campaign is created, you must approve the campaign. For more information, see Update an AWS IoT FleetWise campaign.

Step 1: Configure campaign

In **General information**, do the following:

1. Enter a name for the campaign.
2. (Optional) Enter a description.

Configure the campaign's data collection scheme. A data collection scheme gives the Edge Agent software instructions on what data to collect or when to collect it. In the AWS IoT FleetWise console, you can configure a data collection scheme in the following ways:

- Manually define the data collection scheme.
- Upload a file to automatically define the data collection scheme.

In **Configuration option**, choose one of the following:

- To manually specify the type of data collection scheme and define options to customize the scheme, choose **Define data collection scheme**.

  Manually specify the type of data collection scheme and define options to customize the scheme.

  1. In the **Data collection scheme details** section, choose the type of data collection scheme you want this campaign to use. To use a logical expression to recognize what vehicle data to collect, choose **Condition-based**. To use a specific time period to decide how often to collect vehicle data, choose **Time-based**.
  2. Define the duration of time the campaign collects data.

     > **Note**
     >
     > By default, an approved campaign is activated immediately and doesn't have a set end time. To avoid extra charges, you must specify a time range.

  3. If you specified a condition-based data collection scheme, you must define a logical expression to recognize what data to collect. AWS IoT FleetWise uses a logical expression to recognize what data to collect for a condition-based scheme. The expression must specify a signal's fully qualified name as a variable, a comparison operator, and a comparison value.

     For example, if you specify the `$variable.`myVehicle.InVehicleTemperature` > 50.0` expression, AWS IoT FleetWise collects temperature values that are greater than 50.0. For instructions about how to write expressions, see Logical expressions for AWS IoT FleetWise campaigns.

     Enter the logical expression used to recognize what data to collect.
  4. (Optional) Specify the language version of the conditional expression. The default value is 1.
  5. (Optional) Specify the minimum trigger interval, which is the smallest duration of time between two data collection events. For example, if a signal changes often, you might want to collect data at a slower rate.
  6. Specify the **Trigger mode** condition for the Edge Agent software to collect data. By default, the Edge Agent for AWS IoT FleetWise software **Always** collects data whenever the condition is met. Or, it can collect data only when the condition is met for the first time, **On first trigger**.

7. If you specified a time-based data collection scheme, you must specify a time **Period**, in milliseconds, from 10,000 - 60,000 milliseconds. The Edge Agent software uses the time period to decide how often to collect data.
8. (Optional) Edit the scheme's **Advanced scheme options**.

    a. To save wireless bandwidth and reduce network traffic by compressing data, choose **Snappy**.

    b. (Optional) To define how long, in milliseconds, to continue collecting data after a data collection event, you can specify the **Post trigger collection duration**.

    c. (Optional) To indicate the priority level of the campaign, specify the campaign **Priority**. Campaigns with a smaller number for priority are deployed first and are considered to have a higher priority.

    d. The Edge Agent software can temporarily store data locally when a vehicle isn't connected to the cloud. After the connection is reestablished, the data stored locally is automatically transferred to the cloud. Specify if you want the Edge Agent to **Store data locally** during a lost connection.

    e. (Optional) To provide additional information for a signal, add up to five attributes as **Extra data dimensions**.

- To upload a file to define the data collection scheme, select **Upload a .json file from your local device**. AWS IoT FleetWise automatically defines which options that you can define in the file. You can review and update the selected options.

  Upload a .json file with details about the data collection scheme.

  1. To import information about the data collection scheme, choose **Choose files**. For more information about the required file format, see the CreateCampaign API documentation.

     > **Note**
     >
     > AWS IoT FleetWise currently supports the .json file format extension.

  2. AWS IoT FleetWise automatically defines the data collection scheme based on the information in your file. Review the options that AWS IoT FleetWise selected for you. You can update the options, if needed.

Step 2: Specify storage and upload conditions

To choose if the Edge Agent software will temporarily store data locally when a vehicle isn't connected to the cloud, specify the spooling mode.

- In **Data spooling mode**, choose one of the following:
  - **Not stored** – The Edge Agent software collects but doesn't temporarily store data locally when a vehicle is offline. The Edge Agent software transfers data to the cloud when the vehicle reconnects.
  - **Stored to disk** – The Edge Agent software collects and temporarily stores data locally when a vehicle is offline. Collected data is temporarily stored at a location defined by the Edge Agent config file “persistency” section. The Edge Agent transfers data to the cloud when the vehicle reconnects.
  - **Stored to disk with partitions** – The vehicle always temporarily stores data on the Edge in your specified data partition. You can choose when you want to forward your stored data to the cloud.
    1. (Optional) Enter a partition ID to designate a particular set of data.
    2. Enter a folder name as the location where data will be stored. The absolute path of the storage location is `{persistency_path} / {vehicle_name} / {campaign_name} / {storage_location}`.
    3. Enter the maximum storage size of the data stored in the partition. Newer data overwrites older data when the partition reaches the maximum size.
    4. Enter the minimum amount of time that data in this partition will be kept on the disk.
    5. (Optional) Enter upload conditions for the partition.

Specify signals

You can specify the signals to collect data from during the campaign.

To specify the signals to collect data from

1. Select the **Signal name**.
2. (Optional) For **Max sample count**, enter the maximum number of data samples that the Edge Agent software collects and transfers to the cloud during the campaign.

3. (Optional) For **Min sampling interval**, enter the minimum duration of time between two data sample collection events, in milliseconds. If a signal changes often, you can use this parameter to collect data at a slower rate.
4. To add another signal, choose **Add more signals**. You can add up to 999 signals.
5. Choose **Next**.

Step 3: Configure data destination

> **Note**
>
> If the campaign contains vision system data signals, you can only store the vehicle data in Amazon S3. You can't store it in Timestream or send it to an MQTT topic.
> Vision system data is in preview release and is subject to change.
> Amazon Timestream is not available in the Asia Pacific (Mumbai) Region.

Choose the destination where you want to send or store data collected by the campaign. You can send vehicle data to an MQTT topic, or store it in Amazon S3 or Amazon Timestream.

In **Destination settings**, do the following:

- Choose Amazon S3, Amazon Timestream, or MQTT topic from the dropdown list.

Amazon S3

> **Important**
>
> You can only transfer data to S3 if AWS IoT FleetWise has permissions to write into the S3 bucket. For more information about granting access, see Controlling access with AWS IoT FleetWise.

To store vehicle data in an S3 bucket, choose **Amazon S3**. S3 is an object storage service that stores data as objects within buckets. For more information, see Creating, configuring, and working with Amazon S3 buckets in the *Amazon Simple Storage Service User Guide*.

S3 optimizes the cost of data storage and provides additional mechanisms to use vehicle data, such as data lakes, centralized data storage, data processing pipelines, and analytics. You can use S3 to

store data for batch processing and analysis. For example, you can create reports of hard-braking events for your machine learning (ML) model. Incoming vehicle data is buffered for 10 minutes before delivery.

In **S3 destination settings**, do the following:

1. For **S3 bucket**, choose a bucket that AWS IoT FleetWise has permissions to.
2. (Optional) Enter a custom prefix that you can use to organize data stored in the S3 bucket.
3. Choose the output format, which is the format files that are saved as in the S3 bucket.
4. Choose if you want to compress data stored in the S3 bucket as a .gzip file. We recommend compressing data because it minimizes storage costs.
5. The options you select in **S3 destination settings** change the **Example S3 object URI**. This is an example of what files are saved as in S3.

Amazon Timestream

> **Important**
>
> You can only transfer data to a table if AWS IoT FleetWise has permissions to write data into Timestream. For more information about granting access, see Controlling access with AWS IoT FleetWise.
> Amazon Timestream is not available in the Asia Pacific (Mumbai) Region.

To store vehicle data in a Timestream table, choose **Amazon Timestream**. You can use Timestream to query vehicle data so that you can identify trends and patterns. For example, you can use Timestream to create an alarm for vehicle fuel level. Incoming vehicle data is transferred to Timestream in near real time. For more information, see What is Amazon Timestream? in the *Amazon Timestream Developer Guide*.

In **Timestream table settings**, do the following:

1. For **Timestream database name**, choose the name of your Timestream database from the dropdown list.
2. For **Timestream table name**, choose the name of your Timestream table from the dropdown list.

In **Service access for Timestream**, do the following:

- Choose an IAM role from the dropdown list.

MQTT topic

> **Important**
>
> You can only route data to an MQTT topic if AWS IoT FleetWise has permissions to AWS IoT topics. For more information about granting access, see Controlling access with AWS IoT FleetWise.

To send vehicle data to an MQTT topic, choose **MQTT topic**.

Vehicle data sent by MQTT messaging is delivered in near real-time and allows you to use rules to take action, or route data to other destinations. For more information about using MQTT, see Device communication protocols and Rules for AWS IoT in the *AWS IoT Core Developer Guide*.

1. Under **MQTT topic**, enter the **Topic name**.
2. Under **Service access for MQTT topic**, choose whether you want to let AWS IoT FleetWise **Create and use a new service role** for you. If you want to **Use an existing service role**, select the role in the dropdown list under **Select role**.

- Choose **Next**.

Step 4: Add vehicles

To choose which vehicles to deploy your campaign to, select them in the vehicles list. Filter vehicles by searching for the attributes and their values that you added when creating the vehicles, or by vehicle name.

In **Filter vehicles**, do the following:

1. In the search box, find the attribute or vehicle name and choose it from the list.

> **Note**
>
> Each attribute can be used only once.

2. Enter the value of the attribute or the vehicle name that you want to deploy the campaign to. For example, if the fully qualified name of the attribute is `fuelType`, enter `gasoline` as its value.
3. To search for another vehicle attribute, repeat the preceding steps. You can search for up to five vehicle attributes and an unlimited number of vehicle names.
4. Vehicles that match your search are listed under **Vehicle name**. Choose the vehicles that you want the campaign to deploy to.

> **Note**
>
> Up to 100 vehicles are displayed in search results. Choose **Select all** to add all vehicles to the campaign.

5. Choose **Next**.

Step 5: Review and create

Verify the configurations for the campaign, and then choose **Create campaign**.

> **Note**
>
> After a campaign is created, you or your team must deploy the campaign to vehicles.

Step 6: Deploy a campaign

After you create a campaign, you or your team must deploy the campaign to vehicles.

To deploy a campaign

1. On the **Campaign summary** page, choose **Deploy**.
2. Review and confirm that you want to start the deployment and begin collecting data from vehicles connected to the campaign.

3. Choose **Deploy**.

If you want to pause collecting data from vehicles connected to the campaign, on the **Campaign summary** page, choose **Suspend**. To resume collecting data from vehicles connected to the campaign, choose **Resume**.

Create a campaign (AWS CLI)

You can use the CreateCampaign API operation to create a campaign. The following example uses the AWS CLI.

When you create a campaign, data collected from vehicles can be sent to an MQTT topic or stored in either Amazon S3 (S3) or Amazon Timestream. Choose Timestream for a fast, scalable, and server-less time series database, such as to store data that requires near real time processing. Choose S3 for object storage with industry-leading scalability, data availability, security, and performance. Choose MQTT to deliver data in near real-time and to use Rules for AWS IoT to perform actions you define or route the data to other destinations.

> **Important**
>
> You can only transfer vehicle data to an MQTT topic, Amazon S3, or Amazon Timestream if AWS IoT FleetWise has permissions to send MQTT messages on your behalf, or to write data into S3 or Timestream. For more information about granting access, see Controlling access with AWS IoT FleetWise.
> Amazon Timestream is not available in the Asia Pacific (Mumbai) Region.

Create campaign

> **Important**
>
> - You must have a signal catalog and a vehicle or fleet before you create a campaign. For more information, see Manage AWS IoT FleetWise signal catalogs, Manage AWS IoT FleetWise vehicles, and Manage fleets in AWS IoT FleetWise.

- After a campaign is created, you must use the `UpdateCampaign` API operation to approve the campaign. For more information, see Update an AWS IoT FleetWise campaign

To create a campaign, run the following command.

Replace *file-name* with the name of the .json file that contains the campaign configuration.

```
aws iotfleetwise create-campaign --cli-input-json file://file-name.json
```

- Replace *campaign-name* with the name of the campaign that you're creating.
- Replace *signal-catalog-arn* with the Amazon Resource Name (ARN) of the signal catalog.
- Replace *target-arn* with the ARN of a fleet or vehicle that you created.
- Replace *bucket-arn* with the ARN of the S3 bucket.

```
{
    "name": "campaign-name",
    "targetArn": "target-arn",
    "signalCatalogArn": "signal-catalog-arn",
    "collectionScheme": {
        "conditionBasedCollectionScheme": {
            "conditionLanguageVersion": 1,
            "expression": "$variable.`Vehicle.DemoBrakePedalPressure` > 7000",
            "minimumTriggerIntervalMs": 1000,
            "triggerMode": "ALWAYS"
        }
    },
    "compression": "SNAPPY",
    "diagnosticsMode": "OFF",
    "postTriggerCollectionDuration": 1000,
    "priority": 0,
    "signalsToCollect": [
        {
         "maxSampleCount": 100,
         "minimumSamplingIntervalMs": 0,
         "name": "Vehicle.DemoEngineTorque"
        },
        {
```

```
          "maxSampleCount": 100,
          "minimumSamplingIntervalMs": 0,
          "name": "Vehicle.DemoBrakePedalPressure"
         }
    ],
    "spoolingMode": "TO_DISK",
    "dataDestinationConfigs": [
        {
         "s3Config": {
             "bucketArn": "bucket-arn",
             "dataFormat": "PARQUET",
             "prefix": "campaign-name",
              "storageCompressionFormat": "GZIP"
      }
    }
  ],
     "dataPartitions": [
      { ...  }
  ]
}
```

Note

Amazon Timestream is not available in the Asia Pacific (Mumbai) Region.

- Replace *campaign-name* with the name of the campaign that you're creating.
- Replace *signal-catalog-arn* with the ARN of the signal catalog.
- Replace *target-arn* with the ARN of a fleet or vehicle that you created.
- Replace *role-arn* with the ARN of the task execution role that grants AWS IoT FleetWise permission to deliver data to the Timestream table.
- Replace *table-arn* with the ARN of the Timestream table.

```
{
  "name": "campaign-name",
  "targetArn": "target-arn",
  "signalCatalogArn": "signal-catalog-arn",
  "collectionScheme": {
    "conditionBasedCollectionScheme": {
```

```
      "conditionLanguageVersion": 1,
      "expression": "$variable.`Vehicle.DemoBrakePedalPressure` > 7000",
      "minimumTriggerIntervalMs": 1000,
      "triggerMode": "ALWAYS"
    }
  },
  "compression": "SNAPPY",
  "diagnosticsMode": "OFF",
  "postTriggerCollectionDuration": 1000,
  "priority": 0,
  "signalsToCollect": [
    {
      "maxSampleCount": 100,
      "minimumSamplingIntervalMs": 0,
      "name": "Vehicle.DemoEngineTorque"
    },
    {
      "maxSampleCount": 100,
      "minimumSamplingIntervalMs": 0,
      "name": "Vehicle.DemoBrakePedalPressure"
    }
  ],
  "spoolingMode": "TO_DISK",
  "dataDestinationConfigs": [
    {
      "timestreamConfig": {
        "executionRoleArn": "role-arn",
        "timestreamTableArn": "table-arn"
      }
    }
  ],
   "dataPartitions": [
      { ...  }
  ]
}
```

- Replace *campaign-name* with the name of the campaign that you're creating.
- Replace *signal-catalog-arn* with the Amazon Resource Name (ARN) of the signal catalog.
- Replace *target-arn* with the ARN of a fleet or vehicle that you created.
- Replace *topic-arn* with the ARN of the MQTT topic that you specified as the destination for messages containing vehicle data.

- Replace *role-arn* with the ARN of the task execution role that grants AWS IoT FleetWise permission to send, receive, and take action on messages for the MQTT topic you specified.

```
{
  "name": "campaign-name",
  "targetArn": "target-arn",
  "signalCatalogArn": "signal-catalog-arn",
  "collectionScheme": {
    "conditionBasedCollectionScheme": {
      "conditionLanguageVersion": 1,
      "expression": "$variable.`Vehicle.DemoBrakePedalPressure` > 7000",
      "minimumTriggerIntervalMs": 1000,
      "triggerMode": "ALWAYS"
    }
  },
  "compression": "SNAPPY",
  "diagnosticsMode": "OFF",
  "postTriggerCollectionDuration": 1000,
  "priority": 0,
  "signalsToCollect": [
    {
      "maxSampleCount": 100,
      "minimumSamplingIntervalMs": 0,
      "name": "Vehicle.DemoEngineTorque"
    },
    {
      "maxSampleCount": 100,
      "minimumSamplingIntervalMs": 0,
      "name": "Vehicle.DemoBrakePedalPressure"
    }
  ],
  "spoolingMode": "TO_DISK",
  "dataDestinationConfigs": [
      {
          "mqttTopicConfig": {
              "mqttTopicArn": "topic-arn",
              "executionRoleArn": "role-arn"
          }
      }
  ]
}
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `CreateCampaign` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Logical expressions for AWS IoT FleetWise campaigns

AWS IoT FleetWise uses a logical expression to recognize what data to collect as part of a campaign. For more information about expressions, see Expressions in the *AWS IoT Events Developer Guide*.

The expression variable should be constructed to comply with the rules for the type of data being collected. For telemetry system data, the expression variable should be the signal's fully qualified name. For vision system data, the expression combines the signal's fully qualified name with the path leading from the signal's data type to one of its properties.

For example, if the signal catalog contains the following nodes:

```
{
    myVehicle.ADAS.Camera:
    type: sensor
    datatype: Vehicle.ADAS.CameraStruct
    description: "A camera sensor"
```

```
    myVehicle.ADAS.CameraStruct:
    type: struct
    description: "An obstacle detection camera output struct"
}
```

If the nodes follow the ROS 2 definition:

```
{
    Vehicle.ADAS.CameraStruct.msg:
    boolean obstaclesExists
    uint8[] image
    Obstacle[30] obstacles
}
{
    Vehicle.ADAS.Obstacle.msg:
    float32: probability
    uint8 o_type
    float32: distance
}
```

The following are all possible event expression variables:

```
{
...
    $variable.`myVehicle.ADAS.Camera.obstaclesExists`
    $variable.`myVehicle.ADAS.Camera.Obstacle[0].probability`
    $variable.`myVehicle.ADAS.Camera.Obstacle[1].probability`
...
    $variable.`myVehicle.ADAS.Camera.Obstacle[29].probability`
    $variable.`myVehicle.ADAS.Camera.Obstacle[0].o_type`
    $variable.`myVehicle.ADAS.Camera.Obstacle[1].o_type`
...
    $variable.`myVehicle.ADAS.Camera.Obstacle[29].o_type`
    $variable.`myVehicle.ADAS.Camera.Obstacle[0].distance`
    $variable.`myVehicle.ADAS.Camera.Obstacle[1].distance`
...
    $variable.`myVehicle.ADAS.Camera.Obstacle[29].distance`
}
```

Update an AWS IoT FleetWise campaign

You can use the UpdateCampaign API operation to update an existing campaign. The following command uses AWS CLI.

- Replace *campaign-name* with the name of the campaign that you're updating.
- Replace *action* with one of the following:
  - APPROVE – Approves the campaign to allow AWS IoT FleetWise to deploy it to a vehicle or fleet.
  - SUSPEND – Suspends the campaign. The campaign is deleted from vehicles and all vehicles in the suspended campaign will stop sending data.
  - RESUME – Reactivates the SUSPEND campaign. The campaign is set to be redeployed to all vehicles on next check-in and the vehicles will resume sending data.
  - UPDATE – Updates the campaign by defining attributes and associating them with the campaign.
- Replace *description* with a new description.

  The description can have up to 2,048 characters.
- Replace *data-extra-dimensions* with specified vehicle attributes to enrich data collected during the campaign. For example, you can add vehicle make and model to the campaign, and AWS IoT FleetWise will associate the data with those attributes as dimensions in Amazon Timestream. You can then query the data against vehicle make and model.

```
aws iotfleetwise update-campaign \
            --name campaign-name \
            --action action \
            --description description \
            --data-extra-dimensions data-extra-dimensions
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the UpdateCampaign API operation.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Delete an AWS IoT FleetWise campaign

You can use the AWS IoT FleetWise console or API to delete campaigns.

Delete a campaign (console)

To delete a campaign, use the AWS IoT FleetWise console.

To delete a campaign

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Campaigns**.
3. On the **Campaigns** page, choose the target campaign.
4. Choose **Delete**.
5. In **Delete `campaign-name`?**, enter the name of the campaign to delete, and then choose **Confirm**.

Delete a campaign (AWS CLI)

You can use the DeleteCampaign API operation to delete a campaign. The following example uses AWS CLI.

To delete a campaign, run the following command.

Replace *`campaign-name`* with the name of the vehicle that you're deleting.

```
aws iotfleetwise delete-campaign --name campaign-name
```

Deleted data partitions are not recoverable

Deleting a campaign removes all data from devices and the data in a partition won't upload to the cloud.

Verify campaign deletion

You can use the ListCampaigns API operation to verify if a campaign has been deleted. The following example uses the AWS CLI.

To retrieve a paginated list of summaries for all campaigns, run the following command.

```
aws iotfleetwise list-campaigns
```

Get AWS IoT FleetWise campaign information

You can use the GetCampaign API operation to retrieve vehicle information. The following example uses the AWS CLI.

To retrieve the metadata of a campaign, run the following command.

Replace *campaign-name* with the name of the campaign to you want to retrieve.

```
aws iotfleetwise get-campaign --name campaign-name
```

Note

This operation is eventually consistent. In other words, changes to the campaign might not be reflected immediately.

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `GetCampaign` API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
            ]
        }
    ]
}
```

Store and forward campaign data

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Use data partitions within campaigns to temporarily store signal data on the Edge for vehicles and fleets. By configuring upload and storage options for data partitions, you can optimize your ideal conditions for data forwarding to your designated data destinations (like an Amazon S3 bucket). For example, you can configure the data partition to store data on a vehicle until it connects to Wi-Fi. Then, once the vehicle connects, the campaign triggers the data in that particular partition to be sent to the cloud. Alternatively, you can collect data using AWS IoT Jobs.

Topics

- Create data partitions
- Upload campaign data
- Upload data using AWS IoT Jobs

Create data partitions

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

A data partition in a campaign temporarily stores signal data. You configure when and how to forward the data to the cloud.

A data partition works by first designating a particular set of data using the `dataPartitionId` for a campaign. Then, you can further define partition storage options such as maximum size, minimum time to keep the data partition live (on disk), and where to store the data on the Edge. You can determine the storage location on the vehicle using `storageLocation`. The storage location determines the folder name for the data partition under the campaign storage folder. The campaign storage folder is under a folder named after the vehicle name under a persistency path defined in the Edge config file. This is the absolute path of the storage location: `{persistency_path} / {vehicle_name} / {campaign_name} / {storage_location}`.

The spooling mode set to `TO_DISK` specifies that the partitioned data should be saved to a disk on the vehicle. Data storage for data partitions operates on a FIFO (first in, first out) basis. If you delete a campaign, you also delete the data in the associated data partition. If you don't specify a data partition for connectivity on/off use cases, AWS IoT FleetWise still stores data in a ring buffer on the vehicle when there is no connectivity. When connectivity resumes, AWS IoT FleetWise uploads the data to the cloud. This behavior is configurable in the Edge Agent for AWS IoT FleetWise software.

> **Important**
>
> If your data partition exceeds your set maximum storage limit, newer data overwrites older data when the partition reaches the maximum size. Lost data on the Edge isn't recoverable. Storage size is determined by your Edge storage limit.
> When data is uploaded to the cloud, it can be removed after the minimum time to live passes. Set the minimum time to live appropriately to avoid unintended deletion.

Upload options determine variable expressions and condition language. If upload options are specified, you must also specify storage options. You can also request that signals in data partitions are uploaded into the cloud. For more information, see Upload campaign data.

After data partition conditions are defined, `signalsToCollect` helps specify which signals to account for in the data partition. You can either specify IDs for data partitions, or set the `dataPartitionId` to `default` to use an established default data partition. A signal without a specified `dataPartitionId` will be associated with the default `dataPartition`.

To create a data partition

Using the following example, create a campaign with a data partition storage condition. This example campaign is configured to store vehicle data in Amazon Timestream.

1. Replace *`campaign-name`* with the name of the campaign that you're creating.
2. (Optional) Provide a description.
3. Replace *`role-arn`* with the Amazon Resource Name (ARN) of the task execution role that grants AWS IoT FleetWise permission to deliver data to the Timestream table.
4. Replace *`table-arn`* with the ARN of the Timestream table.
5. Replace *`signal-catalog-arn`* with the ARN of the signal catalog.
6. Replace *`data-partition-id`* both for the `dataPartitions` ID and as the ID to associate with `signalsToCollect`. First, replace the ID of the data partition to use in the signal. For `signalsToCollect`, the ID must match one of the IDs provided in `dataPartitions`.

   > **Note**
   >
   > Establish a default data partition for a campaign by using `default` as the ID.

7. Replace *`target-arn`* with the ARN of a fleet or vehicle that you created.

```
{
    "name": "campaign-name",
    "description": "Measurement of SOC, SOH, thermal, and power optimization for Fleet
 2704",
    "targetArn": "target-arn",
    "collectionScheme": {
        "conditionBasedCollectionScheme": {
            "conditionLanguageVersion": 1,
```

```
            "expression": "$variable.`Vehicle.BMS` > 50",
            "minimumTriggerIntervalMs": 1000,
            "triggerMode": "ALWAYS"
        }
    },
    "compression": "SNAPPY",
    "dataDestinationConfigs": [{
        "timestreamConfig": {
            "executionRoleArn": "role-arn",
            "timestreamTableArn": "table-arn"
        }
    }],
    "dataPartitions": [{
        "id": "data-partition-id",
        "storageOptions": {
            "maximumSize": {
                "unit": "GB",
                "value": 1024
            },
            "minimumTimeToLive": {
                "unit": "WEEKS",
                "value": 6
            },
            "storageLocation": "string"
        },
        "uploadOptions": {
            "conditionLanguageVersion": 1,
            "expression": "$variable.`Vehicle.BMS.PowerOptimization` > 90"
        }
    }],
    "signalCatalogArn": "signal-catalog-arn",
    "signalsToCollect": [{
        "dataPartitionId": "data-partition-id",
        "maxSampleCount": 50000,
        "minimumSamplingIntervalMs": 100,
        "name": "Below-90-percent"
    }],
    "spoolingMode": "TO_DISK",
    "tags": [{
        "Key": "BMS",
        "Value": "Under-90"
    }]
}
```

After meeting all specified conditions, the partitioned data forwards to the cloud, enabling the collection and storage of new partitioned signals.

Next, you'll call the `UpdateCampaign` API to deploy it to the Edge Agent for AWS IoT FleetWise software. For more information, see Upload campaign data.

Upload campaign data

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

There are two ways to upload campaign data on the Edge:

- Campaigns that meet your upload conditions will automatically upload data to the cloud after they are approved. To approve a campaign, use the `updateCampaign` API operation.
- Through AWS IoT Jobs, you can force data to upload even when specified conditions are not met. For more information, see Upload data using AWS IoT Jobs.

To upload campaign data using the `UpdateCampaign` API operation

After you create the campaign, the campaign status displays as `WAITING_FOR_APPROVAL` until you change the `action` to `APPROVED`.

- Use the following sample to update the campaign `action` by calling on the UpdateCampaign API operation.

```
{
   "action": "APPROVED",
   "dataExtraDimensions": [ "string" ],
   "description": "string",
   "name": "string"
}
```

Upload data using AWS IoT Jobs

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

With AWS IoT Jobs, you can configure campaigns to upload stored vehicle data to the cloud whenever you need it.

To create a job document for your campaign

- Use the following example to create a job document for the campaign. A job document is a .json file that contains information about vehicles or fleets required to perform a job. For more information on creating job document, see Create and manage jobs by using the AWS CLI in the *AWS IoT Developer Guide*.

  To request that only one vehicle uploads data, set the job target to the AWS IoT thing that's associated with the vehicle. To request that multiple vehicles (in the same campaign) upload data, create a thing group of all things corresponding with the vehicles, and then set the job target to the thing group.

  ```
  {
    "version": "1.0",
    "parameters": {
       "campaignArn": ${aws:iot:parameter:campaignArn},
       "endTime": ${aws:iot:parameter:endTime}
    }
  }
  ```

  a. Replace `CampaignArn` with the Amazon Resource Name (ARN) of a campaign in the same Region and account. The campaign ARN is required.

  b. (Optional) Replace `endTime` with the timestamp of data collected on the vehicle in ISO 8601 UTC format (without milliseconds). For example, `2024-03-05T23:00:00Z`. The timestamp is exclusive and determines the last datapoint to be uploaded. If you omit `endTime`, the Edge Agent software continues to upload until all of a campaign's

stored data is uploaded. After all data is uploaded, it updates the job execution status to SUCCEEDED. The job's state updates to COMPLETED.

To create a job using a managed job template

1. Choose **IoT-IoTFleetWise-CollectCampaignData** from the list of managed templates. For more information, see Create a job from AWS managed templates in the *AWS IoT Developer Guide*.
2. The managed template has the `CampaignArn` and `endTime` parameters.

   a. Replace `CampaignArn` with the Amazon Resource Name (ARN) of a campaign in the same Region and account. The campaign ARN is required.

   b. (Optional) Replace `endTime` with the timestamp of data collected on the vehicle in ISO 8601 UTC format (without milliseconds). For example, `2024-03-05T23:00:00Z`. The timestamp is exclusive and determines the last datapoint to be uploaded. If you omit `endTime`, the Edge Agent software continues to upload until all of a campaign's stored data is uploaded. After all data is uploaded, it updates the job execution status to SUCCEEDED. The job's state updates to COMPLETED.

For related troubleshooting topics, see Store and forward issues.

For more information on AWS IoT Jobs, see Jobs in the *AWS IoT Developer Guide*.

Collect diagnostic trouble code data using AWS IoT FleetWise

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

When a vehicle detects an error, it generates a diagnostic trouble code (DTC) and records a snapshot of the affected sensors or actuators. DTCs help you learn about errors in near real-time, understand what is causing them, and take corrective actions. AWS IoT FleetWise supports the collection of DTCs, including corresponding DTC snapshots and extended data through a data collection campaign. This topic introduces the concepts, workflows, and keywords that facilitate DTC data collection, illustrated with examples.

The following shows key concepts for using DTC.

Custom defined functions

A custom defined function is the ability to invoke and execute your own functions predefined on the Edge Agent, extending the custom decoding concept. These functions are used in coordination with the AWS IoT FleetWise Agent. The Edge Agent for AWS IoT FleetWise software provides built-in functions for calculating signal statistics like minimum, maximum, and average values. A custom-defined function extends this capability by allowing you to create tailored logic for specific use cases. For diagnostic trouble code (DTC) data collection, developers can leverage custom functions to implement advanced data retrieval mechanisms, such as fetching DTC codes, snapshots, and extended data directly from the vehicle's Edge through Unified Diagnostic Services (UDS) or alternative diagnostic interfaces.

For more information, see the custom function guide and the DTC data collection reference implementation in the *Edge Agent Developer Guide*.

Signal fetching

In data collection campaigns, signals are typically collected continuously from a device and buffered on the Edge Agent software. Signals are then uploaded or stored periodically in time-based campaigns or triggered by specific conditions in condition-based campaigns. However, due to concerns about device traffic congestion, DTC signals can't be collected from devices and buffered continuously. To address this, AWS IoT FleetWise provides signal fetching, which ensures that the target signal is fetched discontinuously from a device.

Signal fetching supports both periodic and condition-driven actions. You can define the fetching driven method, conditions, and exact actions using custom defined functions for each signal that should not be collected from a device continuously. For signals managed by the signal fetching mechanism, the trigger type and conditions for local storage or cloud upload are still governed by the `CollectionScheme`, both `timeBasedCollectionScheme` and `conditionBasedCollectionScheme` are supported, which is the same as regular signals.

The following topics show you how you can create and use DTCs.

Topics

- Diagnostic trouble code keywords
- Create a data collection campaign for diagnostic trouble codes
- Diagnostic trouble code use cases

Diagnostic trouble code keywords

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

`signalsToFetch` parameter for create campaign

Use the *signalsToFetch* syntax to configure how the signal information can be fetched on the Edge. Standard signal fetching is controlled by modeling as rules explicitly defined in a decoder manifest or custom defined through Edge First Modeling. With signals to fetch, you can define when and how data is fetched during campaigns.

Signals to fetch allows the collection of DTC information. For example, you can create a signal of string type named `DTC_Info` that can contain DTC information for every engine control unit (ECU). Or, you can filter for a specific ECU.

- `SignalFetchInformation` structure and param definitions.

```
structure SignalFetchInformation {
    @required
    fullyQualifiedName: NodePath,
    @required
    signalFetchConfig: SignalFetchConfig,
    // Conditional language version for this config
    conditionLanguageVersion: languageVersion,
    @required
    actions: EventExpressionList,
}
```

  - `fullyQualifiedName`: the fully qualified name (FQDN) of the signal that you want to use custom fetch for.
  - `signalFetchConfig`: defines rules on how the above defined signals should be fetched. It supports time-based and condition-based fetch.
  - `conditionLanguageVersion`: the conditional language version used for parsing the expression in the config.

- `actions`: a list of all action expressions evaluated on the Edge. The Edge will get the value of the defined signal.

> **Important**
>
> Actions can only use `custom_function`.

Campaign expression keywords

The following expression takes a signal's fully qualified name supported by the vehicle and returns true if the signal doesn't have any data in the signal buffers on the Edge. Otherside, it returns false.

```
isNull(signalFqdn:String): Boolean
```

Example usage

```
isNull($variable.`Vehicle.ECU1.DTC_INFO`) == false

We want to make sure DTC_Info signal is being generated
on edge.
```

This expression takes the following input:

functionName:String

The name of the custom function that is supported by the Edge

params: varargs*Expression*

Parameters for `functionName`. This can be any list of expressions.

Parameters support literal type: String, Int , Boolean, or Double.

```
custom_function(functionName:String, params: varargsExpression): Void
```

Example usage

```
{
       "fullyQualifiedName":"Vehicle.ECU1.DTC_INFO",
```

```
      "signalFetchConfig":{
         "timeBased":{
            "executionFrequencyMs":2000
         }
      },
      "actions":"custom_function(“DTC_QUERY”, -1, 2, -1)"
   }
```

Create a data collection campaign for diagnostic trouble codes

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

This topic describes how to create a data collection campaign for diagnostic trouble codes (DTC).

1. Define a custom signal on the Edge. You need to define the decoding rules for the DTC signal on the Edge as a custom decoded signal. For more information, see Tutorial: Configure network agnostic data collection using a custom decoding interface.
2. Define custom function on the Edge. You need to define a custom function for collecting DTC signals on the Edge at a compiled time.

   For more information, see the custom function guide and the DTC data collection reference implementation in the *Edge Agent Developer Guide*.

   > **Note**
   >
   > An example custom defined function is `DTC_QUERY` as shown in the demo script.

3. Create a signal catalog that models a DTC signal as a string type.

   ```
   [
    {
       "branch": {
           "fullyQualifiedName": "Vehicle",
           "description": "Vehicle"
           }
         },
   ```

```
      {
    "branch": {
        "fullyQualifiedName": "Vehicle.ECU1",
        "description": "Vehicle.ECU1"
        }
      },
      {
    "sensor": {
        "fullyQualifiedName": "Vehicle.ECU1.DTC_INFO",
        "description": "Vehicle.ECU1.DTC_INFO",
        "dataType": "STRING"
      }
   }
 ]
```

4. Create and activate a vehicle model with the DTC signal added.
5. Create and activate a decoder manifest with the DTC signal added. The DTC signal should be a `CUSTOM_DECODING_SIGNAL` signal decoder type with a `CUSTOM_DECODING_INTERFACE` network interface type.

Example signal decoder

```
[
  {
    "fullyQualifiedName": "Vehicle.ECU1.DTC_INFO",
    "interfaceId": "UDS_DTC",
    "type": "CUSTOM_DECODING_SIGNAL",
    "customDecodingSignal": {
      "id": "Vehicle.ECU1.DTC_INFO"
    }
  }
 ]
```

Example network interface

```
[
  {
    "interfaceId": "UDS_DTC",
    "type": "CUSTOM_DECODING_INTERFACE",
    "customDecodingInterface": {
      "name": "NamedSignalInterface"
    }
```

```
  }
]
```

> **Note**
>
> Controller Area Network (CAN) signals don't support the string data type.

6. Provision and create vehicles. The vehicles must utilize a vehicle model (model manifest) and decoder manifest that were activated in the previous steps.
7. Create and approve the campaign. You need to create a campaign by defining DTC signals (optionally with telemetry signals) and deploy it to vehicles.
8. Access the data in the defined destination. DTC data includes the `DTCCode`, `DTCSnapshot`, and `DTCExtendedDatastrings` as a raw string in the data destination defined in the campaign.

Diagnostic trouble code use cases

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

The following use cases assume the `DTC_QUERY` function was defined in the demo script.

Periodic fetch

Fetch a DTC collection at configured intervals.

The following example is a campaign with periodic signal fetching of `Vehicle.DTC_INFO` for all DTCs with a status mask for all ECUs. There is a condition for data collected for `Vehicle.DTC_INFO`.

```
{
  "compression": "SNAPPY",
  "spoolingMode": "TO_DISK",
  "signalsToFetch": [
    {
      "fullyQualifiedName": "Vehicle.ECU1.DTC_INFO",
```

```
      "signalFetchConfig": {
        "timeBased": {
        // The FleetWise Edge Agent will query the UDS module for all DTCs every five
 seconds.
          "executionFrequencyMs": 5000
        }
      },
      "actions": [
      // Every five seconds, this action is called and its output is stored in the
      // signal history buffer of Vehicle.DTC_INFO
        "custom_function(\"DTC_QUERY\", -1, 2, -1)"
      ]
    }
  ],
  "signalsToCollect": [
    {
      "name": "Vehicle.ECU1.DTC_INFO"
    }
  ],
  "collectionScheme": {
    "conditionBasedCollectionScheme": {
      "conditionLanguageVersion": 1,
      // Whenever a new DTC is filled into the signal, the data is ingested.
      "expression": "!isNull($variable.`Vehicle.ECU1.DTC_INFO`)",
      "minimumTriggerIntervalMs": 1000,
      // Make sure that data is ingested only when there are new DTCs.
      "triggerMode": "RISING_EDGE"
    }
  },
  "dataDestinationConfigs": [
    {
      "s3Config":
        {
          "bucketArn": "bucket-arn",
          "dataFormat": "PARQUET",
          "prefix": "campaign-name",
          "storageCompressionFormat": "GZIP"
        }
    }
  ]
}
```

Condition-driven fetch

Fetch a DTC collection when a condition is met. For example, when the CAN signal is `Vehicle.Ignition == 1`, fetch and upload the DTC data.

The following example campaign has condition-driven signal fetching of `Vehicle.ECU1.DTC_INFO` to check whether the DTC ("AAA123") is pending with recordNumber 1 for ECU-1. This campaign has time-based data collection and upload.

```
{
  "compression": "SNAPPY",
  "spoolingMode": "TO_DISK",
  "signalsToFetch": [
    {
      "fullyQualifiedName": "Vehicle.ECU1.DTC_INFO",
      "signalFetchConfig": {
        "conditionBased": {
        // The action will only run when the ignition is on.
          "conditionExpression": "$variable.`Vehicle.Ignition` == 1",
          "triggerMode": "ALWAYS"
        }
      },
      // The UDS module is only requested for the specific ECU address and the specific
 DTC Number/Status.
      "actions": ["custom_function(\"DTC_QUERY\", 1, 2, 8, \"0xAAA123\")"]
    }
  ],
  "signalsToCollect": [
    {
      "name": "Vehicle.ECU1.DTC_INFO"
    },
    {
      "name": "Vehicle.Ignition"
    }
  ],
  "collectionScheme": {
    "timeBasedCollectionScheme": {
      "periodMs": 10000
    }
  },
  "dataDestinationConfigs": [
    {
      "s3Config":
```

```
        {
          "bucketArn": "bucket-arn",
          "dataFormat": "PARQUET",
          "prefix": "campaign-name",
          "storageCompressionFormat": "GZIP"
        }
    }
  ]
}
```

On-demand fetch

Fetch a specific DTC for a fleet.

For an on-demand use case, you can use the same campaign as defined in the periodic fetch. The on-demand effect is achieved by suspending the campaign shortly after the campaign is deployed using the AWS IoT FleetWise console or by running the following CLI command.

- Replace *command-name* with the command name.

```
aws iotfleetwise update-campaign \
    --name campaign-name \
    --action APPROVE
```

Then, suspend the campaign after the DTC data arrives.

```
aws iotfleetwise update-campaign \
    --name campaign-name \
    --action SUSPEND
```

You can resume the campaign again for DTC data fetching.

```
aws iotfleetwise update-campaign \
    --name campaign-name \
    --action RESUME
```

Visualize AWS IoT FleetWise vehicle data

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

The Edge Agent for AWS IoT FleetWise software sends selected vehicle data to an MQTT topic, or transfers it to Amazon Timestream or Amazon Simple Storage Service (Amazon S3). After your data arrives in the data destination, you can use other AWS services to process, re-route, visualize, and share it.

Note

Amazon Timestream is not available in the Asia Pacific (Mumbai) Region.

Processing vehicle data sent to an MQTT topic

Vehicle data sent by MQTT messaging is delivered in near real-time and allows you to use Rules to take action, or route data to other destinations. For more information about using MQTT, see Device communication protocols and Rules for AWS IoT in the *AWS IoT Core Developer Guide*.

The default schema of data that is sent in an MQTT message contains the following fields.

Field name	Data type	Description
`eventId`	varchar	The ID of the data collection event.
`vehicleName`	varchar	The ID of the vehicle from which the data was collected.
`name`	varchar	The name of the campaign that the

Field name	Data type	Description
		Edge Agent software uses to collect data.
`time`	timestamp	The timestamp of the data point.
`measure_name`	varchar	The name of the signal.
`measure_value::bigint`	bigint	Signal values of type Integer.
`measure_value::double`	double	Signal values of type Double.
`measure_value::boolean`	boolean	Signal values of type Boolean.
`measure_value::varchar`	varchar	Signal values of type varchar.

Process vehicle data in Timestream

Timestream is a fully managed time series database that can store and analyze trillions of time series data points per day. Your data is stored in a customer managed Timestream table. You can use Timestream to query vehicle data so that you can gain insights about your vehicles. For more information, see What is Amazon Timestream?

The default schema of data that is transferred to Timestream contains the following fields.

Field name	Data type	Description
`eventId`	varchar	The ID of the data collection event.

Field name	Data type	Description
`vehicleName`	varchar	The ID of the vehicle from which the data was collected.
`name`	varchar	The name of the campaign that the Edge Agent software uses to collect data.
`time`	timestamp	The timestamp of the data point.
`measure_name`	varchar	The name of the signal.
`measure_value::bigint`	bigint	Signal values of type Integer.
`measure_value::double`	double	Signal values of type Double.
`measure_value::boolean`	boolean	Signal values of type Boolean.
`measure_value::varchar`	varchar	Signal values of type varchar.

Visualize vehicle data stored in Timestream

After your vehicle data is transferred to Timestream, you can use the following AWS services to visualize, monitor, analyze, and share your data.

- Visualize and monitor data in dashboards by using Grafana or Amazon Managed Grafana. You can visualize data from multiple AWS sources (such as Amazon CloudWatch and Timestream) and other data sources with a single Grafana dashboard.
- Analyze and visualize data in dashboards by using Quick.

Process vehicle data in Amazon S3

Amazon S3 is an object storage service that stores and protects any amount of data. You can use S3 for a variety of use cases, such as data lakes, backup and restore, archive, enterprise applications, AWS IoT devices, and big data analytics. Your data is stored in S3 as objects in buckets. For more information, see What is Amazon S3?

The default schema of data that is transferred to Amazon S3 contains the following fields.

Field name	Data type	Description
`eventId`	varchar	The ID of the data collection event.
`vehicleName`	varchar	The ID of the vehicle from which the data was collected.
`name`	varchar	The name of the campaign that the Edge Agent software uses to collect data.
`time`	timestamp	The timestamp of the data point.
`measure_name`	varchar	The name of the signal.
`measure_value_BIGINT`	bigint	Signal values of type Integer.
`measure_value_DOUBLE`	double	Signal values of type Double.
`measure_value_BOOLEAN`	boolean	Signal values of type Boolean.

Field name	Data type	Description
`measure_value_STRUCT`	struct	Signal values of type Struct.
`measure_value_VARCHAR`	varchar	Signal values of type varchar.

Amazon S3 object format

AWS IoT FleetWise transfers vehicle data to S3 where it's saved as an object. You can use the object URI that uniquely identifies the data to find data from the campaign. The S3 object URI format depends on if the collected data is unstructured or processed data.

Unstructured data

Unstructured data is stored in S3 in a not pre-defined manner. It can be in various formats, such as images or videos.

Vehicle messages passed to AWS IoT FleetWise with signal data from Amazon Ion files are decoded and transferred to S3 as objects. The S3 objects represent each signal and are binary encoded.

The unstructured data S3 object URI uses the following format:

```
s3://bucket-name/prefix/unstructured-data/random-ID-yyyy-MM-dd-HH-mm-ss-SSS-vehicleName-signalName-fieldName
```

Processed data

Processed data is stored in S3 and undergoes processing steps that validate, enrich, and transform messages. Object lists and velocity are examples of processed data.

Data transferred to S3 are stored as objects that represent records that were buffered for a period of about 10 minutes. By default, AWS IoT FleetWise adds a UTC time prefix in the format `year=YYYY/month=MM/date=DD/hour=HH` before writing objects to S3. This prefix creates a logical hierarchy in the bucket where each forward slash (/) creates a level in the hierarchy. The processed data also contains the S3 object URI to unstructured data.

The processed data S3 object URI uses the following format:

```
s3://bucket-name/prefix/processed-data/year=YYYY/month=MM/day=DD/hour=HH/
part-0000-random-ID.gz.parquet
```

Raw data

Raw data, also known as primary data, are data collected from Amazon Ion files. You can use raw data to troubleshoot any issues or to root cause errors.

The raw data S3 object URI uses the following format:

```
s3://bucket-name/prefix/raw-data/vehicle-name/eventID-timestamp.10n
```

Analyze vehicle data stored in Amazon S3

After your vehicle data is transferred to S3, you can use the following AWS services to monitor, analyze, and share your data.

Extract and analyze data using Amazon SageMaker AI for downstream labeling and machine learning (ML) workflows.

For more information, see the following topics in the *Amazon SageMaker AI Developer Guide*:

- Process data
- Train machine learning models
- Label Images

Catalog your data using AWS Glue crawler and analyze it in Amazon Athena. By default, objects written to S3 have Apache Hive style time partitions, with data paths that contain key-value pairs connected by equal signs.

For more information, see the following topics in the *Amazon Athena User Guide*:

- Partitioning data in Athena
- Using AWS Glue to connect to data sources in Amazon S3
- Best practices when using Athena with AWS Glue

Visualize data using Quick by either reading your Athena table or S3 bucket directly.

Tip

If you're reading from S3 directly, confirm that your vehicle data is in JSON format because Quick doesn't support Apache Parquet format.

For more information, see the following topics in the *Amazon Quick User Guide*:

- Supported data sources
- Creating a data source

Commands

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.
This documentation describes how to use the commands feature for AWS IoT FleetWise. For information about using the commands feature in AWS IoT Device Management, see commands.
You are solely responsible for deploying commands in a manner that is safe and compliant with applicable laws. For more information on your responsibilities, please see the AWS Service Terms for AWS IoT Services.

Use the commands feature to execute commands on a vehicle from the cloud. Commands target one device at a time, and can be used for low-latency, high-throughput applications, such as to retrieve the device-side logs, or to initiate a device state change.

The *command* is a resource that's managed by AWS IoT Device Management. It contains reusable configurations that are applied when sending a command execution to the vehicle. You can pre-define a set of commands for specific use cases, or use them to create reusable configurations for recurrent use cases. For example, you can configure commands that can be used by an App to lock a vehicle's door or to change the temperature remotely.

Using the AWS IoT commands feature, you can:

- Create a command resource and reuse the configuration to send multiple commands to your target device and then execute them on the device.
- Control the granularity with which you want each command to be executed on the device. For example, you can provision a vehicle as an AWS IoT thing, and then send a command to lock or unlock the doors of the vehicle.
- Run multiple commands concurrently on the target device without waiting for the previous one to be completed.
- Choose to enable notifications for commands events, and retrieve the status and result information from the device as it runs the command and once it's completed.

The following topics show you how to create, send, receive, and manage commands.

Topics

Commands concepts

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Commands are instructions that are sent from the cloud to your target device. The target device can be a vehicle and it must registered as an *AWS IoT thing* in the thing registry. The command can contain parameters that define an action that the actuators of the vehicle need to perform. The vehicle then parses the command and its parameters, and processes them to take the corresponding action. It then responds to the cloud application with the status of the command execution.

For the detailed workflow, see Vehicles and commands.

Topics

Commands key concepts

The following shows some key concepts for using the commands feature and how it works with last known state (LKS) state templates.

Command

A *Command* is an entity that you can use to send instructions to a physical vehicle to have it perform actions such as turning on the engine or changing the position of the windows. You can pre-define a set of commands for specific use cases, or use them to create reusable configurations for recurrent use cases. For example, you can configure commands that can be used by an App to lock a vehicle's door or to change the temperature remotely.

Namespace

When you use the commands feature, you must specify the namespace for the command. When you create a command in AWS IoT FleetWise, you must choose `AWS-IoT-FleetWise` as your namespace. When you use this namespace, you must provide the parameters that will be used to run the command on the vehicle. If you want to create a command in AWS IoT Device Management instead, you must use the `AWS-IoT` namespace instead. For more information, see commands in the *AWS IoT Device Management developer guide*.

Command states

The commands that you create will be in an available state, which means that it can be used to start a command execution on the vehicle. If a command becomes outdated, you can deprecate the command. For a command in the deprecated state, existing command executions will run to completion. You cannot update the command or run any new executions. To send new executions, you must restore the command so that it becomes available.

You can also delete a command if it's no longer required. When you mark a command for deletion, if the command has been deprecated for a duration that's longer than the maximum timeout of 24 hours, the command will be deleted immediately. If the command isn't deprecated, or has been deprecated for a duration shorter than the maximum timeout, the command will be in a pending deletion state. The command will be removed automatically from your account after 24 hours.

Parameters

When creating a command, you can optionally specify the parameters that you want the target vehicle to execute when running the command. The command you create is a reusable configuration and it can be used to send multiple command executions to your vehicle and execute them concurrently. Alternatively, you can also specify the parameters only at runtime and choose to perform a one-time operation of creating a command and sending it to your vehicle.

Target vehicle

When you want to run the command, you must specify a target vehicle that will receive the command and perform specific actions. The target vehicle must have already been registered as a *thing* with AWS IoT. After you send the command to the vehicle, it will start executing an instance of the command based on the parameters and the values that you specified.

Actuators

When you want to run the command, you must specify the actuators on the vehicle that will receive the command and their values that determine the actions to be performed. You can optionally configure default values for the actuators to avoid sending inaccurate commands. For example, you can use a default value of `LockDoor` to a door lock actuator so that the command doesn't accidentally unlock the doors. For general information about actuators, see Key concepts.

Data type support

The following data types are supported for the actuators that are used for the commands feature.

> **Note**
>
> Arrays are not supported for telematics data, commands, or last known state (LKS). You can only use the array data type for vision systems data.

- Floating point types. The following types are supported.
  - Float (32 bits)
  - Double (64 bits)
- Integer (both signed and unsigned). The following integer types are supported.
  - int8 and uint8
  - int16 and uint16
  - int32 and uint32
- Long. The following long types are supported.
  - Long (int64)
  - Unsigned long (uint64)
- String

- Boolean

Command execution

A command execution is an instance of a command running on a target device. The vehicle executes the command using either the parameters that you specified when you created the command or when you started the command execution. The vehicle then performs the operations specified and returns the status of the execution.

> **Note**
>
> For a given vehicle, you can run multiple commands concurrently. For information about the maximum number of concurrent executions that you can run for each vehicle, see AWS IoT Device Management commands quotas.

Last known state (LKS) state templates

State templates provide a mechanism for vehicle owners to track the state of their vehicle. To monitor the last known state (LKS) of your vehicles in near-real time, you can create state templates and associate them with your vehicles.

Using the commands feature, you can perform "On Demand" operations that can be used for state data collection and processing. For example, you can request the current vehicle state one-time (fetch), or activate or deactivate previously deployed LKS state templates to start or stop reporting vehicle data. For examples that show how to use commands with state templates, see Command usage scenarios.

Command execution status

After you start the command execution, your vehicle can publish the status of the execution, and provide the reasons for the status as additional information about the execution. The following sections describe the various command execution statuses, and the status codes.

Topics

Command execution status reason code and description

To report updates to the command execution status, your vehicles can use the `UpdateCommandExecution` API to publish the updated status information to the cloud, using the Commands reserved topics described in the *AWS IoT Core developer guide*. When reporting the status information, your devices can provide additional context about the status of each command execution using the `StatusReason` object, and the fields `reasonCode` and `reasonDescription` that are contained within the object.

Command execution status and status codes

The following table shows the various command execution status codes and the allowed statuses that a command execution can transition to. It also shows whether a command execution is "terminal" (that is, no further status updates are forthcoming), whether the change is initiated by the vehicle or the cloud, and the different pre-defined status codes and how they map to the statuses that are reported by the cloud.

- For information about how AWS IoT FleetWise uses the predefined status codes, and the `statusReason` object, see Command status in the *Edge Agent for AWS IoT FleetWise software documentation*.
- For additional information about terminal and non-terminal executions, and the transitions between the statuses, see Command execution status in the *AWS IoT Core developer guide*.

Command execution status and source

Command execution status	Description	Initiated by device/cloud?	Terminal execution?	Allowed status transitions	Pre-defined status codes
`CREATED`	When the API request to start executing the command (`StartCommandExecution` API) is successfu	Cloud	No	• `IN_PROGRESS` • `SUCCEEDED` • `FAILED` • `REJECTED` • `TIMED_OUT`	None

Command execution status	Description	Initiated by device/cloud?	Terminal execution?	Allowed status transitions	Pre-defined status codes
	l, the command execution status changes to CREATED.				
IN_PROGRESS	When the vehicle starts executing the command, it can publish a message to the response topic to update the status to IN_PROGRESS .	Device	No	• IN_PROGRESS • SUCCEEDED • FAILED • REJECTED • TIMED_OUT	COMMAND_STATUS_COMMAND_IN_PROGRESS

Command execution status	Description	Initiated by device/cloud?	Terminal execution?	Allowed status transitions	Pre-defined status codes
SUCCEEDED	When the vehicle has successfully processed the command and completed the execution, it can publish a message to the response topic to update the status to SUCCEEDED.	Device	Yes	Not applicable	COMMAND_STATUS_SUCCEEDED
FAILED	When the vehicle failed to execute the command, it can publish a message to the response topic to update the status to FAILED.	Device	Yes	Not applicable	COMMAND_STATUS_EXECUTION_FAILED

Command execution status	Description	Initiated by device/cloud?	Terminal execution?	Allowed status transitions	Pre-defined status codes
`REJECTED`	If the vehicle fails to accept the command, it can publish a message to the response topic to update the status to `REJECTED`.	Device	Yes	Not applicable	None

Command execution status	Description	Initiated by device/cloud?	Terminal execution?	Allowed status transitions	Pre-defined status codes
TIMED_OUT	The command execution status can change to TIMED_OUT due to any of the following reasons. • The result of the command execution wasn't received and the cloud automatically reports a TIMED_OUT status. • The vehicle reports that a time out occurred when it was attempting to execute the	Device and cloud	No	• SUCCEEDED • FAILED • REJECTED • TIMED_OUT	COMMAND_STATUS_EXECUTION_TIMEOUT

Command execution status	Description	Initiated by device/cloud?	Terminal execution?	Allowed status transitions	Pre-defined status codes
	command. In this case, the command execution becomes terminal. For more information about this status, see Command execution timeout status.				

Command execution timeout status

A command execution timeout can be reported both by the cloud and the device. After the command is sent to the device, a timer starts. If there was no response received from the device within the specified duration, the cloud reports a `TIMED_OUT` status. In this case, the command execution in `TIMED_OUT` status is non-terminal.

The device can override this status to a terminal status, such as `SUCCEEDED`, `FAILED`, or `REJECTED`. It can also report that a timeout occurred when running the command. In this case, the command execution status stays at `TIMED_OUT` but the fields of the `StatusReason` object are updated based on the information reported by the device. The command execution in the `TIMED_OUT` status now becomes terminal.

For additional information, see Command execution timeout considerations in the *AWS IoT Core developer guide*.

Vehicles and commands

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.
> You are solely responsible for deploying commands in a manner that is safe and compliant with applicable laws.

To use the commands feature:

1. First, create a command resource. Optionally, specify the parameters that contain the information required to execute the command.
2. Specify the target vehicle that will receive the command and perform the specified actions.
3. Now, you can run the command on the target device, and check the command execution details to retrieve the status and use CloudWatch logs to further troubleshoot any issues.

The following sections show you the workflow between vehicles and commands.

Topics

Workflow overview

The following steps provide an overview of the commands workflow between your vehicles and commands. When you use any of the commands HTTP API operations, the request is signed using Sigv4 credentials.

> **Note**
>
> Except for the `StartCommandExecution` API operation, all operations that are performed over HTTP protocol use the control plane endpoint.

1. **Establish MQTT connection and subscribe to commands topics**

   To prepare for the commands workflow, the devices must establish an MQTT connection with the `iot:Data-ATS` endpoint, and subscribe to the commands request topic mentioned above. Optionally, your devices can also subscribe to the commands accepted and rejected response topics.

2. **Create a vehicle model and command resource**

   You can now create a vehicle and a command resource using the `CreateVehicle` and `CreateCommand` control plane API operations. The command resource contains the configurations to be applied when the command is executed on the vehicle.

3. **Start command execution on the target device**

   Start the command execution on the vehicle using the `StartCommandExecution` data plane API with your account-specific `iot:Jobs` endpoint. The API publishes a protobuf-encoded payload message to the commands request topic.

4. **Update the result of the command execution**

   The vehicle processes the command and the payload received, and then publishes the result of the command execution to the response topic using the `UpdateCommandExecution` API. If your vehicle subscribed to the commands accepted and rejected response topics, it will receive a message that indicates whether the response was accepted or rejected by the cloud service.

5. **(Optional) Retrieve command execution result**

   To retrieve the result of the command execution, you can use the `GetCommandExecution` control plane API operation. After your vehicle publishes the command execution result to the response topic, this API will return the updated information.

6. **(Optional) Subscribe and manage commands events**

   To receive notifications for command execution status updates, you can subscribe to the commands events topic. You can then use the `CreateTopicRule` control plane API to route

the commands events data to other applications such as AWS Lambda functions or Amazon SQS and build applications on top of it.

Vehicle workflow

The following steps describe the vehicle workflow in detail when using the commands feature.

> **Note**
>
> The operations that are described in this section use the MQTT protocol.

1. **Establish an MQTT connection**

   To prepare your vehicles to use the commands feature, it must first connect to the AWS IoT Core message broker. Your vehicle must be allowed to perform the `iot:Connect` action to connect to AWS IoT Core and establish an MQTT connection with the message broker. To find the data plane endpoint for your AWS account, use the `DescribeEndpoint` API or the `describe-endpoint` CLI command as shown below.

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Running this command returns the account-specific data plane endpoint as shown below.

   ```
   account-specific-prefix.iot.region.amazonaws.com
   ```

2. **Susbcribe to commands request topic**

   After a connection has been established, your devices can then subscribe to the AWS IoT commands MQTT request topic. When you create a command and start the command execution on your target device, a protobuf encoded payload message will be published to the request topic by the message broker. Your device can then receive the payload message and process the command. In this example, replace *<DeviceID>* with the unique identifier of your target vehicle. This ID can be the unique identifier of your vehicle or a thing name

   > **Note**
   >
   > The payload message that's sent to the device must use the protobuf format.

```
$aws/commands/things/<DeviceID>/executions/+/request/protobuf
```

3. **(Optional) Subscribe to commands response topics**

   Optionally, you can subscribe to these commands response topics to receive a message that indicates whether the cloud service accepted or rejected the response from the device.

   > **Note**
   >
   > It is optional for your vehicles to subscribe to the `/accepted` and `/rejected` response topics. Your vehicles will automatically receive these response messages even if they haven't explicitly subscribed to these topics.

   ```
   $aws/commands/things/<DeviceID>/executions/<ExecutionId>/response/protobuf/accepted
   $aws/commands/things/<DeviceID>/executions/<ExecutionId>/response/protobuf/rejected
   ```

4. **Update the result of a command execution**

   The target vehicle then processes the command. It then uses the `UpdateCommandExecution` API to publish the result of the execution to the following MQTT response topic.

   > **Note**
   >
   > For a given vehicle and command execution, the *<DeviceID>* must match the corresponding field in the request topic that the device subscribed to.

   ```
   $aws/commands/things/<DeviceID>/executions/<ExecutionId>/response/protobuf
   ```

   The `UpdateCommandExecution` API is a data plane API operation over MQTT that's authenticated with TLS.

   - If the cloud service successfully processed the command execution result, a message is published to the MQTT accepted topic. The accepted topic uses the following format.

```
$aws/commands/things/<DeviceID>/executions/<ExecutionId>/response/protobuf/
accepted
```

- If the cloud service failed to process the command execution result, a response is published to the MQTT rejected topic. The rejected topic uses the following format.

```
$aws/commands/things/<DeviceID>/executions/<ExecutionId>/response/protobuf/
rejected
```

For more information about this API and an example, see Update command execution result.

Commands workflow

The following steps describe the commands workflow in detail.

> **Note**
>
> The operations that are described in this section use the HTTP protocol.

1. **Register your vehicle**

   Now that you've prepared your vehicle to use the commands feature, you can prepare your application by registering your vehicle and then creating a command that will be sent to the vehicle. To register the vehicle, create an instance of a vehicle model (model manifest) using the `CreateVehicle` control plane API operation. For more information and examples, see Create a vehicle.

2. **Create a command**

   Use the `CreateCommand` HTTP control plane API operation to model commands that are applicable to the vehicle that you're targeting. Specify any parameters and default values to be used when executing the command, and make sure that it uses the `AWS-IoT-FleetWise` namespace. For more information and examples for using this API, see Create a command resource.

3. **Start the command execution**

   You can now execute the command that you created on the vehicle using the `StartCommandExecution` data plane API operation. AWS IoT Device Management fetches the command and command parameters, and validates the incoming request. It then invokes AWS IoT FleetWise API with the required parameters to generate the vehicle-specific payload. The payload is then sent to the device by AWS IoT Device Management over MQTT to the command request topic that your device subscribed to. For more information and examples for using this API, see Send a command (AWS CLI).

   ```
   $aws/commands/things/<DeviceID>/executions/+/request/protobuf
   ```

   > **Note**
   >
   > If the device was offline when the command was sent from the cloud and MQTT persistent sessions is in use, the command waits at the message broker. If the device comes back online before the time out duration, and if it has subscribed to the commands request topic, the device can then process the command and publish the result to the response topic. If the device doesn't come back online before the time out duration, the command execution will time out and the payload message will expire.

4. **Retrieve the command execution**

   After you've executed the command on the device, use the `GetCommandExecution` control plane API operation to retrieve and monitor the result of the command execution. You can also use the API to obtain additional information about the execution data, such as when it was last updated, when the execution was completed, and the parameters specified.

   > **Note**
   >
   > To retrieve the latest status information, your device must have published the command execution result to the response topic.

   For more information and examples for using this API, see Get command execution.

(Optional) Commands notifications

You can subscribe to commands events to receive notifications when the status of a command execution changes. The following steps show you how to subscribe to commands events, and then process them.

1. **Create a topic rule**

   You can subscribe to the commands events topic and receive notifications when the status of a command execution changes. You can also create a topic rule to route the data processed by the vehicle to other applications such as AWS Lambda functions. You can create a topic rule either using the AWS IoT console, or the `CreateTopicRule` AWS IoT Core control plane API operation. For more information, see Creating and AWS IoT rule.

   In this example, replace *<CommandID>* with the identifier of the command for which you want to receive notifications and *<CommandExecutionStatus>* with the status of the command execution.

   ```
   $aws/events/commandExecution/<CommandID>/<CommandExecutionStatus>
   ```

   > **Note**
   >
   > To receive notifications for all commands and command execution statuses, you can use wildcard characters and subscribe to the following topic.

   ```
   $aws/events/commandExecution/+/#
   ```

2. **Receive and process commands events**

   If you created a topic rule in the previous step to subscribe to commands events, then you can manage the commands push notifications that you receive. You can also optionally build applications on top of it, such as with AWS Lambda, Amazon SQS, Amazon SNS, or AWS Step Functions using the topic rule that you created.

The following code shows a sample payload for the commands events notifications that you'll receive.

```
{
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
    "status":"FAILED",
    "statusReason": {
         "reasonCode": "4",
         "reasonDescription": ""
    },
    "eventType": "COMMAND_EXECUTION",
    "commandArn":"arn:aws:iot:us-east-1:123456789012:command/0b9d9ddf-
e873-43a9-8e2c-9fe004a90086",
    "targetArn":"arn:aws:iot:us-east-1:123456789012:thing/5006c3fc-
de96-4def-8427-7eee36c6f2bd",
    "timestamp":1717708862107
}
```


Create and manage commands

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can configure reusable remote actions or send one-time, immediate instructions to your devices. When you use this feature, you can specify the instructions that your devices can execute in near real time. A command enables you to configure resuable remote actions for your target vehicle. After you create a command, you can start a command execution that targets a specific vehicle.

This topic shows how you can create and manage a command resource using the AWS IoT Core API or the AWS CLI. It shows you how to perform the following actions on a command resource.

Topics

- Delete a command resource

Create a command resource

You can use the `CreateCommand` AWS IoT Core control plane API operation or the AWS IoT FleetWise console to create a command.

Create a command (console)

You can use the AWS IoT FleetWise console to create a command.

To create a command

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Commands**.
3. Choose **Create command**.
4. Specify a unique command ID to help you identify the command that you want to run on the vehicle.
5. (Optional) Specify an optional display name and description.
6. (Optional) Select the actuator and default parameter value. Parameters specify the actions the target vehicle can perform upon receiving the command. If you don't add parameters, you will need to provide them when running the command.
7. Choose an IAM role that grants permissions to generate the payload for commands. See Controlling access.
8. Choose **Create command**.

Create a command (AWS CLI)

The following example shows how to create a command with a parameter.

Considerations when creating a command

When you create a command in AWS IoT FleetWise:

- You must specify the `roleArn` that grants permission to create and run commands on your vehicle. For more information and about sample policies including when KMS keys are enabled, see Grant AWS IoT Device Management permission to generate the payload for commands with AWS IoT FleetWise.

- You must specify `AWS-IoT-FleetWise` as the namespace.
- You can skip the `mandatory-parameters` field and specify them at run time instead. Alternatively, you can create a command with parameters, and optionally specify default values for them. If you specified default values, then at run time, you can use these values or override them by specifying your own values. For these additional examples, see Command usage scenarios.
- You can specify up to three name-value pairs for the `mandatory-parameters` field. However, when executing the command on the vehicle, only one name-value pair is accepted, and the `name` field must use the fully qualified name with the `$actuatorPath.` prefix.

- Replace *`command-id`* with a unique identifier for the command. You can use UUID, alphanumeric characters, "-", and "_".
- Replace *`role-arn`* with the IAM role that grants you permission to create and run commands, for example, "`arn:aws:iam:`*`accountId`*`:role/`*`FwCommandExecutionRole`*".
- (Optional) Replace *`display-name`* with a user-friendly name for the command, and *`description`* with a meaningful description of the command.
- Replace *`name`* and *`value`* of the `mandatory-parameters` object with the required information for the command being created. The `name` field is the fully qualified name as defined in the signal catalog with `$actuatorPath.` as the prefix. For example, `name` can be *`$actuatorPath.Vehicle.Chassis.SteeringWheel.HandsOff.HandsOffSteeringMode`* and `value` can be a boolean that indicates a steering mode status like *`{"B": false}`*.

```
aws iot create-command --command-id command-id \
    --role-arn role-arn \
    --description description \
    --display-name display-name \
    --namespace "AWS-IoT-FleetWise" \
    --mandatory-parameters '[
        {
            "name": name,
            "value": value
        }
   ]'
```

The `CreateCommand` API operation returns a response that contains the ID and ARN (Amazon Resource Name) of the command.

```
{
    "commandId": "HandsOffSteeringMode",
    "commandArn": "arn:aws:iot:ap-south-1:123456789012:command/HandsOffSteeringMode"
}
```

Retrieve information about a command

You can use the GetCommand AWS IoT Core control plane API operation to retrieve information about a command resource.

To get information about a command resource, run the following command. Replace *command-id* with the identifier that was used when creating the command.

```
aws iot get-command --command-id command-id
```

The GetCommand API operation returns a response that contains the following information.

- The ID and ARN (Amazon Resource Name) of the command.
- The date and time when the command was created and last updated.
- The command state which indicates whether it's available to run on the vehicle.
- Any parameters that you specified when creating the command.

```
{
    "commandId": "HandsOffSteeringMode",
    "commandArn": "arn:aws:iot:ap-south-1:123456789012:command/HandsOffSteeringMode"",
    "namespace": "AWS-IoT-FleetWise",
    "mandatoryParameters":[
        {
            "name":
 "$actuatorPath.Vehicle.Chassis.SteeringWheel.HandsOff.HandsOffSteeringMode",
            "value": {"B": false }
        }
    ],
    "createdAt": "2024-03-23T11:24:14.919000-07:00",
    "lastUpdatedAt": "2024-03-23T11:24:14.919000-07:00",
    "deprecated": false,
    "pendingDeletion": false
}
```

List commands in your account

You can use the `ListCommands` AWS IoT Core control plane API operation to list all commands in your account that you created.

To list commands in your account, run the following command. By default, the API returns commands that were created for both namespaces. To filter the list to display only commands that were created for AWS IoT FleetWise, run the following command.

> **Note**
>
> You can also sort the list in ascending or descending order, or filter the list to display only commands that have a specific command parameter name.

```
aws iot list-commands --namespace "AWS-IoT-FleetWise"
```

The `ListCommands` API operation returns a response that contains the following information.

- The ID and ARN (Amazon Resource Name) of the commands.
- The date and time when the command was created and last updated.
- The command state which indicates whether the commands are available to run on the vehicle.

Update or deprecate a command resource

You can use the `UpdateCommand` AWS IoT Core control plane API operation or AWS IoT FleetWise console to update a command resource. You can update the display name and description of a command. You can also deprecate a command if it's not currently being used.

> **Note**
>
> You can't modify the namespace information or the parameters to be used when executing the command.

Update a command (console)

Update a command

To update a command from the console, go to the Commands page of the AWS IoT FleetWise console and perform the following steps.

1. Choose the command that you want to update, and then choose **Edit**.
2. Edit the command details, and then choose **Save changes**.

Deprecate a command

To deprecate a command from the console, go to the Commands page of the AWS IoT FleetWise console and perform the following steps.

1. Choose the command that you want to deprecate, and then choose **Deprecate**.
2. Confirm the deprecation, and then choose **Deprecate**.

Update a command (AWS CLI)

Update a command

To update a command resource, run the following command. Replace *command-id* with the identifier of the command that you want to update, and provide the updated *display-name* and *description*.

```
aws iot update-command \
    --command-id command-id \
    --display-name display-name \
    --description description
```

The UpdateCommand API operation returns the following response.

```
{
    "commandId": "HandsOffSteeringMode",
    "deprecated": false,
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00"
}
```

Deprecate a command

You deprecate a command when you intend to no longer continue using it for your device or when it's outdated. The following example shows how to deprecate a command.

```
aws iot update-command \
    --command-id command-id \
    --deprecated
```

The UpdateCommand API operation returns a response that contains the ID and ARN (Amazon Resource Name) of the command.

```
{
    "commandId": "HandsOffSteeringMode",
    "deprecated": true,
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00"
}
```

Once a command has been deprecated, existing command executions will continue running on the vehicle until they become terminal. To run any new command executions, you must use the UpdateCommand API to restore the command so that it becomes available. For additional information about deprecating and restoring a command and considerations for it, see Deprecate a command resource in the *AWS IoT Core Developer Guide*.

Delete a command resource

You can use the DeleteCommand AWS IoT Core control plane API operation or AWS IoT FleetWise console to delete a command resource.

> **Note**
>
> Deletion actions are permanent and can't be undone. The command will be permanently removed from your account.

Delete a command (console)

To delete a command from the console, go to the Commands page of the AWS IoT FleetWise console and perform the following steps.

1. Choose the command that you want to delete, and then choose **Delete**.
2. Confirm that you want to delete the command, and then choose **Delete**.

Delete a command (AWS CLI)

To delete a command resource, run the following command. Replace *command-id* with the identifier of the command that you want to delete. The following example shows how to delete a command resource.

```
aws iot delete-command --command-id command-id
```

If the deletion request is successful:

- If the command has been deprecated for a duration that's longer than the maximum timeout of 24 hours, the command will be deleted immediately and you'll see a HTTP `statusCode` of 204.
- If the command isn't deprecated, or has been deprecated for a duration shorter than the maximum timeout, the command will be in a `pending deletion` state and you'll see a HTTP `statusCode` of 202. The command will be removed automatically from your account after the maximum timeout of 24 hours.

Start and monitor command executions

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

After you've created a command resource, you can start a command execution on the target vehicle. Once the vehicle starts executing the command, it can start updating the result of the command execution and publish status updates and result information to the MQTT reserved topics. You can then retrieve the status of the command execution and monitor the status of the executions in your account.

This topic shows how you can send a command to your vehicle using the AWS CLI or AWS IoT FleetWise console. It also shows you how to monitor and update the status of the command execution.

Topics

- Get command execution
- List command executions in your account
- Delete a command execution

Send a command (console)

To send a command from the console, go to the Vehicles page of the AWS IoT FleetWise console and perform the following steps.

1. Choose the vehicle that you want to send a command to.
2. Choose **Run command**.
3. Select the command ID.
4. Specify the command execution timeout, and then choose **Run command**.

Send a command (AWS CLI)

You can use the `StartCommandExecution` AWS IoT data plane API operation to send a command to a vehicle. The vehicle then forwards the command to an automotive middleware service (like SOME/IP (Scalable Service-Oriented Middleware over IP)) or publishes it on a vehicle network (like a controller area network (CAN) device interface). The following example uses the AWS CLI.

Topics

- Considerations when sending a command
- Obtain account-specific data plane endpoint
- Send a command example

Considerations when sending a command

When you start a command execution in AWS IoT FleetWise:

- You must provision an AWS IoT thing for the vehicle. For more information, see Provision AWS IoT FleetWise vehicles.
- You must have already created a command with `AWS-IoT-FleetWise` as the namespace and provided a `role-Arn` that grants you permission to create and run commands in AWS IoT FleetWise. For more information, see Create a command resource.

- You can skip the `parameters` field if you choose to use any default values that were specified for the parameters when creating the command. If the `mandatory-parameters` wasn't specified at creation time, or if you want to override any default values by specifying your own values for the parameters, you must specify the `parameters` field. For these additional examples, see Command usage scenarios.
- You can specify up to three name-value pairs for the `mandatory-parameters` field. However, when executing the command on the vehicle, only one name-value pair is accepted, and the `name` field must use the fully qualified name with the `$actuatorPath.` prefix.

Obtain account-specific data plane endpoint

Before you run the API command, you must obtain the account-specific endpoint URL for the `iot:Jobs` endpoint. For example, if you run this command:

```
aws iot describe-endpoint --endpoint-type iot:Jobs
```

It will return the account-specfic endpoint URL as shown in the sample response below.

```
{
    "endpointAddress": "<account-specific-prefix>.jobs.iot.<region>.amazonaws.com"
}
```

Send a command example

To send a command to a vehicle, run the following command.

- Replace *`command-arn`* with the ARN for the command that you want to execute. You can obtain this information from the response of the `create-command` CLI command.
- Replace *`target-arn`* with the ARN for the target device, or AWS IoT thing, for which you want to execute the command.

> **Note**
>
> You can specify the target ARN of an AWS IoT thing (AWS IoT FleetWise vehicle). Thing groups and fleets aren't currently supported.

- Replace *endpoint-url* with the account-specific endpoint that you obtained in Obtain account-specific data plane endpoint, prefixed by `https://`, for example, `https://123456789012abcd.jobs.iot.ap-south-1.amazonaws.com`.
- Replace *name* and *value* with the `mandatory-parameters` field that you specified when you created the command using the `create-command` CLI.

  The `name` field is the fully qualified name as defined in the signal catalog with `$actuatorPath.` as the prefix. For example, `name` can be *$actuatorPath.Vehicle.Chassis.SteeringWheel.HandsOff.HandsOffSteeringMode* and `value` can be a boolean that indicates a steering mode status like *{"B": false}*.
- (Optional) You can also specify an additional parameter, `executionTimeoutSeconds`. This optional field specifies the time in seconds within which the device must respond with the execution result. You can configure the timeout to a maximum value of 24 hours.

  When the command execution has been created, a timer starts. Before the timer expires, if the command execution status doesn't change to a status that makes it terminal, such as `SUCCEEDED` or `FAILED`, then the status automatically changes to `TIMED_OUT`.

  > **Note**
  >
  > The device can also report a `TIMED_OUT` status, or override this status to a status such as `SUCCEEDED`, `FAILED`, or `REJECTED`, and the command execution will become terminal. For more information, see Command execution timeout status.

```
aws iot-jobs-data start-command-execution \
    --command-arn command-arn \
    --target-arn target-arn \
    --execution-timeout-seconds 30 \
    --endpoint-url endpoint-url \
    --parameters '[
        {
            "name": name,
            "value": value
        }
   ]'
```

The `StartCommandExecution` API operation returns a command execution ID. You can use this ID to query the command execution status, details, and command execution history.

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542"
 }
```

After you run the command, your devices will receive a notification that contains the following information. The `issued_timestamp_ms` field corresponds to the time that the `StartCommandExecution` API was invoked. The `timeout_ms` corresponds to the time out value that's configured using the `executionTimeoutSeconds` parameter when invoking the `StartCommandExecution` API.

```
timeout_ms: 9000000
issued_timestamp_ms: 1723847831317
```

Update command execution result

To update the status of the command execution, your device must have established an MQTT connection and subscribed to the following commands request topic.

In this example, replace *<device-id>* with the unique identifier of your target device, which can be the `VehicleId` or the thing name, and *<execution-id>* with the identifier for the command execution.

Note

- The payload must use the protobuf format.
- It is optional for your devices to subscribe to the `/accepted` and `/rejected` response topics. Your devices will receive these response messages even if they haven't explicitly subscribed to them.

```
// Request topic
$aws/devices/<DeviceID>/command_executions/+/request/protobuf

// Response topics (Optional)
```

```
$aws/devices/<DeviceID>/command_executions/<ExecutionId>/response/accepted/protobuf
$aws/devices/<DeviceID>/command_executions/<ExecutionId>/response/rejected/protobuf
```

Your device can publish a message to the commands response topic. After processing the command, it sends a protobuf-encoded response to this topic. The *<DeviceID>* field must match the corresponding field in the request topic.

```
$aws/devices/<DeviceID>/command_executions/<ExecutionId>/response/<PayloadFormat>
```

After your device publishes a response to this topic, you can retrieve the updated status information using the `GetCommandExecution` API. The status of a command execution can be any of those listed here.

- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`
- `REJECTED`
- `TIMED_OUT`

Note that a command execution in any of the statuses `SUCCEEDED`, `FAILED`, and `REJECTED` is terminal, and the status is reported by the device. When a command execution is terminal, this means that no further updates will be made to its status or related fields. A `TIMED_OUT` status may be reported by the device or the cloud. If reported by the cloud, an update of the status reason field may later be made by the device.

For example, the following shows a sample MQTT message that's published by the device.

Note

For the command execution status, if your devices use the `statusReason` object to publish the status information, you must make sure that:

- The `reasonCode` uses the pattern `[A-Z0-9_-]+`, and it does not exceed 64 characters in length.
- The `reasonDescription` doesn't exceed 1,024 characters in length. It can use any characters except control characters such as new lines.

```
{
    "deviceId": "",
    "executionId": "",
    "status": "CREATED",
    "statusReason": {
        "reasonCode": "",
        "reasonDescription": ""
    }
}
```

For an example that shows how you can use the AWS IoT Core MQTT test client to subscribe to the topics and see the command execution messages, see Viewing commands updates using the MQTT test client in the *AWS IoT Core developer guide*.

Get command execution

You can use the `GetCommandExecution` AWS IoT control plane API operation to retrieve information about a command execution. You must have already executed this command using the `StartCommandExecution` API operation.

To retrieve the metadata of an executed command, run the following command.

- Replace *`execution-id`* with the ID of the command. You can obtain this information from the response of the `start-command-execution` CLI command.
- Replace *`target-arn`* with the ARN for the target vehicle, or AWS IoT thing, for which you want to execute the command.

```
aws iot get-command-execution --execution-id execution-id \
    --target-arn target-arn
```

The `GetCommandExecution` API operation returns a response that contains information about the ARN of the command execution, the execution status, and the time when the command started executing and when it completed. The following code shows a sample response from the API request.

To provide additional context about the status of each command execution, the commands feature provides a `statusReason` object. The object contains two fields, `reasonCode` and `reasonDescription`. Using these fields, your devices can provide additional information about

the status of a command execution. This information will override any default `reasonCode` and `reasonDescription` that's reported from the cloud.

To report this information, your devices can publish the updated status information to the cloud. Then, when you retrieve the command execution status using the `GetCommandExecution` API, you'll see the latest status codes.

> **Note**
>
> The `completedAt` field in the execution response corresponds to the time when the device reports a terminal status to the cloud. In the case of `TIMED_OUT` status, this field will be set only when the device reports A timeout. When the `TIMED_OUT` status is set by the cloud, the `TIMED_OUT` status is not updated. For more information about the time out behavior, see Command execution timeout status.

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542",
    "commandArn": "arn:aws:iot:ap-south-1:123456789012:command/LockDoor",
    "targetArn": "arn:aws:iot:ap-south-1:123456789012:thing/myFrontDoor",
    "status": "SUCCEEDED",
    "statusReason": {
        "reasonCode": "65536",
        "reasonDescription": "SUCCESS"
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "completedAt": "2024-03-23T00:50:10.095000-07:00",
    "Parameters": '{
         "$actuatorPath.Vehicle.Chassis.SteeringWheel.HandsOff.HandsOffSteeringMode":

         { "B": true }
    }'
}
```

List command executions in your account

Use the `ListCommandExecutions` AWS IoT Core control plane HTTP API operation to list all command executions in your account. The example uses the AWS CLI.

Topics

Considerations when listing command executions

The following are some considerations when using the `ListCommandExecutions` API.

- You must specify at least the `targetArn` or the `commandArn` depending on whether you want to list executions for a particular command or a target vehicle. The API request cannot be empty and cannot contain both fields in the same request.
- You must provide only the `startedTimeFilter` or the `completedTimeFilter` information. The API request cannot be empty and cannot contain both fields in the same request. You can use the `before` and `after` fields of the object to list command executions that were either created or completed within a specific timeframe.
- Both the `before` and `after` fields must not be greater than the current time. By default, if you don't specify any value, the `before` field is the current time and `after` field is current time - 6 months. That is, depending on the filter that you use, the API will list all executions that were either created or completed within the last six months.
- You can use the `sort-order` parameter to specify whether you want to list the executions in the ascending order. By default, the executions will be listed in the descending order if you don't specify this field.
- You cannot filter the command executions based on their status when listing command executions for a command ARN.

List command executions example

The following example shows you how to list command executions in your AWS account.

When running the command, you must specify whether to filter the list to display only command executions that were created for a particular device using the `targetArn`, or executions for a particular command specified using the `commandArn`.

In this example, replace:

- *`<target-arn>`* with the Amazon Resource Number (ARN) of the device for which you're targeting the execution, such as `arn:aws:iot:`*`us-east-1`*`:`*`123456789012`*`:thing/`*`b8e4157c98f332cffb37627f`*.

- *<target-arn>* with the Amazon Resource Number (ARN) of the device for which you're targeting the execution, such as `arn:aws:iot:`*us-east-1*`:`*123456789012*`:thing/`*b8e4157c98f332cffb37627f*.
- *<after>* with the time after which you want to list the executions that were created, for example, `2024-11-01T03:00`.

```
aws iot list-command-executions \
--target-arn <target-arn> \
--started-time-filter '{after=<after>}' \
--sort-order "ASCENDING"
```

Running this command generates a response that contains a list of command executions that you created, and the time when the executions started executing, and when it completed. It also provides status information, and the `statusReason` object that contains additional information about the status.

```
{
    "commandExecutions": [
        {
            "commandArn": "arn:aws:iot:us-east-1:123456789012:command/TestMe002",
            "executionId": "b2b654ca-1a71-427f-9669-e74ae9d92d24",
            "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/
b8e4157c98f332cffb37627f",
            "status": "TIMED_OUT",
            "createdAt": "2024-11-24T14:39:25.791000-08:00",
            "startedAt": "2024-11-24T14:39:25.791000-08:00"
        },
        {
            "commandArn": "arn:aws:iot:us-east-1:123456789012:command/TestMe002",
            "executionId": "34bf015f-ef0f-4453-acd0-9cca2d42a48f",
            "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/
b8e4157c98f332cffb37627f",
            "status": "IN_PROGRESS",
            "createdAt": "2024-11-24T14:05:36.021000-08:00",
            "startedAt": "2024-11-24T14:05:36.021000-08:00"
        }
    ]
}
```

Delete a command execution

If you no longer want to use a command execution, you can remove it permanently from your account.

> **Note**
>
> A command execution can be deleted only if it has entered a terminal status, such as `SUCCEEDED`, `FAILED`, or `REJECTED`.

The following example shows you how to delete a command execution using the `delete-command-execution` AWS CLI command. Replace *`<execution-id>`* with the identifier of the command execution that you're deleting.

```
aws iot delete-command-execution --execution-id <execution-id>
```

If the API request is successful, then the command execution generates a status code of 200. You can use the `GetCommandExecution` API to verify that the command execution no longer exists in your account.

Example: Using commands to control a vehicle steering mode (AWS CLI)

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

The following example shows you how to use the commands feature using the AWS CLI. This example uses an AWS IoT FleetWise vehicle as a target device to show how you can send a command to remotely control the steering mode.

Topics

- Overview of vehicle steering mode example

Overview of vehicle steering mode example

In this example, you'll:

1. Create a command resource for the operation using the `create-command` AWS CLI to change the steering mode of the vehicle.
2. Retrieve information about the command, such as the time when it was created or last updated using the `get-command` AWS CLI.
3. Send the command to the vehicle using the `start-command-execution` AWS CLI with the steering mode as a mandatory parameter, which will then get executed on the device.
4. Get the result of the command execution using the `get-command-execution` AWS CLI. You can check when the execution completes, and retrieve additional details such as the execution result, and the time it took to complete executing the command.
5. Perform clean up activities by removing any commands and command executions that you no longer want to use.

Prerequisites

Before you run this example:

- Provision your AWS IoT FleetWise vehicle as an AWS IoT thing in the AWS IoT registry. You must also add a certificate to your thing and activate it, and attach a policy to your thing. Your device can then connect to the cloud and execute the commands. For more information, see Provision vehicles.
- Create an IAM user and an IAM policy that grants you permission to perform the API operations for using commands, as shown in IAM policy for using remote commands.

IAM policy for using remote commands

The following table shows a sample IAM policy that grants access to all the control plane and data plane API operations for the commands feature. The user of the application will have permissions to perform all remote command API operations, as shown in the table.

API operation

API action	Control/data plane	Protocol	Description	Resource
`CreateCommand`	Control plane	HTTP	Creates a command resource	• command
`GetCommand`	Control plane	HTTP	Retrieves information about a command	• command
`UpdateCommand`	Control plane	HTTP	Updates information about a command or to deprecate it	• command
`ListCommands`	Control plane	HTTP	Lists commands in your account	• command
`DeleteCommand`	Control plane	HTTP	Deletes a command	• command
`StartCommandExecution`	Data plane	HTTP	Starts executing a command	• command • thing
`UpdateCommandExecution`	Data plane	MQTT	Update a command execution	• command • thing
`GetCommandExecution`	Control plane	HTTP	Retrieves information about a command execution	• command • thing
`ListCommandExecutions`	Control plane	HTTP	Lists command executions in your account	• command • thing

API action	Control/data plane	Protocol	Description	Resource
`DeleteCommandExecution`	Control plane	HTTP	Deletes a command execution	• command • thing

In this example, replace:

- *us-east-1* with your AWS Region, such as `ap-south-1`.
- *111122223333* with your AWS account number, such as `57EXAMPLE833`.
- *command-id*, *command-id1*, and *command-id2* with your unique command identifier, such as `LockDoor` or `TurnOffAC`.
- *thing-name* with your AWS IoT thing name, such as `my_car`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iot:CreateCommand",
                "iot:GetCommand",
                "iot:ListCommands",
                "iot:UpdateCommand",
                "iot:DeleteCommand"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:command/command-id1",
                "arn:aws:iot:us-east-1:111122223333:command/command-id2"
            ]
        },
        {
            "Action": [
                "iot:GetCommandExecution",
                "iot:ListCommandExecutions",
```

```
                "iot:DeleteCommandExecution"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:command/command-id",
                "arn:aws:iot:us-east-1:111122223333:thing/thing-name"
            ]
        },
        {
            "Action": "iot:StartCommandExecution",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:command/command-id",
                "arn:aws:iot:us-east-1:111122223333:thing/thing-name"
            ]
        }
    ]
}
```

Run AWS IoT commands (AWS CLI)

The following shows how you can use the AWS CLI to perform commands operations and change the vehicle steering mode.

1. **Create a command resource for the steering mode operation**

   Create the command that you want to send to your device using the `create-command` CLI. In this example, specify:

   - `command-id` as *`TurnOffSteeringMode`*
   - `role-arn` as "`arn:aws:iam:`*`accountId`*`:role/`*`FwCommandExecutionRole`*" The `role-arn` must be provided, as it is the IAM role that grants permissions to create and run commands on your vehicle. For more information, see Grant AWS IoT Device Management permission to generate the payload for commands with AWS IoT FleetWise.
   - `display-name` as "*`Turn off steering mode`*"
   - `namespace` must be `AWS-IoT-FleetWise`
   - `mandatory-parameters` as a name-value pair, with `name` as "*`$actuatorPath.Vehicle.Chassis.SteeringWheel.TurnOffSteeringMode`*" and defaultValue as *`{ "S": "true" }`*

> **Note**
>
> You can also create a command without specifying any mandatory parameters. You must then specify the parameters to use when executing the command using the `start-command-execution` CLI. For an example, see Command usage scenarios.

> **Important**
>
> When using the `AWS-IoT-FleetWise` namespace, you must ensure that the `Name` field specified as part of the `mandatory-parameters` use the `$actuatorPath.` prefix, and the `Value` field must use the string data type.

```
aws iot create-command \
    --command-id TurnOffSteeringMode \
    --role-arn "arn:aws:iam:accountId:role/FwCommandExecutionRole" \
    --display-name "Turn off steering mode" \
    --namespace AWS-IoT-FleetWise \
    --mandatory-parameters '[
      {
        "name": "$actuatorPath.Vehicle.Chassis.SteeringWheel.TurnOffSteeringMode",
        "defaultValue": { "S": "true" }
      }
    ]'
```

The following output shows a sample response from the CLI, where `ap-south-1` and `123456789012` are examples of the AWS Region and AWS account ID.

```
{
    "commandId": "TurnOffSteeringMode",
    "commandArn": "arn:aws:iot:ap-south-1:123456789012:command/TurnOffSteeringMode"
}
```

For additional examples on using this command, see Create a command resource.

2. **Retrieve information about the command**

Run the following command to retrieve information about the command, where `command-id` is the command ID in the output of the `create-command` operation from above.

> **Note**
>
> If you create more than one command, you can use the `ListCommands` API to list all commands in your account, and then use the `GetCommand` API to obtain additional information about a specific command. For more information, see List commands in your account.

```
aws iot get-command --command-id TurnOffSteeringMode
```

Running this command generates the following response. You'll see the time when the command was created and when it was last updated, any parameters that you specified, and whether the command is available to run on the device.

```
{
    "commandId": "TurnOffSteeringMode",
    "commandArn": "arn:aws:iot:ap-south-1:123456789012:command/
TurnOffSteeringMode",
    "namespace": "AWS-IoT-FleetWise",
    "mandatoryParameters":[
        {
            "name":
 "$actuatorPath.Vehicle.Chassis.SteeringWheel.TurnOffSteeringMode",
            "defaultValue": {"S": "true" }
        }
    ],
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "lastUpdatedAt": "2024-03-23T00:50:10.095000-07:00",
    "deprecated": false
}
```

For additional examples on using this command, see Retrieve information about a command.

3. **Start the command execution**

   Run the following command to start executing the command, where `command-arn` is the command ARN in the output of the `get-command` operation from above. The `target-arn` is the ARN of the target device for which you're executing the command, for example, *myVehicle*.

   In this example, since you provided default values for the parameters when creating the command, the `start-command-execution` CLI can use these values when executing the command. You can also choose to override the default value by specifying a different value for the parameters when using the CLI.

   ```
   aws iot-data start-command-execution \
       --command-arn arn:aws:iot:ap-south-1:123456789012:command/TurnOffSteeringMode \
       --target-arn arn:aws:iot:ap-south-1:123456789012:thing/myVehicle
   ```

   Running this command returns a command execution ID. You can use this ID to query the command execution status, details, and command execution history.

   ```
   {
       "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542"
   }
   ```

   For additional examples on using the CLI, see Send a command (AWS CLI).

4. **Retrieve information about the command execution**

   Run the following command to retrieve information about the command that you executed on the target device. Specify the `execution-id`, which you obtained as output of the `start-command-execution` operation from above, and the `target-arn`, which is the ARN of the device that you're targeting.

   > **Note**
   >
   > - To obtain the latest status information, your devices must have published the updated status information to the MQTT reserved response topic for commands using the `UpdateCommandExecution` MQTT API. For more information, see Update command execution result.

- If you start more than one command execution, you can use the `ListCommandExecutions` API to list all command executions in your account, and then use the `GetCommandExecution` API to obtain additional information about a specific execution. For more information, see List command executions in your account.

```
aws iot get-command-execution \
    --execution-id <"07e4b780-7eca-4ffd-b772-b76358da5542"> \
    --target-arn arn:aws:iot:us-east-1:<account>:thing/myVehicle
```

Running this command returns information about the command execution, the execution status, the time when it started executing, and the time when it was completed. For example, the following response shows that the command execution succeeded on the target device and the steering mode was turned off.

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542",
    "commandArn": "arn:aws:iot:ap-south-1:123456789012:command/
TurnOffSteeringMode",
    "targetArn": "arn:aws:iot:ap-south-1:123456789012:thing/myVehicle",
    "result": "SUCCEEDED",
     "statusReason": {
        "reasonCode": "65536",
        "reasonDescription": "SUCCESS"
    },
    "result": {
        "KeyName": {
            "S": "",
            "B": true,
            "BIN": null
        }
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "completedAt": "2024-03-23T00:50:10.095000-07:00",
    "parameters": '{
         "$actuatorPath.Vehicle.Chassis.SteeringWheel.TurnOffSteeringMode":
         { "S": "true" }
    }'
}
```

Cleaning up

Now that you've created a command and executed it on your device, if you no longer intend to use this command, you can delete it. Any pending command executions that are in progress will continue to run without getting impacted by the deletion request.

> **Note**
>
> Alternatively, you can also deprecate a command if it's outdated and you might need to use it later to run on the target device.

1. **(Optional) Deprecate the command resource**

   Run the following command to deprecate the command, where `command-id` is the command ID in the output of the `get-command` operation from above.

   ```
   aws iot update-command \
      --command-id TurnOffSteeringMode \
      --deprecated
   ```

   Running this command returns an output that shows the command has been deprecated. You can also use the CLI to restore the command.

   > **Note**
   >
   > You can also use the `update-command` CLI to update the display name and description of a command. For additional information, see Update or deprecate a command resource.

   ```
   {
       "commandId": "TurnOffSteeringMode",
       "deprecated": true,
       "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00"
   }
   ```

2. **Delete the command**

   Run the following command to delete the command, specified by the `command-id`.

   > **Note**
   >
   > The deletion action is permanent and can't be undone.

   ```
   aws iot delete-command --command-id TurnOffSteeringMode
   ```

   If the deletion request is successful, you'll see a HTTP `statusCode` of 202 or 204 depending on whether you marked the command for deprecation and when it was deprecated. For more information and an example, see Delete a command resource.

   You can use the `get-command` CLI to verify that the command has been removed from your account.

3. **(Optional) Delete the command executions**

   By default, all command executions will be deleted in six months from the date that you create them. You can view this information using the `timeToLive` parameter from the `GetCommandExecution` API.

   Alternatively, if your command execution has become terminal, such as when your execution status is one of `SUCCEEDED`, `FAILED`, or `REJECTED`, you can delete the command execution. Run the following command to delete the execution, where `execution-id` is the Execution ID in the output of the `get-command-execution` operation from above.

   ```
   aws iot delete-command-execution \
               --execution-id "07e4b780-7eca-4ffd-b772-b76358da5542"
   ```

   You can use the `get-command-execution` CLI to verify that the command execution has been removed from your account.

Command usage scenarios

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

When using the commands feature, you can create and run commands in the following scenarios:

- You can omit the parameters during creation and specify only the command ID. In this case, you need to specify the parameters to be used when running the command on the target device.
- You can specify one or more parameters, and configure default values for them when creating the command. Providing default values will help protect you from sending inaccurate commands.
- You can specify one or more parameters, and configure values for them when creating the command. More than one parameter can be provided but only one of them will be executed, and the `Name` field of this parameter must use the `$actuatorPath` prefix.

This section provides some usage scenarios for the `CreateCommand` and the `StartCommandExecution` API and using the parameters. It also shows you some examples of using commands with state templates.

Topics

Creating a command with no parameters

The following use case shows how you can use the `CreateCommand` API or the `create-command` CLI to create a command with no parameters. When you create a command, you only need to provide a command ID and a role ARN.

This use case is especially useful in recurrent use cases, such as when you want to send the same command multiple times to a vehicle. In this case, the command is not tied to a specific actuator and gives you the flexibility to execute the command on any actuator. You must specify the parameters at run time instead when executing the command using the `StartCommandExecution` API or the `start-command-execution` CLI, which includes the actuators and physical signal values.

Creating a command without `mandatory-parameters` input

This use case shows how to create a command without any mandatory parameters input.

```
aws iot create-command \
    --command-id "UserJourney1" \
    --role-arn "arn:aws:iam:accountId:role/FwCommandExecutionRole" \
    --description "UserJourney1 - No mandatory parameters" \
    --namespace "AWS-IoT-FleetWise"
```

Running a command created without `mandatory-parameters` input

In this first example, the command that was created above allows you to execute a command on any actuator without restrictions. To set `actuator1` to a value of 10, run:

```
aws iot-jobs-data start-command-execution \
    --command-arn arn:aws:iot:region:111122223333:command/UserJourney1 \
    --target-arn arn:aws:iot:region:111122223333:thing/target-vehicle \
    --parameters '{
        "$actuatorPath.Vehicle.actuator1": {"S": "10"}
    }'
```

Similarly, you can run a command that sets `actuator3` to a value of `true`.

```
aws iot-jobs-data start-command-execution \
    --command-arn arn:aws:iot:region:111122223333:command/UserJourney1 \
    --target-arn arn:aws:iot:region:111122223333:thing/target-vehicle \
    --parameters '{
        "$actuatorPath.Vehicle.actuator3": {"S": "true"}
    }'
```

Creating a command with default values for parameters

This command only allows you to execute a command on the specified actuator. Providing default values will help protect you from sending inaccurate commands. For example, a `LockDoor` command that locks and unlocks doors can be configured with a default value to avoid the command from accidentally unlocking doors.

This use case is especially useful when you want to send the same command multiple times and perform different actions on the same actuator, such as locking and unlocking the doors of a vehicle. If you want to set the actuator to the default value, then you don't need to pass qny `parameters` to the `start-command-execution` CLI. If you do specify a different value for the `parameters` in the `start-command-execution` CLI, it will override the default value.

Creating a command with default values for `mandatory-parameters`

The following command shows how to provide a default value for actuator1.

```
aws iot create-command \
    --command-id "UserJourney2" \
    --namespace "AWS-IoT-FleetWise" \
    --role-arn "arn:aws:iam:accountId:role/FwCommandExecutionRole" \
    --mandatory-parameters '[
        {
            "name": "$actuatorPath.Vehicle.actuator1",
            "defaultValue": {"S": "0"}
        }
    ]'
```

Running a command created with default values for `mandatory-parameters`

The command `UserJourney2` allows you to execute a command without the need to pass an input value during runtime. In this case, the execution at runtime will use the default values specified during creation.

```
aws iot-data start-command-execution \
    --command-arn arn:aws:iot:region:111122223333:command/UserJourney3 \
    --target-arn arn:aws:iot:region:111122223333:thing/target-vehicle
```

You can also pass a different value for the same actuator, actuator1, during runtime, which will override the default value.

```
aws iot-jobs-data start-command-execution \
    --command-arn arn:aws:iot:region:111122223333:command/UserJourney3 \
    --target-arn arn:aws:iot:region:111122223333:thing/target-vehicle \
    --parameters '{
        "$actuatorPath.Vehicle.actuator1": {"S": "139"}
    }'
```

Creating a command with parameter values

This command only allows you to execute a command on the specified actuator. It also forces you to set a value for the actuator during runtime.

This use case is especially useful when you want the end user to only perform certain specified actions on some of the actuators when running it on the vehicle.

> **Note**
>
> You can have more than name-value pairs for the `mandatory-parameters` input, with default values for some or all of them. At runtime, you can then determine the parameter that you want to use when running on the actuator, provided the actuator name uses the fully-qualified name with the `$actuatorPath.` prefix.

Creating command without default values for `mandatory-parameters`

This command only allows you to execute a command on the specified actuator. It also forces you to set a value for the actuator during runtime.

```
aws iot create-command \
    --command-id "UserJourney2" \
    --namespace "AWS-IoT-FleetWise" \
    --role-arn "arn:aws:iam:accountId:role/FwCommandExecutionRole" \
    --mandatory-parameters '[
        {
            "name": "$actuatorPath.Vehicle.actuator1"
        }
    ]'
```

Running a command created without default values for `mandatory-parameters`

When running the command, in this case, you must specify a value for actuator1. The command execution shown below will successfully set the value of `actuator1` to `10`.

```
aws iot-data start-command-execution \
    --command-arn arn:aws:iot:region:111122223333:command/UserJourney2 \
    --target-arn arn:aws:iot:region:111122223333:thing/target-vehicle \
    --parameters '{
        "$actuatorPath.Vehicle.actuator1": {"S": "10"}
    }'
```

Using commands with state templates

You can also use the commands API operations for state data collection and processing. For example, you can fetch a one-time state snapshot or to activate or deactivate state templates to start or stop collecting vehicle state data. The following examples show how to use the commands feature with state templates. For more information, see State template operations for data collection and processing

> **Note**
>
> The Name field specified as part of the `mandatory-parameters` input must use the `$stateTemplate` prefix.

Example 1: Creating commands for state templates with default values

This example shows how to use the `create-command` CLI to activate state templates.

```
aws iot create-command \
    --command-id <COMMAND_ID> \
    --display-name "Activate State Template" \
    --namespace AWS-IoT-FleetWise \
    --mandatory-parameters '[
      {
          "name": "$stateTemplate.name"
      },
      {
          "name": "$stateTemplate.operation",
          "defaultValue": {"S": "activate"}
```

```
      }
    ]'
```

Similarly, the following command shows an example of how you can use the `start-command-execution` CLI for state templates.

```
aws iot-data start-command-execution \
    --command-arn arn:aws:iot:region:111122223333:command/<COMMAND_ID> \
    --target-arn arn:aws:iot:region:111122223333:thing/<VEHICLE_NAME> \
    --parameters '{
       "$stateTemplate.name": {"S": "ST345"}
    }'
```

Example 2: Creating commands for state templates without default values

The following command creates multiple state templates without default values for any of the parameters. It forces you to run the command with these parameters and the values for them.

```
aws iot create-command \
    --command-id <COMMAND_ID> \
    --display-name "Activate State Template" \
    --namespace AWS-IoT-FleetWise \
    --mandatory-parameters '[
      {
          "name": "$stateTemplate.name",
          "defaultValue": {"S": "ST123"}
      },
      {
          "name": "$stateTemplate.operation",
          "defaultValue": {"S": "activate"}
      },
      {
          "name": "$stateTemplate.deactivateAfterSeconds",
          "defaultValue": {"L": "120"}
      }
    ]'
```

The following command shows how you can use the `start-command-execution` CLI for the example above.

```
aws iot-data start-command-execution \
```

```
    --command-arn arn:aws:iot:region:111122223333:command/<COMMAND_ID> \
    --target-arn arn:aws:iot:region:111122223333:thing/<VEHICLE_NAME> \
    --parameters '{
        "$stateTemplate.name": {"S": "ST345"},
        "$stateTemplate.operation": {"S": "activate"},
        "$stateTemplate.deactivateAfterSeconds" : {"L": "120"}
```

Monitor the last known state of your vehicles

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can monitor the last known state of your vehicles in near-real time by creating state templates and associating them with your vehicles. Vehicles associated with state templates stream telemetry data with either an onChange or `periodic` update strategy. With an on-change update strategy, the associated vehicles stream telemetry data when there is a change. During a periodic update strategy, the associated vehicles stream telemetry data during a specified time period.

With on-demand operations, you can request the current vehicle state at one time (fetch). You can also activate or deactivate previously deployed state templates to start or stop reporting vehicle state data. Last known state operations are performed using the AWS IoT command APIs.

Each state template contains the following information.

`name`

The unique alias of the state template.

`signalCatalogArn`

The Amazon Resource Name (ARN) of the signal catalog associated with the state template.

`stateTemplateProperties`

A list of signals from which data is collected. The state template properties determine the specific signal updates the vehicle sends to the cloud.

`dataExtraDimensions`

A list of vehicle attributes to be included in the protocol buffers (Protobuf) encoded processed data.

`metadataExtraDimensions`

A list of vehicle attributes to be published with the processed data as an MQTT 5 user property.

`id`

A unique, service-generated identifier.

For methods to collect data sent by a vehicle that uses the Edge Agent for AWS IoT FleetWise software, see Process last known state vehicle data using MQTT messaging. For more information about how to associate a state template with a vehicle, see Create an AWS IoT FleetWise vehicle.

Topics

- Create an AWS IoT FleetWise state template
- Update an AWS IoT FleetWise state template
- Delete an AWS IoT FleetWise state template
- Get AWS IoT FleetWise state template information
- State template operations for data collection and processing

Create an AWS IoT FleetWise state template

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the AWS IoT FleetWise API or console to create a state template. State templates provide a mechanism to track the state of your vehicles. The Edge Agent for AWS IoT FleetWise software that runs on the vehicle collects and sends signal updates to the cloud.

Create a state template (console)

You can use the AWS IoT FleetWise console to create a state template.

To create a state template

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **State templates**.
3. On the **State templates** page, choose **Create state template**.

4. In **State template details**, enter a name for the state template and optionally enter a description.
5. In **Choose signals**, add signals that you want to fetch vehicle status information from.
6. Choose **Create state template**.

After you successfully create a state template, you will see it listed on the **State templates** page. You can now associate it with a vehicle.

Create a state template (AWS CLI)

You can use the CreateStateTemplate API operation to create a state template. The following example uses the AWS CLI.

To create a state template, run the following command.

Replace *create-state-template* with the name of the .json file that contains the state template configuration.

```
aws iotfleetwise create-state-template \
    --cli-input-json file://create-state-template.json
```

Example state template configuration

`stateTemplateProperties` should contain the fully qualified names of the signals.

`dataExtraDimensions` and `metadataExtraDimensions` should contain the fully qualified names of the vehicle attributes. The dimensions specified replace any existing dimension values in the state template.

```
{
    "name": "state-template-name",
    "signalCatalogArn": "arn:aws:iotfleetwise:us-east-1:account:signal-catalog/catalog-name",
    "stateTemplateProperties": [
        "Vehicle.Signal.One",
        "Vehicle.Signal.Two"
    ],
    "dataExtraDimensions": [
        "Vehicle.Attribute.One",
```

```
        "Vehicle.Attribute.Two"
    ],
    "metadataExtraDimensions": [
        "Vehicle.Attribute.Three",
        "Vehicle.Attribute.Four"
    ]
}
```

Associate an AWS IoT FleetWise state template with a vehicle

Associate a state template with a vehicle (console)

You can use the AWS IoT FleetWise console to add associated state templates to a vehicle.

To associate a state template

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicles**.
3. Choose a vehicle from the list to open its details page.
4. On the **State templates** tab, choose **Manage state templates**.
5. Choose **Add state template**.
6. Select a state template and choose its reporting method.
    a. **On change** – The state template will report changes to the vehicle's state.
    b. **Periodic** – The state template will report updates on the specified time interval.
7. Choose **Save changes**.

Associate an AWS IoT FleetWise state template with a vehicle (AWS CLI)

Associate the created state template with a vehicle to allow the collection of state updates from the vehicle to the cloud. To do this, use:

- When creating a vehicle, use the `stateTemplates` field of the `create-vehicle` command. For more information, see Create an AWS IoT FleetWise vehicle.
- When updating a vehicle, use the `stateTemplatesToAdd` or `stateTemplatesToRemove` fields of the `update-vehicle` command. For more information, see Update an AWS IoT FleetWise vehicle.

Update an AWS IoT FleetWise state template

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the UpdateStateTemplate API operation or AWS IoT FleetWise console to update an existing state template.

Update a state template (console)

To update a state template from the console, go to the State templates page of the AWS IoT FleetWise console and perform the following steps.

1. Choose the state template that you want to update, and then choose **Edit**.
2. Edit the state template details, and then choose **Save changes**.

Update a state template (AWS CLI)

To update a state template, run the following command.

Replace *update-state-template* with the name of the .json file that contains the configuration of the state template.

```
aws iotfleetwise update-state-template \
    --cli-input-json file://update-state-template.json
```

Example state template configuration

The `stateTemplateProperties` should contain the fully qualified names of the signals.

The `dataExtraDimensions` and `metadataExtraDimensions` should contain the fully qualified names of the vehicle attributes.

```
{
    "identifier": "state-template-name",
    "stateTemplatePropertiesToAdd": [
        "Vehicle.Signal.Three"
```

```
    ],
    "stateTemplatePropertiesToRemove": [
        "Vehicle.Signal.One"
    ],
    "dataExtraDimensions": [
        "Vehicle.Attribute.One",
        "Vehicle.Attribute.Two"
    ],
    "metadataExtraDimensions": [
        "Vehicle.Attribute.Three",
        "Vehicle.Attribute.Four"
    ]
}
```

Delete an AWS IoT FleetWise state template

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the DeleteStateTemplate API operation or AWS IoT FleetWise console to delete a state template.

Delete a state template (console)

To delete a state template from the console, go to the State templates page of the AWS IoT FleetWise console and perform the following steps.

1. Choose the state template that you want to delete, and then choose **Delete**.
2. Confirm that you want to delete the state template, and then choose **Delete**.

Delete a state template (AWS CLI)

To delete a state template, run the following command.

Replace *identifier* with the name or ID of the state template.

```
aws iotfleetwise delete-state-template \
```

```
    --identifier idenitfier
```

Get AWS IoT FleetWise state template information

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can use the GetStateTemplate API operation to retrieve information about a state template. The following example uses the AWS CLI.

Replace *identifier* with the name of the state template.

```
aws iotfleetwise get-state-template \
    --identifier idenitfier
```

You can use the ListStateTemplates API operation to retrieve a list of your created state templates. The following example uses the AWS CLI.

```
aws iotfleetwise list-state-templates
```

If you enabled encryption using a customer managed AWS KMS key, include the following policy statement so that your role can invoke the `GetStateTemplate` or `ListStateTemplates` API operations.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:111122223333:key/KMS_KEY_ID"
```

```
            ]
        }
    ]
}
```

State template operations for data collection and processing

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

The following sections describe how to use state templates to activate and deactivate data collection, perform a fetch operation, and process state data from your vehicles.

Topics

- Activate and deactivate state data collection using state templates
- Fetch a vehicle state snapshot using state templates
- Process last known state vehicle data using MQTT messaging

Activate and deactivate state data collection using state templates

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

The following sections describe how to activate and deactivate data ingestion with state templates using the AWS CLI.

> **Important**
>
> Before you start, make sure that you already created a state template, and associated it and its update strategy with a vehicle.

You must activate a state template so the Edge Agent can send signal updates to the cloud.

To perform these operations with state templates, first create a command resource and then start the command execution on the vehicle. The following section describes how to use this API and how to activate and deactivate data ingestion.

Topics

Using the `CreateCommand` API

Create a command resource in the "`AWS-IoTFleetwise`" namespace, and use the following parameters when you create or send a command resource for a state template:

- `$stateTemplate.name` – The name of the state template on which to perform the operation. The state template must be applied to the vehicle before you can perform an operation. For more information, see Associate an AWS IoT FleetWise state template with a vehicle.
- `$stateTemplate.operation` – The operation to be performed on the state template. Use one of the following values for this parameter:
  - `activate` – The Edge Agent starts sending signal updates to the cloud based on the `stateTemplateUpdateStrategy` you specified (on-change or periodic) when you applied the state template to the vehicle. For more information, see Associate an AWS IoT FleetWise state template with a vehicle.

    Also, you can define an automatic state template deactivation time to stop updates after a specified time period. If an automatic deactivation time is not provided, the state templates will keep sending updates until a deactivate call is issued.

    As soon as the `activate` command is received, the device should send the signals specified in the state template according to the update strategy. AWS IoT FleetWise recommends that when an activate command is received by the device, the first message it sends should contain a snapshot of all the signals in the state template. Subsequent messages should be sent according to the update strategy.
  - `deactivate` – The Edge Agent stops sending signal updates to the cloud.

- `fetchSnapshot` – The Edge Agent sends a onetime snapshot of the signals defined in the state template regardless of the `stateTemplateUpdateStrategy` you specified when you applied the state template to the vehicle.

- (Optional) `$stateTemplate.deactivateAfterSeconds` – The state template is automatically deactivated after the time specified. This parameter can only be used when the value of the `$stateTemplate.operation` parameter is “activate”. If this parameter isn't specified, or if the value of this parameter is 0, the Edge Agent keeps sending signal updates to the cloud until a "deactivate" operation is received for the state template. The state template is never automatically deactivated.

  Minimum value: 0, maximum value: 4294967295.

Note

- The API returns success in response to an activation request for a template already in the active state.
- The API returns success in response to a deactivation request for a template already in the deactivation state.
- The most recent request that you make on a state template is the one that takes effect. For example, if you make a request for a state template to deactivate in one hour, then make a second request for that same template to deactivate in four hours, the four hour deactivation takes effect due to it being the most recent request.

Important

A validation exception can occur in any of the following scenarios:

- A state template is provided which is not `ASSOCIATED` with a vehicle.
- A request is made to activate a state template but it hasn't been `DEPLOYED` on a vehicle.
- A request is made to a state template but it's being `DELETED` on a vehicle.

Example: Activate a state template

To activate a state template, first create a command resource. You can then send the following command to the vehicle on which you want to activate the state template. This example shows how you can specify default values for the parameters when creating a command. These parameters and their values are used when starting the command execution to activate the state template.

1. **Create a command resource**

   Before you can send a command to the vehicle, you must create a command resource. You can specify alternative values for mandatory parameters when you send the command to the vehicle. For more information, see Create a command resource.

   > **Important**
   >
   > `$stateTemplate.name` and `$stateTemplate.operation` parameters must be provided as a string data type. If any other data type is provided, or if either of these two parameters is missing, the command execution fails with a validation exception. The `$stateTemplate.deactivateAfterSeconds` parameter must be provided as a Long data type.

```
aws iot create-command \
    --description "This command activates a state template on a vehicle"
    --command-id ActivateStateTemplate \
    --display-name "Activate State Template" \
    --namespace AWS-IoTFleetWise \
    --mandatory-parameters '[
    {
        "name": "$stateTemplate.name",
        "defaultValue": {"S": "ST123"}
    },
    {
        "name": "$stateTemplate.operation",
        "defaultValue": {"S": "activate"}
    },
    {
        "name": "$stateTemplate.deactivateAfterSeconds",
        "defaultValue": {"L": "120"}
```

```
    }
]'
```

2. **Start the command execution on the vehicle**

   After the command is created, send the command to the vehicle. If you didn't specify values for the mandatory parameters when you created the command resource, you must specify them now. For more information, see Send a command (AWS CLI).

   > **Important**
   >
   > Make sure that you use the account-specific AWS IoT jobs data plane API endpoint for the API operation.

```
aws iot-jobs-data start-command-execution \
    --endpoint-url <endpoint-url> \
    --command-arn arn:aws:iot:region:111122223333:command/ActivateStateTemplate \
    --target-arn arn:aws:iot:region:111122223333:thing/<VEHICLE_NAME>
```

3. **Retrieve the status of the state template operation**

   After you start the command execution, you can use the `GetCommandExecution` API to retrieve the state template.

```
aws iot get-command-execution --execution-id <EXECUTION_ID>
```

Example: Deactivate a state template

To deactivate a state template, first create a command resource. You can then send the following command to the vehicle on which you want to deactivate the state template. This example shows how you can specify default values for the parameters when creating a command. These parameters and their values are used when starting the command execution to deactivate the state template.

1. **Create a command resource**

   Before you can send a command to the vehicle, you must create a command resource. You can specify alternative values for mandatory parameters when you send the command to the vehicle. For more information, see Create a command resource.

```
aws iot create-command \
    --description "This command deactivates a state template on a vehicle"
    --command-id DeactivateStateTemplate \
    --display-name "Deactivate State Template" \
    --namespace AWS-IoTFleetWise \
    --mandatory-parameters '[
    {
        "name": "$stateTemplate.name",
        "defaultValue": {"S": "ST123"}
    },
    {
        "name": "$stateTemplate.operation",
        "defaultValue": {"S": "deactivate"}
    }
]'
```

2. **Start the command execution on the vehicle**

   After the command is created, send the command to the vehicle. If you didn't specify values for the mandatory parameters when you created the command resource, you must specify them now. For more information, see Send a command (AWS CLI).

```
aws iot-jobs-data start-command-execution \
    --endpoint-url <endpoint-url> \
    --command-arn arn:aws:iot:region:111122223333:command/DeactivateStateTemplate \
    --target-arn arn:aws:iot:region:111122223333:thing/<VEHICLE_NAME>
```

3. **Retrieve the status of the state template operation**

   After you start the command execution, you can use the `GetCommandExecution` API to retrieve the state template.

```
aws iot get-command-execution  --execution-id <EXECUTION_ID>
```

Fetch a vehicle state snapshot using state templates

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

You can fetch a vehicle's last known state using the`CreateCommand` AWS IoT Core control plane API operation or the AWS IoT FleetWise console.

> **Important**
>
> A validation exception can occur in any of the following scenarios:
>
> - A state template is provided which is not `ASSOCIATED` with a vehicle.
> - A request is made to activate a state template but it hasn't been `DEPLOYED` on a vehicle.
> - A request is made to a state template but it's being `DELETED` on a vehicle.

Fetch a vehicle state snapshot (console)

You can use the AWS IoT FleetWise console to fetch a vehicle's last known state. AWS IoT FleetWise will create a command for you to fetch data.

To fetch a vehicle's state

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicles**.
3. Choose a vehicle from the list to open its details page.
4. On the **State templates** tab, choose **Fetch data**.
5. Select the IAM role that grants AWS IoT FleetWise permissions to send a command and fetch data. See Controlling access.
6. Choose **Fetch state**.

Fetch a vehicle state snapshot (AWS CLI)

To fetch a state snapshot, first create a command resource. You can then send the following command to the vehicle for which you want to fetch the state snapshot. For more information about using the `CreateCommand` API and its parameters, see Using the `CreateCommand` API.

1. **Create a command resource**

   The following example shows how to create the command resource to perform the fetch operation. You can specify alternative values for mandatory parameters when you send the command to the vehicle. For more information, see Create a command resource.

```
aws iot create-command \
    --command-id <COMMAND_ID> \
    --display-name "FetchSnapshot State Template" \
    --namespace AWS-IoTFleetWise \
    --mandatory-parameters '[
      {
          "name": "$stateTemplate.name",
          "defaultValue": {"S": "ST123"}
      },
      {
          "name": "$stateTemplate.operation",
          "defaultValue": {"S": "fetchSnapshot"}
      }
    ]'
```

   Response:

```
{
    "commandId": "<COMMAND_ID>",
    "commandArn": "arn:aws:iot:<REGION>:111122223333:command/<COMMAND_ID>"
}
```

2. **Start command execution to fetch state snapshot**

   After the command is created, send the command to the vehicle. If you didn't specify values for the mandatory parameters when you created the command resource, you must specify them now. For more information, see Send a command (AWS CLI).

```
aws iot-jobs-data start-command-execution \
```

```
    --command-arn arn:aws:iot:region:111122223333:command/<COMMAND_ID> \
    --target-arn arn:aws:iot:region:111122223333:thing/<VEHICLE_NAME>
```

Response:

```
{
    "executionId": "<UNIQUE_UUID>"
}
```

3. Retrieve the status of the state template operation

   After you start the command execution, you can use the `GetCommandExecution` API to retrieve the state template.

```
aws iot get-command-execution --execution-id <EXECUTION_ID>
```

Process last known state vehicle data using MQTT messaging

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

To receive updates from your vehicle and process its data, subscribe to the following MQTT topic. For more information, see MQTT topics in the *AWS IoT Core Developer Guide.*

```
$aws/iotfleetwise/vehicles/$vehicle_name/last_known_state/$state_template_name/data
```

Last known state signal update messages might be received out of order, as MQTT doesn't guarantee ordering. Any clients which use MQTT to receive and process vehicle data must handle this. Last known state signal update messages follow the MQTT 5 messaging protocol.

The message header for each MQTT message has the following user properties:

- **vehicleName** – A unique identifier of the vehicles.
- **stateTemplateName** – A unique identifier of the last known state state template.

In addition, you can specify vehicle attributes to be included in the MQTT message header by specifying the `metadataExtraDimensions` request parameter while updating or creating a state template. (See State Templates.)

The user properties in the MQTT message header are useful for routing messages to different destinations without inspecting the payload.

The MQTT message payload contains data collected from the vehicles. You can specify vehicle attributes to be included in the MQTT message payload by specifying the `extraDimensions` request parameter while creating or updating a state template (see Create an AWS IoT FleetWise state template). The extra dimensions enrich the data collected from the vehicles by associating extra dimensions to them.

The MQTT message payload is protocol buffers (Protobuf) encoded, and the MQTT message header contains a content type indicator defined as application/octet-stream. The Protobuf encoding schema is as follows:

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

syntax = "proto3";

option java_package = "com.amazonaws.iot.autobahn.schemas.lastknownstate";
package Aws.IoTFleetWise.Schemas.CustomerMessage;

message LastKnownState {

  /*
   * The absolute timestamp in milliseconds since Unix Epoch of when the event was
 triggered in vehicle.
   */
  uint64 time_ms = 1;

  /*
   * This field is deprecated, use signals instead
   */
  repeated Signal signal = 2 [ deprecated = true ];

  repeated Signal signals = 3;

  repeated ExtraDimension extra_dimensions = 4;
}
```

```
message Signal {

  /*
   * The Fully Qualified Name of the signal is the path to the signal plus the signal's
 name.
   * For example, Vehicle.Chassis.SteeringWheel.HandsOff.HandsOffSteeringState
   * The fully qualified name can have up to 150 characters. Valid characters: a-z, A-
Z, 0-9, : (colon), and _ (underscore).
   */
  string name = 1;

  /*
   * The FWE reported signal value can be one of the following data types.
   */
  oneof SignalValue {
    double double_value = 2;

    bool boolean_value = 3;

    sint32 int8_value = 4;

    uint32 uint8_value = 5;

    sint32 int16_value = 6;

    uint32 uint16_value = 7;

    sint32 int32_value = 8;

    uint32 uint32_value = 9;

    sint64 int64_value = 10;

    uint64 uint64_value = 11;

    float float_value = 12;
    /*
     * An UTF-8 encoded or 7-bit ASCII string
     */
    string string_value = 13;
  }
}
```

```
message ExtraDimension {
  /*
   * The Fully Qualified Name of the attribute is the path to the attribute plus the
 attribute's name.
   * For example, Vehicle.Model.Color
   * The fully qualified name can have up to 150 characters. Valid characters: a-z, A-
Z, 0-9, : (colon), and _ (underscore).
   */
  string name = 1;

  oneof ExtraDimensionValue {
    /*
     * An UTF-8 encoded or 7-bit ASCII string
     */
    string string_value = 2;
  }
}
```

Where:

- `time_ms`:

  The absolute timestamp (in milliseconds since the Unix Epoch) of when the event was triggered in the vehicle. The Edge Agent software uses on the vehicle's clock for this timestamp.
- `signal`:

  An array of `Signals` that contain the signal information: `name` (string) and `signalValue` which supports the following data types - `double`, `bool`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, `uint64`, `float`, `string`.
- `extra_dimensions`:

  An array of `ExtraDimensions` that contain vehicle attribute information: `name` (string) and `extraDimensionValue` which currently only supports the `string` data type.

Tutorial: Configure network agnostic data collection using a custom decoding interface

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Introduction

This tutorial outlines how to configure AWS IoT FleetWise to collect data and run commands using network agnostic data collection, which utilizes a custom decoding interface. With network agnostic data collection, you can use your own methods to decode signals before sending them to your specified data destination. This saves time since you don't need to create signal decoders specifically for AWS IoT FleetWise. You can have a subset of signals decoded using your own implementation, or you can use `defaultForUnmappedSignals` when you create or update a decoder manifest. This also provides flexibility to collect signals and triggers across a wide range of sources within the vehicle.

This tutorial is intended for vehicle signals that are not on a standard Controller Area Network (CAN bus) interface. For example, data encoded in a custom in-vehicle format or scheme.

Environment setup

This tutorial assumes you have gone through the steps to set up your environments to access the AWS IoT FleetWise cloud, and the Edge implementation APIs and code base.

Data models

The next section illustrates how to model vehicle properties using a custom decoding interface. This applies to data collection as well as command use cases. It also applies to any underlying data source modeling used in the vehicle, for example, IDLs.

In the example, there are two vehicle properties: a vehicle sensor (current vehicle position) to collect and a vehicle actuator (Air Conditioner) to control remotely. Both of those are defined in this scheme:

```
// Vehicle WGS84 Coordinates
double Latitude;
double Longitude;

// Vehicle AC
Boolean ActivateAC;
```

The next step is to import these definitions into AWS IoT FleetWise using the custom decoding interface APIs.

Signal catalog updates

Import these definitions in your signal catalog. If you have a signal catalog in AWS IoT FleetWise already, use the update API directly. If you don't have one, first create a signal catalog and then call the update API.

First, you must create the VSS representation of these vehicle signals. VSS is used as a Taxonomy to represent vehicle data in AWS IoT FleetWise. Create a json file called 'vehicle-signals.json' with these contents:

```
// vehicle-signals.json
// Verify that branches and nodes are unique in terms of fully qualified name
// in the signal catalog.
[
 {
    "branch": {
      "fullyQualifiedName": "Vehicle",
      "description": "Vehicle Branch"
    }
  },
  {
    "branch": {
      "fullyQualifiedName": "Vehicle.CurrentLocation",
      "description": "CurrentLocation"
    }
  },
  {
    "sensor": {
      "dataType": "DOUBLE",
      "fullyQualifiedName": "Vehicle.CurrentLocation.Latitude",
      "description": "Latitude"
    }
```

```
  },
  {
    "sensor": {
      "dataType": "DOUBLE",
      "fullyQualifiedName": "Vehicle.CurrentLocation.Longitude",
      "description": "Longitude"
    }
  },
  {
    "actuator": {
      "fullyQualifiedName": "Vehicle.ActivateAC",
      "description": "AC Controller",
      "dataType": "BOOLEAN"
    }
  }
]
```

If you don't have a signal catalog in place, then you need to invoke `create-signal-catalog`:

```
VEHICLE_NODES=`cat vehicle-signals.json`
aws iotfleetwise create-signal-catalog \
        --name my-signal-catalog \
        --nodes "${VEHICLE_NODES}"
```

If you have a signal catalog already, you can add those signals using the `update-signal-catalog` API:

```
VEHICLE_NODES=`cat vehicle-signals.json`
aws iotfleetwise update-signal-catalog \
        --name my-signal-catalog \
        --nodes-to-add "${VEHICLE_NODES}"
```

Vehicle model and decoder

After you insert the signals in the signal catalog, the next step is to create a vehicle model and instantiate those signals. For that, you use the `create-model-manifest` and `create-decoder-manifest` APIs.

First, format the signal names that you want to insert into the vehicle model:

```
# Prepare the signals for insertion into the vehicle model.
VEHICLE_NODES=`cat vehicle-signals.json`
```

```
VEHICLE_NODES=`echo ${VEHICLE_NODES} | jq -r ".[] | .actuator,.sensor
 | .fullyQualifiedName" | grep Vehicle\\.`
VEHICLE_NODES=`echo "${VEHICLE_NODES}" | jq -Rn [inputs]`
# This is how the vehicle model input looks.
echo $VEHICLE_NODES
# [ "Vehicle.CurrentLocation.Latitude",
#   "Vehicle.CurrentLocation.Longitude",
#   "Vehicle.ActivateAC" ]
# Create the vehicle model with those signals.
aws iotfleetwise create-model-manifest \
    --name my-model-manifest \
    --signal-catalog-arn arn:xxxx:signal-catalog/my-signal-catalog \
    --nodes "${VEHICLE_NODES}"

# Activate the vehicle model.
 aws iotfleetwise update-model-manifest \
    --name my-model-manifest --status ACTIVE
```

Now, use the custom decoding interface to create a decoder manifest.

> **Note**
>
> You only need to create network interfaces and signals if you want to specify custom IDs, which isn't part of this example.
> For information about mapping decoding information when the fully qualified name (FQN) differs from the custom decoding signal ID, see the *Edge Agent Developer Guide*.

```
// Create a network interface that is of type : CUSTOM_DECODING_INTERFACE
// custom-interface.json
[
  {
    "interfaceId": "NAMED_SIGNAL",
    "type": "CUSTOM_DECODING_INTERFACE",
    "customDecodingInterface": {
      "name": "NamedSignalInterface"
    }
  },
  {
    "interfaceId": "AC_ACTUATORS",
    "type": "CUSTOM_DECODING_INTERFACE",
    "customDecodingInterface": {
```

```
      "name": "NamedSignalInterface"
    }
  }
]
// custom-decoders.json
// Refer to the fully qualified names of the signals, make them of
// type CUSTOM_DECODING_SIGNAL, and specify them as part of the same interface ID
// that was defined above.
[
    {
      "fullyQualifiedName": "Vehicle.CurrentLocation.Longitude",
      "interfaceId": "NAMED_SIGNAL",
      "type": "CUSTOM_DECODING_SIGNAL",
      "customDecodingSignal": {
        "id": "Vehicle.CurrentLocation.Longitude"
      }
    },
    {
      "fullyQualifiedName": "Vehicle.CurrentLocation.Latitude",
      "interfaceId": "NAMED_SIGNAL",
      "type": "CUSTOM_DECODING_SIGNAL",
      "customDecodingSignal": {
        "id": "Vehicle.CurrentLocation.Latitude"
      }
    },
    {
        "fullyQualifiedName": "Vehicle.ActivateAC",
        "interfaceId": "AC_ACTUATORS",
        "type": "CUSTOM_DECODING_SIGNAL",
        "customDecodingSignal": {
          "id": "Vehicle.ActivateAC"
        }
    }
]
# Create the decoder manifest.
 CUSTOM_INTERFACE=`cat custom-interface.json`
 CUSTOM_DECODERS=`cat custom-decoders.json`

aws iotfleetwise create-decoder-manifest \
   --name my-decoder-manifest \
   --model-manifest-arn arn:xxx:model-manifest/my-model-manifest \
   --network-interfaces "${CUSTOM_INTERFACE}" \
   --signal-decoders "${CUSTOM_DECODERS}"
```

```
 # Activate the decoder manifest.
 aws iotfleetwise update-decoder-manifest \
    --name my-decoder-manifest \
    --status ACTIVE
```

At this point, you have fully modeled these signals in AWS IoT FleetWise. Next you create the vehicle and associate it with the model you created. You use the `create-vehicle` API for that:

```
aws iotfleetwise create-vehicle \
    --decoder-manifest-arn arn:xxx:decoder-manifest/my-decoder-manifest \
    --association-behavior ValidateIotThingExists \
    --model-manifest-arn arn:xxx:model-manifest/my-model-manifest \
    --vehicle-name "my-vehicle"
```

The next step is to focus on the AWS IoT FleetWise Edge code base and write the necessary code extension.

> **Note**
>
> For information about the Edge implementation, see the *Edge Agent Developer Guide*.

Send command

Now, compile the software (make sure you add your headers and C++ files to the CMake file), and then go back to the cloud APIs to test a command on this actuator:

```
// Create a command targeting your vehicle.
aws iot create-command --command-id activateAC \
    --namespace "AWS-IoT-Fleetwise" \
    --endpoint-url endpoint-url \
    --role-arn ${SERVICE_ROLE_ARN} \
    --mandatory-parameters '[ { "name": "$actuatorPath.Vehicle.ActivateAC",
 "defaultValue": {"B": "false"} } ]' \
// You will receive the command ARN.

{
    "commandId": "activateAC",
    "commandArn": "arn:aws:iot:xxx:command/activateAC"
}
```

```
// You can send the command to activate the AC targeting your vehicle.

JOBS_ENDPOINT_URL=`aws iot describe-endpoint --endpoint-type iot:Jobs | jq -
j .endpointAddress`
aws iot-jobs-data start-command-execution \
    --command-arn arn:aws:iot:xxx:command/activateAC \
    --target-arn arn:xxx:vehicle/my-vehicle \
    --parameters '{ "$actuatorPath.Vehicle.ActivateAC" : {"B": "true"}}' \
    --endpoint-url https://${JOBS_ENDPOINT_URL}
// You will receive the corresponding execution ID.
{
    "executionId": "01HSK4ZH6ME7D43RB2BV8JC51D"
}

// If you have the AWS IoT FleetWise Edge Agent running, you can see the logs.
[AcCommandDispatcher.cpp:26] [setActuatorValue()]:
[Actuator Vehicle.ActivateAC executed successfully for command ID
 01HSK4ZH6ME7D43RB2BV8JC51D]
```

Use AWS CLI and AWS SDKs with AWS IoT FleetWise

This section provides information about making AWS IoT FleetWise API requests. For more information about AWS IoT FleetWise operations and data types, see the *AWS IoT FleetWise API Reference*.

To use AWS IoT FleetWise with a variety of programming languages, use the AWS SDKs, which contain the following automatic functionality:

- Cryptographically signing your service requests
- Retrying requests
- Handling error responses

For command line access, use AWS IoT FleetWise with the AWS CLI. You can control AWS IoT FleetWise, and your other services, from the command line, and automate them through scripts.

Troubleshooting AWS IoT FleetWise

Use the troubleshooting information and solutions in this section to help resolve issues with AWS IoT FleetWise.

The following information might help you troubleshoot common issues with AWS IoT FleetWise.

Topics

AWS IoT FleetWise decoder manifest issues

Troubleshoot decoder manifest issues.

Diagnosing decoder manifest API calls

Error	Troubleshooting guidelines
`UpdateOperationFailure.ConflictingDecoderUpdate`	The same decoder manifest has multiple update requests. Wait and try again.
`UpdateOperationFailure.InternalFailure`	InternalFailure is launched as an encapsulated exception. The problem itself depends on the exception encapsulated.
`UpdateOperationFailure.ActiveDecoderUpdate`	The decoder manifest is in an `Active` state and can't be updated. Change the decoder manifest state to `DRAFT`, and then try again.
`UpdateOperationFailure.ConflictingModelUpdate`	AWS IoT FleetWise is trying to validate against a vehicle model (model manifest) that's being modified by someone else. Wait and try again.
`UpdateOperationFailure.ModelManifestValidationResponse :`	The vehicle model doesn't have any signals associated with it. Add signals to the vehicle

Error	Troubleshooting guidelines
`FailureReason.MODEL_DATA_ENTRIES_NOT_FOUND`	model and verify that the signals can be found in the associated signal catalog.
`UpdateOperationFailure.ModelManifestValidationResponse : FailureReason.MODEL_NOT_ACTIVE`	Update the vehicle model so that it's in `ACTIVE` state, and then try again.
`UpdateOperationFailure.ModelManifestValidationResponse : FailureReason.MODEL_NOT_FOUND`	AWS IoT FleetWise can't find the vehicle model associated with the decoder manifest. Verify the Amazon Resource Name (ARN) of the vehicle model and try again.
`UpdateOperationFailure.ModelManifestValidationResponse (FailureReason.MODEL_DATA_ENTRIES_READ_FAILURE`	The validation of the vehicle model failed because signal names from the vehicle model weren't found in the signal catalog. Verify that the signals in the vehicle model are all included in the associated signal catalog.
`UpdateOperationFailure.ValidationFailure`	Signals or network interfaces that aren't valid were found in the request to update the decoder manifest. Verify that all signals and network interfaces returned by the exception exist, that all signals used are associated with an available interface, and that you won't remove an interface that has signals associated with it.
`UpdateOperationFailure.KmsKeyAccessDenied`	There's a permission issue on the AWS Key Management Service (AWS KMS) key used for the operation. Verify that you're using a role that has access to the key and try again.
`UpdateOperationFailure.DecoderDoesNotExist`	The decoder manifest doesn't exist. Verify the decoder manifest name and try again.

Vision system data error messages with the `SIGNAL_DECODER_INCOMPATIBLE_WITH_SIGNAL_CATALOG` reason will include a hint in the response that provides information about why the request failed. You can use the hint to determine which troubleshooting guidelines to follow.

> **Note**
>
> Vision system data is in preview release and is subject to change.

Diagnosing decoder manifest vision system data validation

Error	Troubleshooting guidelines
`InvalidSignalDecoder.withReason(SignalDecoderFailureReason.NO_SIGNAL_IN_CATALOG_FOR_DECODER_SIGNAL)`	AWS IoT FleetWise didn't find the root signal structure used in the signal decoder using the signal catalog. Verify that the root signal of the structure is properly defined in the signal catalog.
`InvalidSignalDecoder.withReason(SignalDecoderFailureReason.SIGNAL_DECODER_TYPE_INCOMPATIBLE_WITH_MESSAGE_SIGNAL_TYPE)`	A primitive message in the signal catalog wasn't defined with the same data type in the decoder manifest update request. Verify that the primitive messages defined in the request match their corresponding signal catalog definition.
`InvalidSignalDecoder.withReason(SignalDecoderFailureReason.STRUCT_SIZE_MISMATCH)`	The number of properties defined in a struct in the signal catalog don't match the number of properties you're trying to decode in the decoder manifest. Verify that you have the correct number of signals to decode by comparing it with the signals defined in the signal catalog.
`InvalidSignalDecoder.withReason(SignalDecoderFailureRe`	AWS IoT FleetWise found a signal defined as a STRUCT in the signal catalog without a structuredMessageDefinition defined in the decoder manifest request. Make sure that each

Error	Troubleshooting guidelines
ason.SIGNAL_DECODER_INCOMPA TIBLE_WITH_SIGNAL_CATALOG)	struct is defined as a structuredMessageD efinition in the decoder manifest update request.
InvalidSignalDecoder.withRe ason(SignalDecoderFailureRe ason.SIGNAL_DECODER_INCOMPA TIBLE_WITH_SIGNAL_CATALOG)	The root signal of the structure used in the decoder manifest is not properly defined as a structure in the signal catalog. The root signal structure used in the decoder manifest must have its field structFullyQualifiedName defined. It also needs a STRUCT node with that fullyQualifiedName.
InvalidSignalDecoder.withRe ason(SignalDecoderFailureRe ason.SIGNAL_DECODER_INCOMPA TIBLE_WITH_SIGNAL_CATALOG)	One of the leaf messages used in the decoder manifest request is not defined as a primitive message. Verify that all leaf objects in the request are defined as primitive messages.
InvalidSignalDecoder.withRe ason(SignalDecoderFailureRe ason.SIGNAL_DECODER_INCOMPA TIBLE_WITH_SIGNAL_CATALOG)	An array object in the signal catalog wasn't defined as a structuredMessageListDefini tion in the decoder manifest update request. Verify that all array properties are defined as structuredMessageListDefinition in the decoder manifest update request.

Edge Agent for AWS IoT FleetWise software issues

Troubleshoot Edge Agent software issues.

Issues

- Issue: The Edge Agent software doesn't start.
- Issue: [ERROR] [IoTFleetWiseEngine::connect]: [Failed to init persistency library]
- Issue: The Edge Agent software doesn't collect on-board diagnostics (OBD) II PIDs and diagnostic trouble codes (DTCs).

- Issue: The Edge Agent for AWS IoT FleetWise software doesn't collect data from the network or isn't able to apply data inspection rules.
- Issue: [ERROR] [AwsIotConnectivityModule::connect]: [Connection failed with error] or [WARN] [AwsIotChannel::send]: [No alive MQTT Connection.]

Issue: The Edge Agent software doesn't start.

You might see the following errors when the Edge Agent software doesn't start.

- ```
  Error from reader: * Line 1, Column 1
  Syntax error: value, object or array expected.
  ```

  Solution: Make sure the Edge Agent for AWS IoT FleetWise software configuration file is using valid JSON format. For example, make sure that commas are used correctly. For more information about the configuration file, do the following to download the *Edge Agent for AWS IoT FleetWise software Developer Guide*.

  1. Open the AWS IoT FleetWise console.
  2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.

- ```
  [ERROR] [SocketCANBusChannel::connect]: [ SocketCan with name xxx is not accessible]
  [ERROR] [IoTFleetWiseEngine::connect]: [ Failed to Bind Consumers to Producers ]
  ```

  Solution: You might see this error when the Edge Agent software fails to establish socket communication with the network interfaces defined in the configuration file.

  To check that every network interface defined in the configuration is available, run the following command.

  ```
  ip link show
  ```

  To bring a network interface online, run the following command. Replace *network-interface-id* with the ID of the network interface.

  ```
  sudo ip link set network-interface-id up
  ```

- ```
  [ERROR] [AwsIotConnectivityModule::connect]: [Connection failed with error]
  [WARN] [AwsIotChannel::send]: [No alive MQTT Connection.]
  # or
  [WARN] [AwsIotChannel::send]: [aws-c-common: AWS_ERROR_FILE_INVALID_PATH]
  ```

  Solution: You might see this error when the Edge Agent software fails to establish an MQTT connection to AWS IoT Core. Check that the following are configured correctly and restart the Edge Agent software.

  - `mqttConnection::endpointUrl` – AWS account's IoT device endpoint.
  - `mqttConnection::clientID` – The ID of the vehicle in which the Edge Agent software is running.
  - `mqttConnection::certificateFilename` – The path to the vehicle certificate file.
  - `mqttConnection::privateKeyFilename` – The path to the vehicle private key file.
  - You have used AWS IoT Core to provision the vehicle. For more information, see Provision AWS IoT FleetWise vehicles.

  For more troubleshooting information, see AWS IoT Device SDK for C++ Frequently Asked Questions.

Issue: [ERROR] [IoTFleetWiseEngine::connect]: [Failed to init persistency library]

Solution: You might see this error when the Edge Agent software fails to locate the persistence storage. Check that the following is configured correctly and restart the Edge Agent software.

`persistency:persistencyPath` – A local path used to persist collection schemes, decoder manifests, and data snapshots.

Issue: The Edge Agent software doesn't collect on-board diagnostics (OBD) II PIDs and diagnostic trouble codes (DTCs).

Solution: You might see this error if `obdInterface:pidRequestIntervalSeconds` or `obdInterface:dtcRequestIntervalSeconds` is configured to 0.

If the Edge Agent software is running in an automatic transmission vehicle, make sure `obdInterface:hasTransmissionEcu` is configured to `true`.
  ```

If your vehicle supports extended Controller Area Network (CAN bus) arbitration IDs, make sure `obdInterface:useExtendedIds` is configured to `true`.

Issue: The Edge Agent for AWS IoT FleetWise software doesn't collect data from the network or isn't able to apply data inspection rules.

Solution: You might see this error when the default quotas are breached.

Resource	Quota	Adjustable	Note
Value of the signal ID	The signal ID must be less than or equal to 50,000	Yes	The Edge Agent software won't collect data from signals that have an ID greater than 50,000. We recommend that you check how many signals the signal catalog contains before you change this quota.
Number of active data collection schemes per vehicle	256	Yes	We recommend that you check how many campaigns that you've created in the cloud and how many schemes each campaign contains before you change this quota.
Size of the signal history buffer	20 MB	Yes	If the quota is breached, the Edge Agent software stops collecting new data.

Issue: [ERROR] [AwsIotConnectivityModule::connect]: [Connection failed with error] or [WARN] [AwsIotChannel::send]: [No alive MQTT Connection.]

Solution: You might see this error when the Edge Agent software isn't connected to the cloud. By default, the Edge Agent software sends a ping request to AWS IoT Core every minute and waits for three minutes. If there's no response, the Edge Agent software automatically reestablishes the connection to the cloud.

Store and forward issues

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Issue: Receiving an `AccessDeniedException` with all required IAM permissions

Solution: The store and forward feature for data partitioning in campaigns requires gated access through allow listing. Contact the service team to ensure that your resources have adequate permissions through allow listing.

Issue: The data uploaded to AWS IoT Jobs ignores the `endTime`

Solution: You have specified an invalid `endtime` in the job document. For example, the `endtime` doesn't following ISO 8601 UTC format). On AWS IoT FleetWise Agent logs, there could be a warning-level statement that says, `Malformed IoT Job endTime: customer configured endTime. Not setting endTime.`

Issue: The data upload to AWS IoT Jobs has a `REJECTED` execution status.

Solution: You have specified an invalid `campaignArn` in the job document. For example, if you specify an ARN for a campaign that is not running on a vehicle, there could be a error-level

statement that says, `CampaignArn value in the received job document does not match the ARN of a Store and Forward campaign` in the AWS IoT FleetWise Agent logs.

Security in AWS IoT FleetWise

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS IoT FleetWise, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using AWS IoT FleetWise. It shows you how to configure AWS IoT FleetWise to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS IoT FleetWise resources.

Contents

Data protection in AWS IoT FleetWise

The AWS shared responsibility model applies to data protection in AWS IoT FleetWise. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS IoT FleetWise or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

AWS IoT FleetWise is intended to be used with an Edge Agent that you develop and install on supported vehicle hardware in order to transmit vehicle data to the AWS Cloud. Extracting data

from vehicles *might* be subject to data privacy regulations in certain jurisdictions. Before using AWS IoT FleetWise and installing your Edge Agent, we strongly recommend that you assess your compliance obligations under applicable law. This includes any applicable legal requirements to provide legally adequate privacy notices and obtain any necessary consents for extracting vehicle data.

Encryption at rest in AWS IoT FleetWise

The data collected from a vehicle is transmitted to the cloud through an AWS IoT Core message with the MQTT message protocol. AWS IoT FleetWise delivers the data to your Amazon Timestream database. In Timestream, your data is encrypted. All AWS services encrypt data at rest by default. For more information, see Protecting data with encryption in the Amazon S3 User Guide and Data protection in Timestream for LiveAnalytics.

Encryption at rest integrates with AWS Key Management Service (AWS KMS) to manage the encryption key that's used to encrypt your data. You can choose to use a customer managed key to encrypt data collected by AWS IoT FleetWise. You can create, manage, and view your encryption key through AWS KMS. For more information, see What is AWS Key Management Service? in the *AWS Key Management Service Developer Guide*.

Encryption in transit

All data exchanged with AWS IoT services is encrypted in transit by using Transport Layer Security (TLS). For more information, see Transport security in the *AWS IoT Developer Guide*.

Also, AWS IoT Core supports authentication and authorization to help securely control access to AWS IoT FleetWise resources. Vehicles can use X.509 certificates to get authenticated (signed in) to use AWS IoT FleetWise and use AWS IoT Core policies to get authorized (have permissions) to perform specified actions. For more information, see the section called “Provision vehicles”.

Data encryption in AWS IoT FleetWise

Data encryption refers to protecting data while in-transit (as it travels to and from AWS IoT FleetWise, and between gateways and servers), and at rest (while it's stored on local devices or in AWS services). You can protect data at rest using client-side encryption.

> **Note**
>
> AWS IoT FleetWise edge processing exposes APIs that are hosted within AWS IoT FleetWise gateways and are accessible over the local network. These APIs are exposed over a TLS

connection backed by a server-certificate owned by the AWS IoT FleetWise Edge connector. For client authentication, these APIs use an access-control password. The server-certificate private-key and the access-control password are both stored on disk. AWS IoT FleetWise edge processing relies on file-system encryption for the security of these credentials at rest.

For more information about server-side encryption and client-side encryption, review the following topics.

Contents

Encryption at rest in AWS IoT FleetWise

AWS IoT FleetWise stores your data in the AWS Cloud and on gateways.

Data at rest in the AWS Cloud

AWS IoT FleetWise stores data in other AWS services that encrypt data at rest by default. Encryption at rest integrates with AWS Key Management Service (AWS KMS) for managing the encryption key that is used to encrypt your asset property values and aggregate values in AWS IoT FleetWise. You can choose to use a customer managed key to encrypt asset property values and aggregate values in AWS IoT FleetWise. You can create, manage, and view your encryption key through AWS KMS.

You can choose an AWS owned key or a customer managed key to encrypt your data.

How it works

Encryption at rest integrates with AWS KMS for managing the encryption key that is used to encrypt your data.

- AWS owned key – Default encryption key. AWS IoT FleetWise owns this key. You can't view, manage, or use this key in your AWS account. You also can't see operations on the key in AWS CloudTrail logs. You can use this key at no additional charge.
- Customer managed key – The key is stored in your account, which you create, own, and manage. You have full control over the KMS key. Additional AWS KMS charges apply.

AWS owned keys

AWS owned keys aren't stored in your account. They're part of a collection of KMS keys that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned keys to protect your data.

You can't view, manage, or use AWS owned keys, or audit their use. However, you don't need to take any action or change any programs to protect keys that encrypt your data.

You won't be charged a fee if you use AWS owned keys, and they don't count against AWS KMS quotas for your account.

Customer managed keys

Customer managed keys are KMS keys in your account that you create, own, and manage. You have full control over these KMS keys, such as the following:

- Establishing and maintaining their key policies, IAM policies, and grants
- Enabling and disabling them
- Rotating their cryptographic material
- Adding tags
- Creating aliases that refer to them
- Scheduling them for deletion

You can also use CloudTrail and Amazon CloudWatch Logs to track the requests that AWS IoT FleetWise sends to AWS KMS on your behalf.

If you're using customer managed keys, you must grant AWS IoT FleetWise access to the KMS key stored in your account. AWS IoT FleetWise uses envelope encryption and key hierarchy to encrypt data. Your AWS KMS encryption key is used to encrypt the root key of this key hierarchy. For more information, see Envelope encryption in the *AWS Key Management Service Developer Guide*.

The following example policy grants AWS IoT FleetWise permissions to use your AWS KMS key.

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotfleetwise.amazonaws.com"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
  ]
}
```

Important

When you add the new sections to your KMS key policy, don't change any existing sections in the policy. AWS IoT FleetWise can’t perform operations to your data if encryption is enabled for AWS IoT FleetWise and any of the following is true:

- The KMS key is disabled or deleted.
- The KMS key policy isn't correctly configured for the service.

Using vision system data with encryption at rest

Note

Vision system data is in preview release and is subject to change.

If you have customer managed encryption with AWS KMS keys enabled on your AWS IoT FleetWise account, and you want to use vision system data, reset your encryption settings to be compatible with complex data types. This enables AWS IoT FleetWise to establish additional permissions needed for vision system data.

> **Note**
>
> Your decoder manifest could be stuck in a validating status if you haven't reset your encryption settings for vision system data.

1. Use the GetEncryptionConfiguration API operation to check if AWS KMS encryption is enabled. No further action is needed if the encryption type is `FLEETWISE_DEFAULT_ENCRYPTION`.
2. If the encryption type is `KMS_BASED_ENCRYPTION`, use the PutEncryptionConfiguration API operation to reset the encryption type to `FLEETWISE_DEFAULT_ENCRYPTION`.

```
aws iotfleetwise put-encryption-configuration \
      --encryption-type FLEETWISE_DEFAULT_ENCRYPTION
```

3. Use the PutEncryptionConfiguration API operation to re-enable the encryption type to `KMS_BASED_ENCRYPTION`.

```
aws iotfleetwise put-encryption-configuration \
        --encryption-type KMS_BASED_ENCRYPTION \
        --kms-key-id kms_key_id
```

For more information about enabling encryption, see Key management in AWS IoT FleetWise.

Key management in AWS IoT FleetWise

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

AWS IoT FleetWise cloud key management

By default, AWS IoT FleetWise uses AWS managed keys to protect your data in the AWS Cloud. You can update your settings to use a customer managed key to encrypt data in AWS IoT FleetWise. You can create, manage, and view your encryption key through AWS Key Management Service (AWS KMS).

AWS IoT FleetWise supports server-side encryption with customer managed keys stored in AWS KMS to encrypt data for the following resources.

AWS IoT FleetWise resource	Data type	Fields that are encrypted at rest with customer managed keys
Signal catalog		description
	Attribute	description, allowedValues, defaultValue, min, max
	Actuator	description, allowedValues, min, max
	Sensor	description, allowedValues, min, max
Vehicle model (model manifest)		description
Decoder manifest		description
	CanInterface	protocolName, protocolVersion
	ObdInterface	requestMessageId, dtcRequestIntervalSeconds, hasTransmissionEcu, obdStandard, pidRequestIntervalSeconds, useExtendedIds
	CanSignal	factor, isBigEndian, isSigned, length, messageId, offset, startBit
	ObdSignal	byteLength, offset, pid, pidResponseLength, scaling, serviceMode, startByte, bitMaskLength, bitRightShift
Vehicle		attributes
Campaign		description

AWS IoT FleetWise resource	Data type	Fields that are encrypted at rest with customer managed keys
	conditionBasedCollectionScheme	expression, conditionLanguageVersion, minimumTriggerIntervalMs, triggerMode
	TimeBasedCollectionScheme	periodMs
State template		description

> **Note**
>
> Other data and resources are encrypted using the default encryption with keys managed by AWS IoT FleetWise. This key is created and stored in the AWS IoT FleetWise account.

For more information, see What is AWS Key Management Service? in the *AWS Key Management Service Developer Guide*.

Enable encryption using KMS keys (console)

To use customer managed keys with AWS IoT FleetWise, you must update your AWS IoT FleetWise settings.

To enable encryption using KMS keys (console)

1. Open the AWS IoT FleetWise console.
2. Navigate to **Settings**.
3. In **Encryption**, choose **Edit** to open the **Edit encryption** page.
4. For **Encryption key type**, choose **Choose a different AWS KMS key**. This enables encryption with customer managed keys stored in AWS KMS.

> **Note**
>
> You can only use customer managed key encryption for AWS IoT FleetWise resources. This includes the signal catalog, vehicle model (model manifest), decoder manifest, vehicle, fleet, and campaign.

5. Choose your KMS key with one of the following options:

   - **To use an existing KMS key** – Choose your KMS key alias from the list.
   - **To create a new KMS key** – Choose **Create an AWS KMS key**.

   > **Note**
   >
   > This opens the AWS KMS console. For more information about creating a KMS key, see Creating keys in the *AWS Key Management Service Developer Guide*.

6. Choose **Save** to update your settings.

Enable encryption using KMS keys (AWS CLI)

You can use the PutEncryptionConfiguration API operation to enable encryption for your AWS IoT FleetWise account. The following example uses AWS CLI.

To enable encryption, run the following command.

- Replace *kms_key_id* with the ID of the KMS key.

```
aws iotfleetwise put-encryption-configuration \
      --encryption-type KMS_BASED_ENCRYPTION \
      --kms-key-id kms_key_id
```

Example response

```
{
 "kmsKeyId": "customer_kms_key_id",
 "encryptionStatus": "PENDING",
 "encryptionType": "KMS_BASED_ENCRYPTION"
}
```

KMS key policy

After you create a KMS key, you must, at minimum, add the following statement to your KMS key policy for it to work with AWS IoT FleetWise. The AWS IoT FleetWise service principal `iotfleetwise.amazonaws.com` in the KMS key policy statement allows AWS IoT FleetWise to access the KMS key.

```
{
  "Sid": "Allow FleetWise to encrypt and decrypt data when customer managed KMS key
 based encryption is enabled",
  "Effect": "Allow",
  "Principal": {
    "Service": "iotfleetwise.amazonaws.com"
  },
  "Action": [
    "kms:GenerateDataKey*",
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:DescribeKey",
    "kms:CreateGrant",
    "kms:RetireGrant",
    "kms:RevokeGrant"
  ],
  "Resource": "*"
}
```

As a security best practice, add `aws:SourceArn` and `aws:SourceAccount` condition keys to the KMS key policy. The IAM global condition key `aws:SourceArn` helps ensure that AWS IoT FleetWise uses the KMS key only for service-specific resource Amazon Resource Names (ARNs).

If you set the value of `aws:SourceArn`, it must always be `arn:aws:iotfleetwise:us-east-1:account_id:*`. This allows the KMS key to access all AWS IoT FleetWise resources for this AWS account. AWS IoT FleetWise supports one KMS key per account for all resources in that AWS Region. Using any other value for the `SourceArn`, or not using the wildcard (*) for the ARN resource field, prevents AWS IoT FleetWise from accessing the KMS key.

The value of `aws:SourceAccount` is your account ID, which is used to further restrict the KMS key so that it can only be used for your specific account. If you add `aws:SourceAccount` and `aws:SourceArn` condition keys to the KMS key, make sure the key is not used by any other service or account. This helps avoid failures.

The following policy includes a service principal (an identifier for a service), as well as `aws:SourceAccount` and `aws:SourceArn` set up for use based on the AWS Region and your account ID.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {
    "Service": "iotfleetwise.amazonaws.com"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:SourceAccount": "AWS-account-ID"
    },
    "ArnLike": {
      "aws:SourceArn": "arn:aws:iotfleetwise:region:AWS-account-ID:*"
    }
  }
}
```

For more information about editing a KMS key policy for use with AWS IoT FleetWise, see Changing a key policy in the *AWS Key Management Service Developer Guide*.

Important

When you add the new sections to your KMS key policy, don't change any existing sections in the policy. AWS IoT FleetWise can’t perform operations to your data if encryption is enabled for AWS IoT FleetWise and any of the following is true:

- The KMS key is disabled or deleted.
- The KMS key policy isn't correctly configured for the service.

Permissions for AWS KMS encryption

If you enabled AWS KMS encryption, you must specify permissions in the role policy so that you can call AWS IoT FleetWise APIs. The following policy allows access to all AWS IoT FleetWise actions, as well as AWS KMS specific permissions.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotfleetwise:*",
        "kms:GenerateDataKey*",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:DescribeKey"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The following policy statement is required for your role to invoke encryption APIs. This policy statement allows `PutEncryptionConfiguration` and `GetEncryptionConfiguration` actions from AWS IoT FleetWise.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "iotfleetwise:GetEncryptionConfiguration",
        "iotfleetwise:PutEncryptionConfiguration",
        "kms:GenerateDataKey*",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:DescribeKey"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Recovery after AWS KMS key deletion

If you delete an AWS KMS key after enabling encryption with AWS IoT FleetWise, you must reset your account by deleting all data before using AWS IoT FleetWise again. You can use the list and delete API operations to clean up resources in your account.

To clean up resources in your account

1. Use list APIs with the `listResponseScope` parameter set to `METADATA_ONLY`. This provides a list of resources, including resource names and other metadata such as ARNs and timestamps.
2. Use delete APIs to remove individual resources.

You must clean up resources in the following order.

1. Campaigns
   a. List all campaigns with the `listResponseScope` parameter set to `METADATA_ONLY`.
   b. Delete the campaigns.
2. Fleets and vehicles
   a. List all fleets with the `listResponseScope` parameter set to `METADATA_ONLY`.
   b. List all vehicles for each fleet with the `listResponseScope` parameter set to `METADATA_ONLY`.

c. Disassociate all vehicles from each fleet.

d. Delete the fleets.

e. Delete the vehicles.

3. Decoder manifests

   a. List all decoder manifests with the `listResponseScope` parameter set to `METADATA_ONLY`.

   b. Delete all decoder manifests.

4. Vehicle models (model manifests)

   a. List all vehicle models with the `listResponseScope` parameter set to `METADATA_ONLY`.

   b. Delete all vehicle models.

5. State templates

   a. List all state templates with the `listResponseScope` parameter set to `METADATA_ONLY`.

   b. Delete all state templates.

6. Signal catalogs

   a. List all signal catalogs.

   b. Delete all signal catalogs.

Controlling access with AWS IoT FleetWise

Important

Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

The following sections cover how to control access to and from your AWS IoT FleetWise resources. The information they cover includes how to grant your application access so AWS IoT FleetWise can transfer vehicle data during campaigns. They also describe how you can grant AWS IoT FleetWise access to your Amazon S3 (S3) bucket or Amazon Timestream database and table to store data, or to MQTT messages used to send data from vehicles.

The technology for managing all these forms of access is AWS Identity and Access Management (IAM). For more information about IAM, see What is IAM?.

Contents

Grant AWS IoT FleetWise permission to send and receive data on an MQTT topic

When you use an MQTT topic, your vehicles send data using the AWS IoT MQTT message broker. You must grant AWS IoT FleetWise permission to subscribe to the MQTT topic you specify. If you also use AWS IoT Rules to take action, or route data to other destinations, you must attach policies to an IAM role to allow AWS IoT FleetWise to forward data to IoT Rules.

In addition, your other apps or devices can subscribe to the topic you specify to receive vehicle data in near real-time, and these apps or devices must be granted permissions and access as needed.

For more information about using MQTT and the roles and permissions required, see:

- Device communication protocols
- Rules for AWS IoT
- Granting an AWS IoT rule the access it requires
- Pass role permissions

Before you start, check the following:

Important

- You must use the same AWS Region when you create vehicle campaign resources for AWS IoT FleetWise. If you switch AWS Regions, you might have issues accessing the resources.
- AWS IoT FleetWise is available in US East (N. Virginia) and Europe (Frankfurt).

You can use the AWS CLI to create an IAM role with a trust policy for MQTT messaging. To create an IAM role, run the following command.

To create an IAM role with a trust policy

- Replace *IotTopicExecutionRole* with the name of the role you're creating.
- Replace *trust-policy* with the JSON file that contains the trust policy.

```
aws iam create-role --role-name IotTopicExecutionRole --assume-role-policy-document
 file://trust-policy.json
```

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "mqttTopicTrustPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotfleetwise.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": [
            "arn:aws:iotfleetwise:region:123456789012:campaign/campaign-name"
          ],
          "aws:SourceAccount": [
            "123456789012"
          ]
        }
      }
    }
  ]
}
```

Create a permissions policy to give AWS IoT FleetWise permissions to publish messages to the MQTT topic you specified. To create a permissions policy, run the following command.

To create a permissions policy

- Replace *AWSIoTFleetwiseAccessIotTopicPermissionsPolicy* with the name of the policy you're creating.
- Replace *permissions-policy* with the name of the JSON file that contains the permissions policy.

```
aws iam create-policy --policy-name AWSIoTFleetwiseAccessIotTopicPermissionsPolicy --
policy-document file://permissions-policy.json
```

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "iot:Publish"
        ],
        "Resource": [
          "topic-arn"
        ]
      }
    ]
}
```

To attach the permissions policy to your IAM role

1. From the output, copy the Amazon Resource Name (ARN) of the permissions policy.
2. To attach the IAM permissions policy to your IAM role, run the following command.
   - Replace *permissions-policy-arn* with the ARN that you copied in the previous step.
   - Replace *IotTopicExecutionRole* with the name of the IAM role that you created.

```
aws iam attach-role-policy --policy-arn permissions-policy-arn --role-
name IotTopicExecutionRole
```

For more information, see Access management for AWS resources in the *IAM User Guide*.

Grant AWS IoT FleetWise access to an Amazon S3 destination

When you use an Amazon S3 destination, AWS IoT FleetWise delivers vehicle data to your S3 bucket and can optionally use an AWS KMS key that you own for data encryption. If error logging is enabled, AWS IoT FleetWise also sends data delivery errors to your CloudWatch log group and streams. You're required to have an IAM role when creating a delivery stream.

AWS IoT FleetWise uses a bucket policy with the service principal for the S3 destination. For more information about adding bucket policies, see Adding a bucket policy by using the Amazon S3 console in the *Amazon Simple Storage Service User Guide*.

Use the following access policy to enable AWS IoT FleetWise to access your S3 bucket. If you don't own the S3 bucket, add `s3:PutObjectAcl` to the list of Amazon S3 actions. This grants the bucket owner full access to the objects delivered by AWS IoT FleetWise. For more information about how you can secure access to objects in your buckets, see Bucket policy examples in the *Amazon Simple Storage Service User Guide*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotfleetwise.amazonaws.com"
        ]
      },
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": "arn:aws:s3:::bucket-name"
```

```
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotfleetwise.amazonaws.com"
        ]
      },
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::bucket-name/*",
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "campaign-arn",
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

The following bucket policy is for all campaigns in an account in an AWS Region.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotfleetwise.amazonaws.com"
        ]
      },
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": "arn:aws:s3:::bucket-name"
```

```
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotfleetwise.amazonaws.com"
        ]
      },
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::bucket-name/*",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:iotfleetwise:region:123456789012:campaign/*",
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

If you have a KMS key attached to your S3 bucket, the key will need the following policy. For information about key management, see Protecting data using server-side encryption with AWS Key Management Service keys (SSE-KMS) in the *Amazon Simple Storage Service User Guide*.

```
{
  "Version": "2012-10-17",
  "Effect": "Allow",
  "Principal": {
    "Service": "iotfleetwise.amazonaws.com"
  },
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
   ],
  "Resource": "key-arn"
}
```

Important

When you create a bucket, S3 creates a default access control lists (ACL) that grants the resource owner full control over the resource. If AWS IoT FleetWise can't deliver data to S3, make sure you disable the ACL on the S3 bucket. For more information, see Disabling ACLs for all new buckets and enforcing Object Ownership in the *Amazon Simple Storage Service User Guide*.

Grant AWS IoT FleetWise access to an Amazon Timestream destination

When you use a Timestream destination, AWS IoT FleetWise delivers vehicle data to a Timestream table. You must attach the policies to the IAM role to allow AWS IoT FleetWise to send data to Timestream.

If you use the console to create a campaign, AWS IoT FleetWise automatically attaches the required policy to the role.

Note

Amazon Timestream is not available in the Asia Pacific (Mumbai) Region.

Before you start, check the following:

Important

- You must use the same AWS Region when you create Timestream resources for AWS IoT FleetWise. If you switch AWS Regions, you might have issues accessing the Timestream resources.
- AWS IoT FleetWise is available in US East (N. Virginia), Europe (Frankfurt), and Asia Pacific (Mumbai).
- For the list of supported Regions, see Timestream endpoints and quotas in the *AWS General Reference*.

- You must have a Timestream database. For a tutorial, see Create a database in the *Amazon Timestream Developer Guide*.

- You must have a table created in the specified Timestream database. For a tutorial, see Create a table in the *Amazon Timestream Developer Guide*.

You can use the AWS CLI to create an IAM role with a trust policy for Timestream. To create an IAM role, run the following command.

To create an IAM role with a trust policy

- Replace *TimestreamExecutionRole* with the name of the role you're creating.
- Replace *trust-policy* with the .json file that contains the trust policy.

```
aws iam create-role --role-name TimestreamExecutionRole --assume-role-policy-document
 file://trust-policy.json
```

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "timestreamTrustPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotfleetwise.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
           "aws:SourceArn": [
            "arn:aws:iotfleetwise:region:123456789012:campaign/campaign-name"
           ],
           "aws:SourceAccount": [
            "123456789012"
          ]
        }
      }
    }
  ]
```

```
}
```

Create a permissions policy to give AWS IoT FleetWise permissions to write data into Timestream. To create a permissions policy, run the following command.

To create a permissions policy

- Replace *AWSIoTFleetwiseAccessTimestreamPermissionsPolicy* with the name of the policy you're creating.
- Replace *permissions-policy* with the name of the JSON file that contains the permissions policy.

```
aws iam create-policy --policy-name AWSIoTFleetwiseAccessTimestreamPermissionsPolicy --
policy-document file://permissions-policy.json
```

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "timestreamIngestion",
      "Effect": "Allow",
      "Action": [
        "timestream:WriteRecords",
        "timestream:Select",
        "timestream:DescribeTable"
      ],
      "Resource": "table-arn"
    },
    {
      "Sid": "timestreamDescribeEndpoint",
      "Effect": "Allow",
      "Action": [
        "timestream:DescribeEndpoints"
      ],
      "Resource": "*"
    }
```

```
  ]
}
```

To attach the permissions policy to your IAM role

1. From the output, copy the Amazon Resource Name (ARN) of the permissions policy.
2. To attach the IAM permissions policy to your IAM role, run the following command.

   - Replace *permissions-policy-arn* with the ARN that you copied in the previous step.
   - Replace *TimestreamExecutionRole* with the name of the IAM role that you created.

```
aws iam attach-role-policy --policy-arn permissions-policy-arn --role-
name TimestreamExecutionRole
```

For more information, see Access management for AWS resources in the *IAM User Guide*.

Grant AWS IoT Device Management permission to generate the payload for commands with AWS IoT FleetWise

When you use the commands feature to start a command execution, AWS IoT Device Management will fetch the command and command parameters from the incoming request. It then requires permissions to access AWS IoT FleetWise resources to validate the request and generate the payload. The payload is then sent to the vehicle by AWS IoT Device Management over MQTT to the command request topic that your vehicle has subscribed to.

You must first create an IAM role that grants AWS IoT Device Management the required permissions for generating the payload. Then, provide the ARN of this role to the `CreateCommand` API using the `roleArn` field. The following shows some policy examples.

> **Important**
>
> For the IAM role, you must use the same AWS Region as the one where you created the vehicle and command resources. If you switch AWS Region, you might have issues accessing the resources.

The IAM role need to have the following trust policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "RemoteCommandsTrustPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Grant permissions to all vehicles (IoT things)

The following example shows how to grant permissions to generate the payload for all vehicles registered as AWS IoT things.

> **Note**
>
> - This policy can be overly permissive. Use the principle of least privilege to make sure that you grant only the necessary permissions.
> - To deny permissions instead, change `"Effect": "Allow"` to `"Effect": "Deny"` in the IAM policy.

In this example, replace:

- *region* with your AWS Region where you are using the AWS IoT FleetWise resources.
- *111122223333* with your AWS account number.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotfleetwise:GenerateCommandPayload",
            "Resource": "*"
        }
    ]
}
```

Grant permission to specific vehicle (IoT thing)

The following example shows how to grant permissions to generate the payload for a specific vehicle registered as an AWS IoT thing.

In this example, replace:

- *region* with your AWS Region where you are using the AWS IoT FleetWise resources.
- *111122223333* with your AWS account number.
- *<VEHICLE_NAME>* with the IoT thing name for your vehicle .

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotfleetwise:GenerateCommandPayload",
            "Resource": "arn:aws:iot:us-east-1:111122223333:thing/<VEHICLE_NAME>"
        }
    ]
}
```

Grant permissions to specific vehicles and signals

The following example shows how to grant permissions to generate the payload for the actuator for a specific vehicle.

In this example, replace:

- *region* with your AWS Region where you are using the AWS IoT FleetWise resources.
- *111122223333* with your AWS account number.
- *<VEHICLE_NAME>* with the IoT thing name for your vehicle.
- *<SIGNAL_FQN>* with the name of the signal, such as *<Vehicle.actuator2>*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": "iotfleetwise:GenerateCommandPayload",
            "Resource": "arn:aws:iot:us-
east-1:111122223333:thing/<VEHICLE_NAME>",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iotfleetwise:Signals": [
                        "<SIGNAL_FQN>"
                    ]
                }
            }
        }
    ]
}
```

Grant permissions to specific vehicles and state templates

The following example shows how to grant permissions to generate the payload for a specific vehicle and state template.

In this example, replace:

- *region* is your AWS Region where you are using the AWS IoT FleetWise resources.
- *111122223333* is your AWS account number.
- *<VEHICLE_NAME>* is the IoT thing name for your vehicle.
- *<STATE_TEMPLATE_ID>* with the identifier of your state template.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": "iotfleetwise:GenerateCommandPayload",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:thing/<VEHICLE_NAME>",
                "arn:aws:iotfleetwise:us-east-1:111122223333:state-
template/<STATE_TEMPLATE_ID>"
            ]
        }
    ]
}
```

Grant permissions to use customer managed KMS keys

If you've enabled customer managed KMS keys for AWS IoT FleetWise, then the following example shows how to grant permissions to generate the payload.

In this example, replace:

- *region* with your AWS Region where you are using the AWS IoT FleetWise resources.
- *111122223333* with your AWS account number.
- *<KMS_KEY_ID>* with the ID of your KMS key.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotfleetwise:GenerateCommandPayload",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/<KMS_KEY_ID>"
        }
    ]
}
```

Identity and Access Management for AWS IoT FleetWise

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS IoT FleetWise resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS IoT FleetWise identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS IoT FleetWise works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for AWS IoT FleetWise)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS IoT FleetWise works with IAM

Before you use IAM to manage access to AWS IoT FleetWise, learn what IAM features are available to use with AWS IoT FleetWise.

IAM features you can use with AWS IoT FleetWise

IAM feature	AWS IoT FleetWise support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	No

To get a high-level view of how AWS IoT FleetWise and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS IoT FleetWise

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS IoT FleetWise

To view examples of AWS IoT FleetWise identity-based policies, see Identity-based policy examples for AWS IoT FleetWise.

Resource-based policies within AWS IoT FleetWise

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS IoT FleetWise

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS IoT FleetWise actions, see Actions Defined by AWS IoT FleetWise in the *Service Authorization Reference*.

Policy actions in AWS IoT FleetWise use the following prefix before the action:

```
iotfleetwise
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "iotfleetwise:action1",
      "iotfleetwise:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "iotfleetwise:List*"
```

To view examples of AWS IoT FleetWise identity-based policies, see Identity-based policy examples for AWS IoT FleetWise.

Policy resources for AWS IoT FleetWise

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS IoT FleetWise resource types and their ARNs, see Resources Defined by AWS IoT FleetWise in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS IoT FleetWise .

To view examples of AWS IoT FleetWise identity-based policies, see Identity-based policy examples for AWS IoT FleetWise.

Policy condition keys for AWS IoT FleetWise

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS IoT FleetWise condition keys, see Condition Keys for AWS IoT FleetWise in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS IoT FleetWise .

To view examples of AWS IoT FleetWise identity-based policies, see Identity-based policy examples for AWS IoT FleetWise.

Access control lists (ACLs) in AWS IoT FleetWise

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with AWS IoT FleetWise

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

> **Note**
>
> AWS IoT FleetWise only supports `iam:PassRole`, which is required for the `CreateCampaign` API operation.

Using Temporary credentials with AWS IoT FleetWise

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for AWS IoT FleetWise

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS IoT FleetWise

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break AWS IoT FleetWise functionality. Edit service roles only when AWS IoT FleetWise provides guidance to do so.

Service-linked roles for AWS IoT FleetWise

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Using service-linked roles for AWS IoT FleetWise

AWS IoT FleetWise uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS IoT FleetWise. Service-linked roles are predefined by AWS IoT FleetWise and include the permissions that AWS IoT FleetWise needs to send metrics to Amazon CloudWatch. For more information, see Monitor AWS IoT FleetWise with Amazon CloudWatch.

A service-linked role makes setting up AWS IoT FleetWise quicker because you don't have to manually add the necessary permissions. AWS IoT FleetWise defines the permissions of its service-linked roles, and unless defined otherwise, only AWS IoT FleetWise can assume its roles. The defined permissions include the trust policy and the permissions policy. This permissions policy can't be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your AWS IoT FleetWise resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM, and look for the services that have **Yes** in the **Service-linked roles** column. To view the service-linked role documentation for that service, choose a **Yes** with a link.

Service-linked role permissions for AWS IoT FleetWise

AWS IoT FleetWise uses the service-linked role named **AWSServiceRoleForIoTFleetWise** – An AWS managed policy that is used for all out-of-the-box permissions for AWS IoT FleetWise.

The AWSServiceRoleForIoTFleetWise service-linked role trusts the following services to assume the role:

- `IoTFleetWise`

The role permissions policy named AWSIoTFleetwiseServiceRolePolicy allows AWS IoT FleetWise to complete the following actions on the specified resources:

- Action: `cloudwatch:PutMetricData` on resource: *

For information about changes to this policy, see AWSIoTFleetwiseServiceRolePolicy policy updates.

The service-linked role has permissions to publish metrics to the following CloudWatch namespaces:

- `AWS/IoTFleetWise` – For service-specific metrics
- `AWS/Usage` – For usage metrics

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for AWS IoT FleetWise

You don't need to manually create a service-linked role. When you register an account in the AWS IoT FleetWise console, the AWS CLI, or the AWS API, AWS IoT FleetWise creates the service-linked role for you. For more information, see Configure your AWS IoT FleetWise settings.

Creating a service-linked role in AWS IoT FleetWise (console)

You don't need to manually create a service-linked role. When you register an account in the AWS IoT FleetWise console, the AWS CLI, or the AWS API, AWS IoT FleetWise creates the service-linked role for you.

Editing a service-linked role for AWS IoT FleetWise

You can't edit the AWSServiceRoleForIoTFleetWise service-linked role in AWS IoT FleetWise. Because various entities might reference any service-linked role you create, you can't change the name of the role. However, you can edit the description of the role by using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
>
> If AWS IoT FleetWise is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again. To learn how to delete the service-linked-role through the console, AWS CLI, or AWS API, see Using service-linked roles in the *IAM User Guide*.

If you delete this service-linked role, and then need to create it again, you can register an account with AWS IoT FleetWise. AWS IoT FleetWise then creates the service-linked role for you again.

Identity-based policy examples for AWS IoT FleetWise

By default, users and roles don't have permission to create or modify AWS IoT FleetWise resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AWS IoT FleetWise, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for AWS IoT FleetWise in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS IoT FleetWise resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS IoT FleetWise console

To access the AWS IoT FleetWise console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS IoT FleetWise resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AWS IoT FleetWise console, also attach the AWS IoT FleetWise `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Access resources in Amazon Timestream

Before using AWS IoT FleetWise, you must register your AWS account, IAM, and Amazon Timestream resources to grant AWS IoT FleetWise permission to send vehicle data to AWS Cloud on your behalf. To register, you need:

- An Amazon Timestream database.
- A table created in the specified Amazon Timestream database.
- An IAM role that allows AWS IoT FleetWise to send data to Amazon Timestream.

For more information, including procedures and example policies, see Configure your AWS IoT FleetWise settings.

Troubleshooting AWS IoT FleetWise identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS IoT FleetWise and IAM.

Topics

I am not authorized to perform an action in AWS IoT FleetWise

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`myVehicle`* resource but does not have the `iotfleetwise:`*`GetVehicleStatus`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotfleetwise:GetVehicleStatus on resource: myVehicle
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`myVehicle`* resource using the `iotfleetwise:`*`GetVehicleStatus`* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS IoT FleetWise.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS IoT FleetWise. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS IoT FleetWise resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS IoT FleetWise supports these features, see How AWS IoT FleetWise works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

AWS IoT FleetWise API actions and resources reference

When you're Managing access using policies for an IAM identity, you can use the following table as a reference. The table lists each AWS IoT FleetWise API, the corresponding actions for which you

can grant permissions to perform the action, and the AWS resource for which you can grant the permissions.

Specify the actions in the policy's `Action` field, and the resource value in the policy's `Resource` field. To specify an action, use the `iotfleetwise:` prefix followed by the action name. For example, `iotfleetwise:CreateSignalCatalog`.

Currently, AWS IoT FleetWise supports the following Actions, resources, and condition keys.

Use the scroll bar to see the rest of the table.

AWS IoT FleetWise actions	Required permissions	Resources
AssociateVehicleFleet	iotfleetwise:AssociateVehicleFleet	arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id* arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id*
CreateCampaign	iotfleetwise:CreateCampaign	arn:*partition* :iotfleetwise:*region*:*account_id* :campaign/*campaign-name* arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id*
CreateDecoderManifest	iotfleetwise:CreateDecoderManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*

AWS IoT FleetWise actions	Required permissions	Resources
		arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name*
CreateFleet	iotfleetwise:CreateFleet	arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id* arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*
CreateModelManifest	iotfleetwise:CreateModelManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*
CreateSignalCatalog	iotfleetwise:CreateSignalCatalog	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*
CreateStateTemplate	iotfleetwise:CreateStateTemplate	arn:*partition* :iotfleetwise:*region*:*account_id* :state-template/*state-template-id* arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*

AWS IoT FleetWise actions	Required permissions	Resources
CreateVehicle	iotfleetwise:CreateVehicle	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*
BatchCreateVehicle	iotfleetwise:CreateVehicle	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*
DeleteCampaign	iotfleetwise:DeleteCampaign	arn:*partition* :iotfleetwise:*region*:*account_id* :campaign/*campaign-name*
DeleteDecoderManifest	iotfleetwise:DeleteDecoderManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*
DeleteFleet	iotfleetwise:DeleteFleet	arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id*

AWS IoT FleetWise actions	Required permissions	Resources
DeleteModelManifest	iotfleetwise:DeleteModelManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*
DeleteSignalCatalog	iotfleetwise:DeleteSignalCatalog	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*
DeleteStateTemplate	iotfleetwise:DeleteStateTemplate	arn:*partition* :iotfleetwise:*region*:*account_id* :state-template/*state-template-id*
DeleteVehicle	iotfleetwise:DeleteVehicle	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id*
DisassociateVehicleFleet	iotfleetwise:DisassociateVehicleFleet	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id*
GetCampaign	iotfleetwise:GetCampaign	arn:*partition* :iotfleetwise:*region*:*account_id* :campaign/*campaign-name*
GetDecoderManifest	iotfleetwise:GetDecoderManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*
GetEncryptionConfiguration	iotfleetwise:GetEncryptionConfiguration	

AWS IoT FleetWise actions	Required permissions	Resources
GetFleet	iotfleetwise:GetFleet	arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id*
GetLoggingOptions	iotfleetwise:GetLoggingOptions	
GetModelManifest	iotfleetwise:GetModelManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name*
GetRegisterAccountStatus	iotfleetwise:GetRegisterAccountStatus	
GetSignalCatalog	iotfleetwise:GetSignalCatalog	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*
GetStateTemplate	iotfleetwise:GetStateTemplate	arn:*partition* :iotfleetwise:*region*:*account_id* :state-template/*state-template-id*
GetVehicle	iotfleetwise:GetVehicle	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id*
GetVehicleStatus	iotfleetwise:GetVehicleStatus	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id*
ImportDecoderManifest	iotfleetwise:ImportDecoderManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*
ImportSignalCatalog	iotfleetwise:ImportSignalCatalog	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*

AWS IoT FleetWise actions	Required permissions	Resources
ListCampaigns	iotfleetwise:ListCampaigns	
ListDecoderManifestNetworkInterfaces	iotfleetwise:ListDecoderManifestNetworkInterfaces	arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*
ListDecoderManifests	iotfleetwise:ListDecoderManifests	
ListDecoderManifestSignals	iotfleetwise:ListDecoderManifestSignals	arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*
ListFleets	iotfleetwise:ListFleets	
ListFleetsForVehicle	iotfleetwise:ListFleetsForVehicle	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id*
ListModelManifestNodes	iotfleetwise:ListModelManifestNodes	arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name*
ListModelManifests	iotfleetwise:ListModelManifests	
ListSignalCatalogNodes	iotfleetwise:ListSignalCatalogNodes	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*
ListSignalCatalogs	iotfleetwise:ListSignalCatalogs	
ListStateTemplates	iotfleetwise:ListStateTemplates	

AWS IoT FleetWise actions	Required permissions	Resources
ListVehicles	iotfleetwise:ListVehicles	
ListVehiclesInFleet	iotfleetwise:ListVehiclesInFleet	arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id*
ListTagsForResource	iotfleetwise:ListTagsForResource	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id* arn:*partition* :iotfleetwise:*region*:*account_id* :campaign/*campaign-name* arn:*partition* :iotfleetwise:*region*:*account_id* :state-template/*state-template-id*
PutEncryptionConfiguration	iotfleetwise:PutEncryptionConfiguration	

AWS IoT FleetWise actions	Required permissions	Resources
PutLoggingOptions	iotfleetwise:PutLoggingOptions	
RegisterAccount	iotfleetwise:RegisterAccount	
TagResource	iotfleetwise:TagResource	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id* arn:*partition* :iotfleetwise:*region*:*account_id* :campaign/*campaign-name* arn:*partition* :iotfleetwise:*region*:*account_id* :state-template/*state-template-id*

AWS IoT FleetWise actions	Required permissions	Resources
UntagResource	iotfleetwise:UntagResource	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id* arn:*partition* :iotfleetwise:*region*:*account_id* :campaign/*campaign-name* arn:*partition* :iotfleetwise:*region*:*account_id* :state-template/*state-template-id*
UpdateCampaign	iotfleetwise:UpdateCampaign	arn:*partition* :iotfleetwise:*region*:*account_id* :campaign/*campaign-name*
UpdateDecoderManifest	iotfleetwise:UpdateDecoderManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*

AWS IoT FleetWise actions	Required permissions	Resources
UpdateFleet	iotfleetwise:UpdateFleet	arn:*partition* :iotfleetwise:*region*:*account_id* :fleet/*fleet-id*
UpdateModelManifest	iotfleetwise:UpdateModelManifest	arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name*
UpdateSignalCatalog	iotfleetwise:UpdateSignalCatalog	arn:*partition* :iotfleetwise:*region*:*account_id* :signal-catalog/*name*
UpdateStateTemplate	iotfleetwise:UpdateStateTemplate	arn:*partition* :iotfleetwise:*region*:*account_id* :state-template/*state-template-id*
UpdateVehicle	iotfleetwise:UpdateVehicle	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*

AWS IoT FleetWise actions	Required permissions	Resources
BatchUpdateVehicle	iotfleetwise:UpdateVehicle	arn:*partition* :iotfleetwise:*region*:*account_id* :vehicle/*vehicle-id* arn:*partition* :iotfleetwise:*region*:*account_id* :model-manifest/*name* arn:*partition* :iotfleetwise:*region*:*account_id* :decoder-manifest/*name*

AWS managed policy updates for AWS IoT FleetWise

View details about updates to AWS managed policies for AWS IoT FleetWise since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS IoT FleetWise Document history page.

AWSIoTFleetwiseServiceRolePolicy

This policy allows AWS IoT FleetWise to publish metrics to Amazon CloudWatch on your behalf.

Change	Description	Date
`AWSIoTFleetwiseServiceRolePolicy` – Update to an existing policy	Added permissions to publish usage metrics to the AWS/Usage namespace in addition to the existing `AWS/IoTFleetWise` namespace permissions.	June 13, 2025
`AWSIoTFleetwiseServiceRolePolicy` – AWS IoT FleetWise	AWS IoT FleetWise started tracking changes for this policy.	September 27, 2022

Change	Description	Date
started tracking changes		

Compliance Validation for AWS IoT FleetWise

> **Note**
>
> AWS IoT FleetWise isn't in scope of any AWS compliance programs.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in AWS IoT FleetWise

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Note

Data processed by AWS IoT FleetWise is stored in an Amazon Timestream database. Timestream supports backups to other AWS Availability Zones or Regions. However, you can write your own application using the Timestream SDK to query data and save it to the destination of your choice.
For more information about Amazon Timestream, see the *in the Amazon Timestream Developer Guide*.
Amazon Timestream is not available in the Asia Pacific (Mumbai) region.

Infrastructure security in AWS IoT FleetWise

As a managed service, AWS IoT FleetWise is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access AWS IoT FleetWise through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

You can call these API operations from any network location, but AWS IoT FleetWise does support resource-based access policies, which can include restrictions based on the source IP address. You can also use AWS IoT FleetWise policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given AWS IoT FleetWise resource from only the specific VPC within the AWS network.

Topics

- Connecting to AWS IoT FleetWise through an interface VPC endpoint

Connecting to AWS IoT FleetWise through an interface VPC endpoint

You can connect directly to AWS IoT FleetWise by using an interface VPC endpoint (AWS PrivateLink) in your Virtual Private Cloud (VPC), instead of connecting over the internet. When you use an interface VPC endpoint, communication between your VPC and AWS IoT FleetWise is conducted entirely within the AWS network. Each VPC endpoint is represented by one or more Elastic network interfaces (ENIs) with private IP addresses in your VPC subnets.

The interface VPC endpoint connects your VPC directly to AWS IoT FleetWise without an internet gateway, NAT device, VPN connection, or Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the AWS IoT FleetWise API.

To use AWS IoT FleetWise through your VPC, you must connect from an instance that is inside the VPC or connect your private network to your VPC by using an AWS Virtual Private Network (VPN) or Direct Connect. For information about Amazon VPN, see VPN connections in the *Amazon Virtual Private Cloud User Guide*. For information about AWS Direct Connect, see Creating a connection in the *Direct Connect User Guide*.

You can create an interface VPC endpoint to connect to AWS IoT FleetWise by using the AWS console or AWS Command Line Interface (AWS CLI) commands. For more information, see Creating an interface endpoint.

After you create an interface VPC endpoint, if you enable private DNS hostnames for the endpoint, the default AWS IoT FleetWise endpoint resolves to your VPC endpoint. The default service name endpoint for AWS IoT FleetWise is in the following format.

```
iotfleetwise.Region.amazonaws.com
```

If you don't enable private DNS hostnames, Amazon VPC provides a DNS endpoint name that you can use in the following format.

```
VPCE_ID.iotfleetwise.Region.vpce.amazonaws.com
```

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

AWS IoT FleetWise supports making calls to all of its API actions inside your VPC.

You can attach VPC endpoint policies to a VPC endpoint to control access for IAM principals. You can also associate security groups with a VPC endpoint to control inbound and outbound access

based on the origin and destination of network traffic, such as a range of IP addresses. For more information, see Controlling access to services with VPC endpoints.

> **Note**
>
> AWS IoT FleetWise supports all VPC endpoints with dual-stack mode. For information about service endpoints, see AWS IoT FleetWise endpoints and quotas.

Creating a VPC endpoint policy for AWS IoT FleetWise

You can create a policy for Amazon VPC endpoints for AWS IoT FleetWise to specify the following:

- The principal that can or can't perform actions
- The actions that can or can't be performed

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example– VPC endpoint policy to deny all access from a specified AWS account

The following VPC endpoint policy denies AWS account *123456789012* all API calls using the endpoint.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
```

```
        }
    ]
}
```

Example– VPC endpoint policy to allow VPC access only to a specified IAM principal (user)

The following VPC endpoint policy allows full access only to the a user *lijuan* in AWS account *123456789012*. It denies all other IAM principals access to the endpoint.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:user/lijuan"
                ]
            }
        }]
}
```

Example– VPC endpoint policy for AWS IoT FleetWise actions

The following is an example of an endpoint policy for AWS IoT FleetWise. When attached to an endpoint, this policy grants access to the listed AWS IoT FleetWise actions for the IAM user *fleetWise* in the AWS account *123456789012*.

```
{
    "Statement": [
        {
             "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:user/fleetWise"
                },
            "Resource": "*",
            "Effect": "Allow",
            "Action": [
                "iotfleetwise:ListFleets",
                "iotfleetwise:ListCampaigns",
                "iotfleetwise:CreateVehicle",
```

```
            ]
          }
    ]
}
```

Configuration and vulnerability analysis in AWS IoT FleetWise

IoT environments can consist of large numbers of devices that have diverse capabilities, are long-lived, and are geographically distributed. These characteristics make device setup complex and error-prone. Also, because devices are often constrained in computational power, memory, and storage capabilities, the use of encryption and other forms of security on the devices is limited. Devices often use software with known vulnerabilities. These factors make IoT devices, including vehicles collecting data for AWS IoT FleetWise, an attractive target for hackers and make it difficult to secure them on an ongoing basis.

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Security best practices for AWS IoT FleetWise

AWS IoT FleetWise provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

To learn about security in AWS IoT see Security best practices in AWS IoT Core in the *AWS IoT Developer Guide*

Grant minimum possible permissions

Follow the principle of least privilege by using the minimum set of permissions in IAM roles. Limit the use of the `*` wildcard for the `Action` and `Resource` properties in your IAM policies. Instead, declare a finite set of actions and resources when possible. For more information about least privilege and other policy best practices, see the section called “Policy best practices”.

Don't log sensitive information

You should prevent the logging of credentials and other personally identifiable information (PII). We recommend that you implement the following safeguards:

- Don't use sensitive information in device names.
- Don't use sensitive information in the names and IDs of AWS IoT FleetWise resources, for example in the names of campaigns, decoder manifests, vehicle models, and signal catalogs, or the IDs of vehicles and fleets.

Use AWS CloudTrail to view API call history

You can view a history of AWS IoT FleetWise API calls made on your account for security analysis and operational troubleshooting purposes. To receive a history of AWS IoT FleetWise API calls made on your account, simply turn on CloudTrail in the AWS Management Console. For more information, see the section called “CloudTrail logs”.

Keep your device clock in sync

It's important to have an accurate time on your device. X.509 certificates have an expiry date and time. The clock on your device is used to verify that a server certificate is still valid. Device clocks can drift over time or batteries can get discharged.

For more information, see the Keep your device's clock in sync best practice in the *AWS IoT Core Developer Guide*.

Monitor AWS IoT FleetWise

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS IoT FleetWise and your other AWS solutions. AWS provides the following monitoring tools to watch AWS IoT FleetWise, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Logs* can be used to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account. Then, it delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

Monitor AWS IoT FleetWise with Amazon CloudWatch

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Amazon CloudWatch metrics are a way to monitor your AWS resources and how they're performing. AWS IoT FleetWise sends metrics to CloudWatch. You can use the AWS Management Console, the AWS CLI, or an API to list the metrics that AWS IoT FleetWise sends to CloudWatch. For more information, see the Amazon CloudWatch User Guide.

Important

You must configure settings so that AWS IoT FleetWise can send metrics to CloudWatch. For more information, see Configure your AWS IoT FleetWise settings.

The `AWS/IoTFleetWise` namespace includes the following metrics.

Signal metrics

Metric	Description
IllegalMessageFromEdge	A message sent from the vehicle and received by AWS IoT FleetWise didn't match the required format. Units: Count Dimensions: None Valid statistics: Sum
MessageThrottled	A message sent from the vehicle to AWS IoT FleetWise was throttled. This is because you exceeded the service limits for this account in the current Region. Units: Count Dimensions: None Valid statistics: Sum
ModelingError	A message sent from the vehicle and received by AWS IoT FleetWise contains signals that fail to validate against the vehicle model. Units: Count Dimensions: ModelName, StateTemplateName (Optional), SignalCatalogName (Optional)

Metric	Description
DecodingError	A message sent from the vehicle and received by AWS IoT FleetWise contains signals that fail to decoder against the vehicle's decoder manifest. Units: Count Dimensions: DecoderName Valid statistics: Sum
MessageSizeLimitExceeded	A message sent from the vehicle to AWS IoT FleetWise was dropped. This is because you exceeded the maximum size of a message service limit for this account in the current Region. Units: Count Dimensions: None Valid statistics: Sum
CallCount	The number of messages ingested over the specified time period. Units: Count Dimensions: AccountID

Metric	Description
CheckInThrottled	A check-in sent from the vehicle to AWS IoT FleetWise was throttled. This is because you exceeded the service limit for this account in the current Region. Units: Count Dimensions: VehicleName Valid statistics: Sum
VehicleAttributeNotFound	A message sent from the vehicle and received by AWS IoT FleetWise could not be enriched with the specified vehicle attributes. Units: Count Dimensions: campaignName (optional), stateTemplateName (optional), vehicleName Valid statistics: Sum

Vehicle metrics

Metric	Description
VehicleNotFound	A message received by AWS IoT FleetWise, where the vehicle is unknown. Units: Count Dimensions: None Valid statistics: Sum

Deployment metrics

Metric	Description
PayloadSize	Size of the message sent from AWS IoT FleetWise to the vehicle. Units: Count Dimensions: VehicleName, ResourceTypes (StateTemplates, Campaigns, DecoderManifest)
PayloadSizeLimitExceeded	A message sent from AWS IoT FleetWise to the vehicle exceeded the maximum size of a payload service limit for this account in the current Region. Units: Count Dimensions: VehicleName Valid statistics: Sum

Campaign metrics

Metric	Description
CampaignInvalid	A message sent from the vehicle and received by AWS IoT FleetWise, where the campaign isn't valid. Units: Count Dimensions: CampaignName Valid statistics: Sum

Metric	Description
CampaignNotFound	A message sent from the vehicle and received by AWS IoT FleetWise, where the campaign is unknown. Units: Count Dimensions: CampaignName Valid statistics: Sum

State template metrics

Metric	Description
NoStateTemplatesAssociated	A message sent from the vehicle and received by AWS IoT FleetWise, where no state templates are associated with the vehicle. Units: Count Valid statistics: Sum

Campaign data destination metrics

Metric	Description
TimestreamWriteError	AWS IoT FleetWise couldn't write a message from the vehicle to the Amazon Timestream table. Units: Count Dimensions: DatabaseName, TableName Valid statistics: Sum

Metric	Description
S3WriteError	AWS IoT FleetWise couldn't write a message from the vehicle to the Amazon Simple Storage Service (Amazon S3) bucket. Units: Count Dimensions: BucketName Valid statistics: Sum
S3ReadError	AWS IoT FleetWise couldn't read an object key from the vehicle in the Amazon Simple Storage Service (Amazon S3) bucket. Units: Count Dimensions: BucketName Valid statistics: Sum

Customer managed AWS KMS key metrics

Metric	Description
KMSKeyAccessDenied	AWS IoT FleetWise couldn't write a message from the vehicle to the Timestream table or the Amazon S3 bucket because of an AWS KMS key access denied error. Units: Count Dimensions: KMSKeyId Valid statistics: Sum

Monitor AWS IoT FleetWise with Amazon CloudWatch Logs

> **Important**
>
> Access to certain AWS IoT FleetWise features is currently gated. For more information, see AWS Region and feature availability in AWS IoT FleetWise.

Amazon CloudWatch Logs monitors the events that occur in your resources and alerts you if there are any issues. If you receive an alert, you can access the log files to get information about the specific event. For more information, see the Amazon CloudWatch Logs User Guide.

View AWS IoT FleetWise logs in the CloudWatch console

> **Important**
>
> Before you can see the AWS IoT FleetWise log group in the CloudWatch console, make sure that the following is true:
>
> - You've enabled logging in AWS IoT FleetWise. For more information about logging, see Configure AWS IoT FleetWise logging.
> - There are already log entries written by AWS IoT operations.

To view your AWS IoT FleetWise logs in the CloudWatch console

1. Open the CloudWatch console.
2. On the navigation pane, choose **Logs**, **Log groups**.
3. Choose the log group.
4. Choose **Search log group**. You'll see a complete list of the log events generated for your account.
5. Choose the expand icon to look at an individual stream and find all logs that have a log level of ERROR.

   You can also enter a query in the **Filter events** search box. For example, you can try the following query:

   ```
   { $.logLevel = "ERROR" }
   ```

For more information about creating filter expressions, see Filter and pattern syntax in the *Amazon CloudWatch Logs User Guide*.

Example log entry

```
{
  "accountId": "123456789012",
  "vehicleName": "test-vehicle",
  "message": "Unrecognized signal ID",
  "eventType": "MODELING_ERROR",
  "logLevel": "ERROR",
  "timestamp": 1685743214239,
  "campaignName": "test-campaign",
  "signalCatalogName": "test-catalog",
  "signalId": 10242
}
```

Signal event types

Event type	Description
MODELING_ERROR	A message sent from the vehicle and received by AWS IoT FleetWise contains signals that fail to validate against the vehicle model. Attributes: vehicleName, campaignName (optional), signalCatalogName, signalId (optional), signalValue (optional), signalValueRangeMin (optional), signalValueRangeMax (optional), modelManifestName (optional), signalIds, stateTemplateName
ILLEGAL_MESSAGE_FROM_EDGE	A message sent from the vehicle and received by AWS IoT FleetWise didn't match the required format. Attributes: vehicleName, campaignName, signalCatalogName
DECODING_ERROR	A message sent from the vehicle and received by AWS IoT FleetWise contains signals that

Event type	Description
	fail to decoder against the vehicle's decoder manifest. Attributes: campaignName, signalCatalogName, decoderManifestName, (optional) signalName, (optional) s3URI
MESSAGE_THROTTLED	A message sent from the vehicle to AWS IoT FleetWise was throttled. This is because you exceeded the service limits for this account in the current Region. Attributes: accountId, vehicleName, message, eventType, logLevel, timestamp
MESSAGE_SIZE_LIMIT_EXCEEDED	A message sent from the vehicle and received by AWS IoT FleetWise exceeds the maximum size of a message service limit. Attributes: accountId, vehicleName
CHECKIN_THROTTLED	A check-in sent from the vehicle to AWS IoT FleetWise was throttled. This is because you exceeded the service limit for this account in the current Region. Attributes: vehicleName
VEHICLE_ATTRIBUTE_NOT_FOUND	A message sent from the vehicle and received by AWS IoT FleetWise could not be enriched with the specified vehicle attributes. Attributes: campaignName (optional), stateTemplateName (optional), vehicleName, vehicleAttributeNames

Vehicle event types

Event type	Description
VEHICLE_NOT_FOUND	A message received by AWS IoT FleetWise, where the vehicle was unknown. Attributes: vehicleName, campaignName (optional), stateTemplateName (optional)

Deployment event types

Event type	Description
PAYLOAD_SIZE_LIMIT_EXCEEDED	A message sent from AWS IoT FleetWise to the vehicle exceeded the maximum size service limit. Attributes: vehicleName, campaignName (optional), stateTemplateName (optional)

Campaign event types

Event type	Description
CAMPAIGN_NOT_FOUND	A message sent from the vehicle and received by AWS IoT FleetWise, where the campaign was unknown. Attributes: vehicleName (optional), campaignName
CAMPAIGN_INVALID	A message sent from the vehicle and received by AWS IoT FleetWise, where the campaign was not valid. Attributes: vehicleName (optional), campaignName

Campaign data destination event types

Event type	Description
TIMESTREAM_WRITE_ERROR	AWS IoT FleetWise couldn't write a message from the vehicle to the Amazon Timestream table. Attributes: vehicleName, campaignName, timestreamDatabaseName, timestrea mTableName
S3_WRITE_ERROR	AWS IoT FleetWise couldn't write a message from the vehicle to the Amazon Simple Storage Service (Amazon S3) bucket. Attributes: campaignName, destinationName
S3_READ_ERROR	AWS IoT FleetWise couldn't read an object key from the vehicle in the Amazon Simple Storage Service (Amazon S3) bucket. Attributes: campaignName, destinationName

State template event types

Event type	Description
STATE_TEMPLATE_NOT_FOUND	A message sent from the vehicle and received by AWS IoT FleetWise, where the state template was unknown. Attributes: vehicleName (optional), stateTemp lateName

Customer managed AWS KMS key event types

Event type	Description
KMS_KEY_ACCESS_DENIED	AWS IoT FleetWise couldn't write a message from the vehicle to the Timestream table or the Amazon S3 bucket because of an AWS KMS key access denied error. Attributes: kmsKeyId (optional), resourceArn (optional)

Attributes

All CloudWatch Logs entries include these attributes:

accountId

Your AWS account ID.

eventType

The event type for which the log was generated. The value of the event type depends on the event that generated the log entry. Each log entry description includes the value of `eventType` for that log entry.

logLevel

The log level that is being used. For more information, see Log levels in the *AWS IoT Core Developer Guide*.

message

Contains specific details about the log.

timestamp

The epoch millisecond timestamp of when AWS IoT FleetWise processed the log.

Optional attributes

CloudWatch Logs entries optionally include these attributes, depending on the `eventType`:

decoderManifestName

The name of the decoder manifest that contains the signal.

destinationName

The name of the destination for vehicle data. For example, the Amazon S3 bucket name.

campaignName

The name of the campaign.

signalCatalogName

The name of the signal catalog that contains the signal.

signalId

The ID of the error signal.

signalIds

A list of error signal IDs.

signalName

The name of the signal.

signalTimestampEpochMs

The timestamp of the error signal.

signalValue

The value of the error signal.

signalValueRangeMax

The maximum range of the error signal.

signalValueRangeMin

The minimum range of the error signal.

s3URI

The Amazon S3 unique identifier of an Amazon Ion file from a vehicle message.

timestreamDatabaseName

The name of the Timestream database.

timestreamTableName

The name of the Timestream table.

vehicleName

The name of the vehicle.

vehicleAttributeNames

A list of vehicle attribute names that could not be found.

Configure AWS IoT FleetWise logging

You can send your AWS IoT FleetWise log data to a CloudWatch log group. CloudWatch Logs give visibility in case AWS IoT FleetWise fails to process messages from vehicles. For example, this can happen because of a faulty configuration or other client errors. You're notified of any errors so you can identify and mitigate issues.

Before you can send logs to CloudWatch, you must create a CloudWatch log group. Configure the log group with the same account and in the same Region that you used with AWS IoT FleetWise. When you enable logging in AWS IoT FleetWise, provide the log group name. After logging is enabled, AWS IoT FleetWise delivers logs to the CloudWatch log group in log streams.

You can view log data sent from AWS IoT FleetWise in the CloudWatch console. For more information about configuring a CloudWatch log group and viewing log data, see Working with Log Groups. For more information on setting up logging for AWS services, see AWS services that publish logs to CloudWatch Logs in the *Amazon CloudWatch Logs User Guide*.

Permissions to publish logs to CloudWatch

Configuring logging for a CloudWatch log group requires the permissions settings described in this section. For information about managing permissions, see Access management for AWS resources in the *IAM User Guide*.

With these permissions, you can change the logging configuration, configure log delivery for CloudWatch, and retrieve information about your log group.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iotfleetwise:PutLoggingOptions",
                "iotfleetwise:GetLoggingOptions"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow",
            "Sid": "IoTFleetwiseLoggingOptionsAPI"
        },
        {
            "Sid": "IoTFleetwiseLoggingCWL",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

When actions are permitted on all AWS resources, it's indicated in the policy with a `"Resource"` setting of `"*"`. This means that the actions are permitted on all AWS resources *that each action supports*.

Configure logging using the console

This section describes how to use the AWS IoT FleetWise console to configure logging.

To use the AWS IoT FleetWise console to configure logging

1. Open the AWS IoT FleetWise console.
2. In the left pane, choose **Settings**.
3. In the **Logging** section of the **Settings** page, choose **Edit**.
4. In the **CloudWatch logging** section, enter the **Log group**.
5. To save your changes, choose **Submit**.

After you enable logging, you can view your log data in the CloudWatch console.

Configure logging using the CLI

This section describes how to configure logging for AWS IoT FleetWise by using the CLI.

You can also perform this procedure with the API by using the methods in the AWS API that correspond to the CLI commands shown here. You can use the GetLoggingOptions API operation to fetch the current configuration and the PutLoggingOptions API operation to modify the configuration.

To use the CLI to configure logging for AWS IoT FleetWise

1. To get the logging options for your account, use the **get-logging-options** command.

   ```
   aws iotfleetwise get-logging-options
   ```

2. To enable logging, use the **put-logging-options** command.

   ```
   aws iotfleetwise put-logging-options --cloud-watch-log-delivery
    logType=ERROR,logGroupName=MyLogGroup
   ```

   where:

   logType

   The type of log to send data to CloudWatch Logs. To disable logging, change the value to OFF.

logGroupName

The CloudWatch Logs group the operation sends data to. Make sure you create the log group name before you enable logging for AWS IoT FleetWise.

After you enable logging, see Search log entries using the AWS CLI.

Log AWS IoT FleetWise API calls using AWS CloudTrail

AWS IoT FleetWise is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS IoT FleetWise. CloudTrail captures all API calls for AWS IoT FleetWise as events. The calls captured include calls from the AWS IoT FleetWise console and code calls to the AWS IoT FleetWise API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS IoT FleetWise. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS IoT FleetWise, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS IoT FleetWise information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS IoT FleetWise, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS IoT FleetWise, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail

- Receiving CloudTrail log files from multiple Regions
- Receiving CloudTrail log files from multiple accounts

All AWS IoT FleetWise actions are logged by CloudTrail and are documented in the AWS IoT FleetWise API Reference. For example, calls to the `CreateCampaign`, `AssociateVehicleFleet`, and `GetModelManifest` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understand AWS IoT FleetWise log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the *`AssociateVehicleFleet`* operation.


```
{
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:assumed-role/NikkiWolf",
        "accountId": "111122223333",
        "accessKeyId": "access-key-id",
        "userName": "NikkiWolf"
      },
      "eventTime": "2021-11-30T09:56:35Z",
      "eventSource": "iotfleetwise.amazonaws.com",
```

```
      "eventName": "AssociateVehicleFleet",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.21",
      "userAgent": "aws-cli/2.3.2 Python/3.8.8 Darwin/18.7.0 botocore/2.0.0",
      "requestParameters": {
          "fleetId": "f1234567890",
          "vehicleId": "v0213456789"
       },
      "responseElements": {
      },
      "requestID": "9f861429-11e3-11e8-9eea-0781b5c0ac21",
      "eventID": "17385819-4927-41ee-a6a5-29ml0br812v4",
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    }
```

Document history for the *AWS IoT FleetWise Developer Guide*

The following table describes the documentation releases for AWS IoT FleetWise.

Change	Description	Date
Monitoring update	Updated the AWS IoT FleetWise monitoring to include more metrics and logging options. For more information, see Monitor AWS IoT FleetWise with Amazon CloudWatch and Monitor AWS IoT FleetWise with Amazon CloudWatch Logs.	December 23, 2025
Service-linked role policy update	Updated the `AWSIoTFleetwiseServiceRolePolicy` to include permissions for publishing usage metrics to the `AWS/Usage` namespace. For more information, see AWSIoTFleetwiseServiceRolePolicy policy updates.	June 13, 2025
Region expansion	AWS IoT FleetWise is now available in the Asia Pacific (Mumbai) Region (gated access only).	November 21, 2024
Gated general availability of new features	AWS IoT FleetWise now supports gated access for campaigns to store and forward data, configure	November 21, 2024

	an MQTT topic as a data destination, and collect diagnostic trouble code data. It also now supports gated access for network agnostic data collection using custom decoding interfaces, configuri ng commands, and monitorin g the last known state of vehicles.	
Send campaign data to an MQTT topic	AWS IoT FleetWise now supports sending data collected during campaigns to an MQTT topic that you specify, in addition to the ability to store the data in Amazon S3 or Amazon Timestream.	May 1, 2024
Vision system data preview	You can use the preview of vision system data from AWS IoT FleetWise to collect and organize data from vehicle vision systems, including from cameras, radars, and lidars. It keeps both structure d and unstructured vision system data, metadata (event ID, campaign, vehicle), and standard sensor (telemetry data) automatically synchroni zed in the cloud.	November 26, 2023

AWS KMS customer managed keys	AWS IoT FleetWise now supports AWS KMS customer managed keys. You can use KMS key to encrypt server-side data related to AWS IoT FleetWise resources (signal catalog, vehicle model, decoder manifest, vehicles, and data collection campaign configurations) stored in AWS Cloud.	October 16, 2023
Object storage in Amazon S3	AWS IoT FleetWise now supports storing data using Amazon Simple Storage Service (Amazon S3). You can store data collected during campaigns in Amazon S3, in addition to Amazon Timestream.	June 1, 2023
General availability	This is the public release of AWS IoT FleetWise.	September 27, 2022
Initial release	This is the preview release of the AWS IoT FleetWise Developer Guide.	November 30, 2021