Revised: September 15, 2025

Model-Driven Telemetry

Model-Driven Telemetry

Telemetry is an automated communications process that collects measurements and other data at remote points and transmits them to the receiving equipments for monitoring. Model-driven telemetry streams YANG-modeled data to a data collector.

Applications subscribe to specific data items using standards-based YANG data models over NETCONF, RESTCONF, or gRPC Network Management Interface (gNMI) protocols. You can also create subscriptions using CLIs for configured subscriptions.

Structured data is published at a defined cadence or on-change, based on the subscription criteria and data type.

Systems using telemetry involves different roles. This document describes these telemetry roles:

- publisher: Network element that sends the telemetry data,
- receiver: Receives the telemetry data; also called the collector,
- controller: Network element that creates subscriptions but does not receive the telemetry data. The telemetry data associated with the subscriptions it creates goes to receivers. Also called the management agent or management entity, and
- subscriber: Network element that creates subscriptions and is also the receiver in this document.

Feature history for model-driven telemetry

This table provides release and related information for the features explained in this module.

These features are available in all the releases subsequent to the one they were introduced in, unless noted otherwise.

Release	Feature	Feature Information
Cisco IOS XE 17.18.1	Model-driven telemetry	Model-driven telemetry allows network devices to continuously stream real time configuration and operating state information to subscribers. This feature was implemented on the following platforms: • Cisco C9350 Series Smart Switches • Cisco C9610 Series Smart Switches

Use the Cisco Feature Navigator to find information about platform and software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/.

Prerequisites for model-driven telemetry

These prerequisites apply to model-driven telemetry.

- Knowledge of YANG is needed to understand and define the data that is required when using telemetry.
- Knowledge of XML, XML namespaces, and XML XPath.

- Knowledge of standards and principles defined by the IETF telemetry specifications.
- The *urn:ietf:params:netconf:capability:notification:1.1* capability must be listed in hello messages. This capability is advertised only on devices that support IETF telemetry.
- NETCONF-YANG must be configured and running on the device.

Note

Either NETCONF-YANG or gNXI must be configured for telemetry to work. If your platform does not support gNXI, you must configure NETCONF, even if NETCONF is not used. For more information on configuring NETCONF-YANG, see the *NETCONF Protocol* module. For more information on gNXI, see the *gNMI Protocol* module.

Verify that the following processes are running, by using the **show platform software yang-management process** command:

```
Device# show platform software yang-management process

confd    : Running
nesd     : Running
syncfd   : Running
ncsshd   : Running
dmiauthd : Running
nginx    : Running
ndbmand  : Running
pubd     : Running
gnmib    : Running
```

Note

The process *pubd* is the model-driven telemetry process, and if it is not running, model-driven telemetry will not work.

The following table provides details about each of the Device Management Interface (DMI) processes.

Table 1: Field descriptions

DMI process name	Primary role
confd	Configuration daemon.
nesd	Network element synchronizer daemon.
syncfd	Sync daemon (maintains synchronization between the running state and corresponding models).
ncsshd	NETCONF Secure Shell (SSH) daemon.
dmiauthd	DMI authentication daemon.
nginx	NGINX web server. Acts as a web server for RESTCONF.
ndbmand	NETCONF database manager.
pubd	Publication manager and publisher used for model-driven telemetry.

DMI process name	Primary role
gnmib	GNMI protocol server.

Restrictions for model-driven telemetry

The following restrictions apply to model-driven telemetry.

- Automatic hierarchy in selections is not supported for on-change subscriptions when using the *yang-push* stream. This means that when selecting a list, child lists of the list are not automatically included. For example, the subscriber must manually create a subscription for each child list.

  This restriction also applies to periodic subscriptions, if subscribed to the elements in the list below:

  - openconfig-access-points
  - openconfig-ap-manager
  - openconfig-lacp
  - openconfig-platform-psu

- Checking the authorization of data access is not supported. All the data requested by a subscriber is sent.
- Subtree filters are not supported. If subtree filters are specified, the subscription is marked as invalid.
- Defining multiple receivers within subscription parameters is not supported; only the first receiver destination is attempted. Other defined receivers are ignored.

What are subscriptions?

Subscriptions create associations between telemetry roles and define the data sent between them.

Subscriptions define the set of data that is requested as part of the telemetry data, such as, when the data is required, how the data is formatted, which receivers should receive the data, and so on.

Telemetry uses two types of subscriptions, dynamic and configured.

- Dynamic subscriptions are created by clients or subscribers that connect to the publisher, so they are dial-in.
- Configured subscriptions cause the publisher to initiate connections to receivers, making them dial-out.

Subscriptions can either be configured or dynamic, and use any combination of transport protocols. Periodic triggered subscriptions (100 centiseconds minimum) and on-change triggered subscriptions are also supported.

The maximum number of supported subscriptions is platform-dependent, and all platforms support at least 100 subscriptions. NETCONF and other northbound programmable interfaces (such as RESTCONF or gNMI) are supported to configure subscriptions. If too many subscriptions operate simultaneously and prevent all valid configured subscriptions from being active, removing an active subscription may activate an inactive but valid configured subscription.

Data source specifications

The sources of telemetry data in a subscription are specified using a stream and a filter. Stream refers to a related set of events. RFC 5277 defines an event stream as a set of event notifications matching some forwarding criteria.

Normally, the system filters the set of events from a stream, and different filter types are used for different stream types.

Cisco IOS XE software supports two streams: *yang-push* and *yang-notif-native*.

Update notifications

As part of a subscription, you can specify when data is required; however this depends on the stream. Some streams support making data available only when there is a change, or after an event within the stream. Other streams make data available when there is a change or at a defined time period.

The *when* specification results in a series of update notifications that carry the telemetry data of interest. How the data is sent depends on the protocol used for the connection between the publisher and the receiver.

Subscription identifiers

Subscriptions are identified by a 32-bit positive integer value. The IDs for configured subscriptions are set by the controller, and for dynamic subscriptions, by the publisher.

Controllers must limit the values they use for configured subscriptions in the range 0 to 2,147,483,647 to avoid collisions with the dynamic subscriptions created on the publisher. The dynamic subscription ID space is global, meaning that the subscription IDs for independently-created dynamic subscriptions do not overlap.

Dial-in and dial-out subscriptions

The two flavors of model-driven telemetry are dial-in and dial-out subscriptions. This table compares both these types of subscriptions.

Table 2: Dial-in and dial-out subscriptions

Dial-in	**Dial-out**
Telemetry updates are sent to the initiator or subscriber.	Telemetry updates are sent to the specified receiver or collector.
Life of a subscription is tied to the connection or session that created it, and over which telemetry updates are sent. No change is observed in the running configuration.	Subscription is created as part of the running configuration; it remains as the device configuration till the configuration is removed.
Dial-in subscriptions need to be re-initiated after a reload, because established connections or sessions are killed during a stateful switchover.	Dial-out subscriptions are created as part of the device configuration and they automatically reconnect to the receiver after a stateful switchover.
Subscription ID is dynamically generated upon the successful establishment of a subscription.	Subscription ID is fixed and configured on the device as part of the configuration.

Streams

Streams define a set of events that can be subscribed to, and this set of events can be almost anything. However, as per the definition of each stream, all possible events are related in some way. This section describes the supported streams, yang-push and yang-notif-native.

To view the set of streams that are supported, use management protocol operations to retrieve the *streams* table from the Cisco-IOS-XE-mdt-oper-v2 module (from the YANG model Cisco-IOS-XE-mdt-oper-v2.yang) in the *mdt-streams* container.

This example shows how to use NETCONF to retrieve supported streams.

```
<get>
<filter>
<mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
<mdt-streams/>
</mdt-oper-data>
</filter>
</get>

* Enter a NETCONF operation, end with an empty line

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <data>
    <mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
      <mdt-streams>
        <stream>native</stream>
        <stream>yang-notif-native</stream>
        <stream>yang-push</stream>
      </mdt-streams>
    </mdt-oper-data>
  </data>
</rpc-reply>
```

The example shows that three streams are supported: *native*, *yang-notif-native*, and *yang-push*. The stream *native* is not available for general use and can be ignored.

Note

Currently there are no CLIs to return the list of supported streams.

yang-push stream

The yang-push stream is a standard that allows automatic, continuous streaming of updates from a YANG datastore to a client using notifications. The yang-push stream is the data in configuration and operational databases that is described by a supported YANG model. This stream supports an XPath filter to specify what data is of interest within the stream, and where the XPath expression is based on the YANG model that defines the data of interest.

Update notifications are sent either when data changes or during fixed periods, but not for both, for a given subscription. Subscriptions for data that does not currently exist are permitted, and these run as normal subscriptions. The only target database that is supported is *running*.

Telemetry using the *yang-push* stream is based on the IETF NETCONF working group's early drafts for telemetry. These are:

- *Custom Subscription to Event Notifications, Version 03*
- *Subscribing to YANG datastore push updates, Version 07*

These features that are described in the corresponding drafts are not supported.

- Subtree filters
- Out-of-band notifications
- Any subscription parameter not explicitly stated as supported

Determining on-change capability

Currently, there is *no* indication within YANG models about the type of data that can be subscribed to, by using an on-change subscription. Attempts to subscribe to data that cannot be subscribed to by using on-change subscription results in a failure (dynamic)

or an invalid subscription (configured). For more information on On-Change Publication, see the section, *On-Change Publication for yang-push*.

X-Path filter for yang-push

XML Path Language (XPath) is a query and navigation language designed to select and address nodes of an XML document. It uses path-like expressions, similar to filesystem paths, to navigate through the hierarchical structure of an XML document and identify elements, attributes, text, and other node types.

The dataset within the *yang-push* stream to be subscribed to should be specified by the use of an XPath filter. The following guidelines apply to the XPath expression.

- XPath expressions can have keys to specify a single entry in a list or container. The supported key specification syntax is

  ```
  [{key name}={key value}]
  ```

  This is a sample XPath expression:

  ```
  filter xpath /rt:routing-state/routing-instance[name="default"]
  /ribs/rib[name="ipv4-default"]/routes/route # VALID!
  ```

  Compound keys are supported by the use of multiple key specifications. Key names and values must be exact; no ranges or wildcard values are supported.

- In XPath expressions, select multiple keys by specifying them one after another. The following is an example of an XPath expression:

  ```
  /wireless-access-point-oper:access-point-oper-data/radio-oper-data/
  radio-slot-id[wtp-mac="00:11:22:33:44:55"][radio-slot-id="1"]
  ```

- XPath expressions support the use of the union operator (|) to allow a single subscription to support multiple objects. The union operator only works for NETCONF transport and not for gRPC.

Periodic publication for yang-push

Periodic publication of a YANG-push stream refers to a subscription mode where updates from the YANG datastore are sent automatically at regular, configured time intervals, regardless of whether the data has changed. This means the subscriber receives data snapshots periodically, based on a defined time interval.

With periodic subscriptions, the first push-update with the subscribed information is sent immediately; but it can be delayed if the device is busy or due to network congestion. Updates are then sent at the expiry of the configured periodic timer. For example, if the period is configured as 10 minutes, the first update is sent immediately after the subscription is created and every 10 minutes thereafter.

The period is time, in centiseconds (1/100 of a second), between periodic push updates. A period of 1000 will result in getting updates to the subscribed information every 10 seconds. The minimum period that can be configured is 100, or one second. There is no default value. This value must be explicitly set in the establish-subscription RPC for dynamic subscriptions and in the configuration for configured subscriptions.

Periodic updates contain a full copy of the subscribed data element or table for all supported transport protocols.

When subscribing for empty data using a periodic subscription, empty update notifications are sent at the requested period. If data comes into existence, its values at the next period are sent as a normal update notification.

On-change publication for yang-push

In on-change publication, updates are pushed to the subscriber only when the monitored data in the targeted YANG datastore changes.

When creating an on-change subscription, the dampening period must be set to 0 to indicate that there is no dampening period; no other value is supported.

With on-change subscriptions, the first push update is the entire set of subscribed data (the initial synchronization as defined in the IETF documents). This is not controllable. Subsequent updates are sent when the data changes, and consist of only the changed data. However, the minimum data resolution for a change is a row. So, if an on-change subscription is to a leaf within a row, if any item in that row changes, an update notification is sent. The exact contents of the update notification depend on the transport protocol.

In addition, on-change subscriptions are not hierarchical. That is, when subscribing to a container that has child containers, changes in the child container are not seen by the subscription.

Subscriptions for data that does not currently exist are permitted and run as normal subscriptions. The initial synchronization update notification is empty and there are no further updates until data is available.

XPath expressions must specify a single object. That object can be a container, a leaf, a leaf list or a list.

yang-notif-native stream

The yang-notif-native stream is any YANG notification in the publisher where the underlying source of events for the notification uses Cisco IOS XE native technology. This stream also supports an XPath filter that specifies which notifications are of interest. Update notifications for this stream are sent only when events that the notifications are for occur.

Since this stream supports only on-change subscriptions, the dampening interval must be specified with a value of 0.

XPath filter for yang-notif-native

The dataset within the yang-notif-native stream to be subscribed to is specified by the use of an XPath filter. These guidelines apply to the XPath expression.

- XPath expressions must specify an entire YANG notification; attribute filtering is not supported.
- The union operator (|) is not supported.

Transport protocols

Protocols used for connecting a publisher and a receiver decides how the data is sent. This protocol is referred to as the transport protocol, and is independent of the management protocol for configured subscriptions. The transport protocol affects both the encoding of the data, for example XML, Google Protocol Buffers (GPB), and the format of the update notification itself. gNMI, gRPC, and NETCONF are the supported transport protocols.

Note

The stream that is chosen may also affect the format of the update notification.

NETCONF protocol

The NETCONF protocol is available only for the transport of dynamic subscriptions, and can be used with *yang-push* and *yang-notif-native* streams.

When using NETCONF as the transport protocol, three update notification formats are used.

- When the subscription uses the *yang-push* stream, and if it is periodic or when the initial synchronization update notification is sent on an on-change subscription.
- When the subscription uses the *yang-push* stream and it is an on-change subscription, other than the initial synchronization update notification.
- When the subscription uses the *yang-notif-native* stream.

yang-push format

When the *yang-push* source stream is sent over NETCONF as the transport with XML encoding, two update notification formats are defined. These update notification formats are based on the *draft-ietf-netconf-yang-push-07*. For more information, see section 3.7 of the IETF draft.

yang-notif-native format

When the source stream is *yang-notif-native*, the format of the update notification when encoded in XML over NETCONF is as defined by *RFC 7950*. For more information, see section 7.16.2 of the RFC.

Unlike the formats for the *yang-push* stream, the subscription ID is not found in the update notification.

gRPC protocol

The gRPC protocol is available only for the transport of configured subscriptions, and can be used with *yang-push* and *yang-notif-native* streams. Only kvGPB encoding is supported with gRPC transport protocol.

Receiver connection retries based on gRPC protocol (exponential back-off) are supported.

For telemetry messages defined in .proto files, see: *mdt_grpc_dialout.proto* and telemetry.proto.

Mutual authentication for gRPC telemetry

gRPC is one of the supported dial-out protocols used to transmit telemetry data. For dial-out protocols, the device is considered the *client* and the collector, the *server*. gRPC supports both unencrypted TCP and encrypted TLS-based connections.

A new gRPC-TLS profile that contains a pair of trustpoints is added to the telemetry configuration, so that a client ID certificate can be used for mutual authentication. The profile contains two trustpoints, one is the Certificate Authority (CA) certificate for server validation, and the other is the ID certificate for client validation.

When a device connects to a receiver for the first time, based on the server configuration, client or mutual authentication may be required. The device will receive the receiver's identity certificate and validate whether the certificate is signed by the CA identified in the certificate associated with the trustpoint configured in the receiver profile. If the receiver then requests for the certificate ID of the device, the device sends the client ID certificate previously installed in the profile's ID-trustpoint field.

If the server is configured to require mutual authentication, and there is no client ID trustpoint in the profile, the client authentication will not happen, nor will the connection succeed.

The same *trustpoint* label can be configured for multiple profiles, and the same profile can be configured for multiple receivers.

Note

The trustpoint with the client ID is not mandatory in the profile configuration, as mutual authentication is not required for gRPC over TLS, and it can be configured only with server validation.

To add the client ID trustpoint, use the **telemetry protocol grpc profile <name>** command.

Mutual authentication for gRPC telemetry cannot be disabled; but it can be left unused by not configuring the receivers to use the gRPC-TLS protocol, or by removing or not configuring the client ID trustpoint field in the receiver configuration.

This example shows how to configure a gRPC-TLS profile for mutual authentication.

```
Device# configure terminal
Device(config)# telemetry receiver protocol grpc-mtls
Device(config-mdt-protocol-receiver)# host name collector.cisco.com 57500
Device(config-mdt-protocol-receiver)# protocol grpc-tls profile different-ca
Device(config-mdt-protocol-receiver)# exit
```

```
Device(config)# telemetry protocol grpc profile myprofile
Device(config-mdt-protocol-grpc-profile)# exit
Device(config)# telemetry ietf subscription 100
Device(config-mdt-subs)# encoding encode-kvgpb
Device(config-mdt-subs)# filter xpath /memory-ios-xe-oper:memory-statistics/memory-statistic
Device(config-mdt-subs)# receiver-type protocol
Device(config-mdt-subs)# stream yang-push
Device(config-mdt-subs)# update-policy periodic 5000
Device(config-mdt-subs)# receiver name grpc-mtls
Device(config-mdt-subs)# end
Device#
```

Subscription management

Any form of management operation can be used to create, delete, and modify configured subscriptions. This includes both CLIs and network protocol management operations.

All subscriptions, both configured and dynamic, can be displayed using **show** commands and network protocol management operations.

This table describes the supported streams and encodings along with the combinations that are supported.

Table 3: Supported combination of protocols

Transport Protocol	**NETCONF**		**gRPC**		**gNMI**	
	Dial-In	Dial-Out	Dial-In	Dial-Out	Dial-In	Dial-Out
Stream						
yang-push	Yes	No	No	Yes	Yes	No
yang-notif-native	Yes	No	No	Yes	No	No
Encodings	XML	No	No	Key-value Google Protocol Buffers (kvGPB)	• JSON_IETF • PROTO	No

Creating, modifying, and deleting NETCONF subscriptions

You can send and receive YANG XML remote procedure calls (RPCs) in established NETCONF sessions to create, modify, and delete subscriptions.

To establish a new subscription, use the establish-subscription RPC. When an establish-subscription RPC is sent, the reply from a publisher contains an rpc-reply message with a subscription-result element containing a result string.

To terminate a subscription, you can either delete it through delete-subscription RPC or forcibly kill it through kill-subscription RPC. The delete-subscription RPC allows a subscriber to delete a subscription that was previously created by the same subscriber who used the establish-subscription RPC.

Dynamic subscriptions

Dynamic subscriptions enable clients to specify which data paths to monitor and how often updates should be sent, allowing real-time monitoring tailored to current operational needs

The lifetime of a subscription is limited to the lifetime of the connection between the subscriber and the publisher, and telemetry data is sent only to that subscriber. Dynamic subscriptions do not persist if either the publisher or the subscriber is rebooted. Dynamic subscriptions are dial-in subscriptions

Subscribers who connect to a publisher create dynamic subscriptions by using a mechanism within that connection, usually an RPC.

Creating dynamic subscriptions

You can create dynamic subscriptions by using the in-band establish-subscription RPC. This RPC is sent from an IETF telemetry subscriber to the network device. The stream, XPath-filter, and period fields in the RPC are mandatory.

RPCs that are used to create and delete dynamic subscriptions using NETCONF are defined in *Custom Subscription to Event Notifications draft-ietf-netconf-subscribed-notifications-03* and *Subscribing to YANG datastore push updates draft-ietf-netconf-yang-push-07*.

Periodic dynamic subscriptions

A periodic dynamic subscription is a client-initiated subscription that requests telemetry data updates from the network server at regular, configured time intervals. This subscription is dynamic because it is created by the subscriber or client during the session (dial-in) and lasts only as long as the connection between the subscriber and the device or publisher is active.

This is a sample periodic subscription for NETCONF dial-in.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <establish-subscription
          xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"
          xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
        <stream>yp:yang-push</stream>
        <yp:xpath-filter>/mdt-oper:mdt-oper-data/mdt-subscriptions</yp:xpath-filter>
        <yp:period>1000</yp:period>
      </establish-subscription>
    </rpc>
```

On-change dynamic subscriptions

An on-change dynamic subscription refers to a telemetry subscription where updates are sent only when the value of the subscribed data changes, rather than at fixed intervals.

This is a sample on-change dynamic subscription over NETCONF.

```
<establish-subscription xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"
xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <stream>yp:yang-push</stream>
    <yp:xpath-filter>/cdp-ios-xe-oper:cdp-neighbor-details/cdp-neighbor-detail</yp:xpath-filter>
    <yp:dampening-period>0</yp:dampening-period>
</establish-subscription>
```

gNMI dial-in subscriptions

These are sample gNMI dial-in dynamic subscriptions.

```
subscribe: <
  prefix: <>
  subscription: <
    path: <
      origin: "openconfig"
      elem: <name: "routing-policy">
    >
    mode: SAMPLE
    sample_interval: 10000000000
  >
  mode: STREAM
  encoding: JSON_IETF
>'
```

```
subscribe: <
  prefix: <>
  subscription: <
    path: <
      origin: "legacy"
      elem: <name: "oc-platform:components">
      elem: <
        name: "component"
        key: <
          key: "name"
          value: "PowerSupply8/A"
        >
      >
      elem: <name: "power-supply">
      elem: <name: "state">
    >
    mode: SAMPLE
    sample_interval: 10000000000
  >
  mode: STREAM
  encoding: JSON_IETF
>'
```

Receiving a response message

When a subscription is successfully created, the device responds with a subscription result of notif-bis:ok and a subscription ID. This is a sample response RPC message for a dynamic subscription.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
<subscription-result xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"
xmlns:notif-bis="urn:ietf:params:xml:ns:yang:ietf-event-notifications">notif-bis:
ok</subscription-result>
<subscription-id xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications">2147484201</subscription-id>
</rpc-reply>
```

Receiving subscription push updates for NETCONF dial-in

Subscription updates pushed from the device are in the form of an XML RPC and are sent over the same NETCONF session on which these are created. The subscribed information element or tree is returned within the *datastore-contents-xml* tag. This is a sample RPC message that provides the subscribed information.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2017-05-09T21:34:51.74Z</eventTime>
    <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
        <subscription-id>2147483650</subscription-id>
        <datastore-contents-xml>
            <cpu-usage xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-process-cpu-oper"><cpu-utilization>
            <five-minutes>5</five-minutes></cpu-utilization></cpu-usage>
        </datastore-contents-xml>
    </push-update>
</notification>
```

If the information element to which a subscription is made is empty, or if it is dynamic, for example, a named access list that does not exist, the periodic update will be empty and will have a self-closing *datastore-contents-xml* tag. This is a sample RPC message in which the periodic update is empty.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2017-05-09T21:34:09.74Z</eventTime>
    <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
        <subscription-id>2147483649</subscription-id>
        <datastore-contents-xml />
    </push-update>
</notification>
```

Deleting dynamic subscriptions

Dynamic subscriptions cannot be modified but can be terminated at any time. Dynamic subscriptions automatically terminate if the session is terminated. In a dynamic subscription, the subscriber and receiver are always the same entity.

You can delete dynamic subscriptions by using the in-band delete-subscription RPC, the **clear telemetry subscription dynamic** command, and the kill-subscription RPC along with disconnecting the transport session.

In gNMI, for each subscription in the SubscribeRequest.subscribe.subscription a separate dynamic subscription ID is generated. Killing any of these subscription IDs, either through the kill-subscription RPC or the **clear telemetry subscription dynamic** command, will cause all subscriptions specified in the SubscribeRequest to be killed.

The delete-subscription RPC can be issued only by a subscriber, and it deletes only the subscriptions owned by that subscriber. You can also use the **clear telemetry subscription dynamic** command to delete dynamic subscriptions. The kill-subscription RPC deletes dynamic subscriptions in the same way as the **clear telemetry subscription dynamic** command.

In NETCONF, when the parent NETCONF session is torn down or disconnected, subscriptions are deleted. If the network connection is interrupted, it may take some time for the SSH or NETCONF session to timeout, and for subsequent subscriptions to be removed.

The kill-subscription RPC is similar to the delete-subscription RPC. However, the kill-subscription RPC uses the *identifier* element that contains the ID of the subscription to be deleted, instead of the *subscription-id* element used by the delete-subscription RPC. The transport session used by the target subscription also differs from the one used by the delete-subscription RPC.

Deleting subscriptions using the CLI

The output of the **show telemetry ietf subscription all** command displays all the available subscriptions.

```
Device# show telemetry ietf subscription all

 Telemetry subscription brief

  ID               Type        State       Filter type
  ---------------------------------------------------------
  2147483648       Dynamic     Valid       xpath
  2147483649       Dynamic     Valid       xpath
```

This example shows how to delete a dynamic subscription.

```
Device# clear telemetry subscription dynamic 2147483648
```

Deleting subscriptions using NETCONF delete-subscription RPC

This example shows how to delete a subscription using the delete-subscription RPC.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <delete-subscription xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"
    xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
        <subscription-id>2147483650</subscription-id>
    </delete-subscription>
</rpc>
```

Deleting subscriptions using NETCONF kill-subscription RPC

This example shows how to delete subscriptions using the kill-subscription RPC.

```
<get>
<filter>
<mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
<mdt-subscriptions/>
</mdt-oper-data>
</filter>
</get>

* Enter a NETCONF operation, end with an empty line

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <data>
    <mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
      <mdt-subscriptions>
        <subscription-id>2147483652</subscription-id>
        <base>
…
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
…
        </mdt-receivers>
        <last-state-change-time>2018-12-13T21:16:48.848241+00:00</last-state-change-time>
      </mdt-subscriptions>
```

```
      <mdt-subscriptions>
        <subscription-id>2147483653</subscription-id>
        <base>
…
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
…
        </mdt-receivers>
        <last-state-change-time>2018-12-13T21:16:51.319279+00:00</last-state-change-time>
      </mdt-subscriptions>
      <mdt-subscriptions>
        <subscription-id>2147483654</subscription-id>
        <base>
…
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
…
        </mdt-receivers>
        <last-state-change-time>2018-12-13T21:16:55.302809+00:00</last-state-change-time>
      </mdt-subscriptions>
      <mdt-subscriptions>
        <subscription-id>2147483655</subscription-id>
        <base>
…
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
…
        </mdt-receivers>
        <last-state-change-time>2018-12-13T21:16:57.440936+00:00</last-state-change-time>
      </mdt-subscriptions>
    </mdt-oper-data>
  </data>
</rpc-reply>
<kill-subscription xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"
 xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <identifier>2147483653</identifier>
</kill-subscription>

* Enter a NETCONF operation, end with an empty line

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <subscription-result xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"
   xmlns:notif-bis="urn:ietf:params:xml:ns:yang:ietf-event-notifications">notif-bis:ok</subscription-result>
</rpc-reply>
<get>
<filter>
<mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
<mdt-subscriptions/>
</mdt-oper-data>
</filter>
</get>

* Enter a NETCONF operation, end with an empty line
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <data>
    <mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
      <mdt-subscriptions>
        <subscription-id>2147483652</subscription-id>
        <base>
…
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
…
        </mdt-receivers>
        <last-state-change-time>2018-12-13T21:16:48.848241+00:00</last-state-change-time>
      </mdt-subscriptions>
      <mdt-subscriptions>
        <subscription-id>2147483654</subscription-id>
        <base>
…
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
…
        </mdt-receivers>
        <last-state-change-time>2018-12-13T21:16:55.302809+00:00</last-state-change-time>
      </mdt-subscriptions>
      <mdt-subscriptions>
        <subscription-id>2147483655</subscription-id>
        <base>
…
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
…
        </mdt-receivers>
        <last-state-change-time>2018-12-13T21:16:57.440936+00:00</last-state-change-time>
      </mdt-subscriptions>
   </mdt-oper-data>
  </data>
</rpc-reply>
```

Service gNMI

Service gRPC Network Management Interface (gNMI) is a network management protocol developed by Google that uses gRPC (a remote procedure call framework) to manage network devices. It provides a unified service for configuring network devices, retrieving operational state data, and streaming telemetry data in real time.

The gNMI specification identifies a single top-level service named gNMI that contains high-level RPCs. This is a service definition that contains the subscribe-service RPC.

```
service gNMI{
  .
  .
  .
   rpc Subscribe(stream SubscribeRequest)
```

```
       returns (stream SubscribeResponse);
```

SubscribeRequest message

This message is sent by a client to request updates from the target for a specified set of paths. This is a sample SubscribeRequest message definition.

```
message SubscribeRequest {
  oneof request {
    SubscriptionList subscribe = 1;
    PollRequest poll = 3;
    AliasList aliases = 4;
  }
  Repeated gNMI_ext.Extensions = 5;
}
```

Note

Only request.subscribe is supported.

SubscribeResponse message

This message is carried from the target to the client over an established subscribe RPC. This is a sample SubscribeRespose message definition.

```
message SubscribeResponse {
  oneof response {
    Notification update = 1;
    Bool sync_response = 3;
    Error error = 4 [deprecated=true];
  }
}
```

Note

Only Notification update is supported.

gNMI sync_response message

A SubscribeResponse message is transmitted by a target (a switch or a router) to a gNMI client or collector over an established gNMI subscribe RPC. The sync_response is a boolean field that is part of the SubscribeResponse message, which indicates that all data values that corresponds to the paths in the SubscriptionList has been transmitted at least once. The sync_response message is sent after the first update message, and this field is enabled for both gNMI on-change and periodic notifications.

This sample SubscribeResponse message displays the sync_response field.

```
message SubscribeResponse {
  oneof response {
    Notification update = 1;  // Changed or sampled value for a path.
    // Indicate target has sent all values associated with the subscription
    // at least once.
```

```
    bool sync_response = 3;
    // Deprecated in favour of google.golang.org/genproto/googleapis/rpc/status
    Error error = 4 [deprecated = true];
  }
  // Extension messages associated with the SubscribeResponse. See the
  // gNMI extension specification for further definition.
  repeated gnmi_ext.Extension extension = 5;
}
```

SubscriptionList message

This message is used to indicate a set of paths for which common subscription behavior is required. Within the specification of the SubscriptionList message, the client can identify one or more subscriptions to a given prefix in the model. This is a sample SubscriptionList message defintion.

```
message SubscriptionList {
  Path prefix = 1;
   repeated Subscription subscription = 2;
  bool use_aliases = 3;
  QOSMarking qos = 4;
  enum Mode {
      STREAM = 0;
     ONCE = 1;
     POLL = 2;
  }
  Mode mode = 5;
  bool allow_aggregation = 6;
  repeated ModelData use_models = 7;
  Encoding encoding = 8;
  Bool updates_only = 9;

}
```

Note

Path prefix (only explicit element names), Subscription subscription, Mode mode STREAM, and Encoding encoding IETF_JSON are supported.

Prefix message

A valid subscription list may or may not contain a filled in prefix, composed of the shared (across all requested subscriptions) portion of the xPath.

```
message Path {
  repeated string element  = 1; [ deprecated ]
  string origin = 2;
   repeated PathElem elem = 3;
   optional string target = 4;

}
```

Note

“Origin (supported values are “openconfig”, "legacy", and “rfc7951”), elem (supported element name is prefix-free), and target are supported.

"legacy" indicates that YANG module prefixes should be used in the path, and origin "rfc7951", indicates that module name prefixes should be used in the path.

Subscription message

This message generically describes a set of data that is to be subscribed to by a client. It contains a path and attributes used to govern the notification behaviors. This is a sample Subscription message definition.

```
message Subscription {
   Path path = 1;
    SubscriptionMode mode = 2;
    uint64 sample_interval = 3;
   bool suppress_redundant = 4;
   uint64 heartbeat_interval = 5;
}
```

Note

Path path, SubscriptionMode mode, and Uint64 sample_interval are supported.

Path message

A valid subscription contains a filled in path, which when added to the prefix associated with the subscription list constitutes a full qualified path. This is a sample Path message definition.

```
message Path {
  repeated string element  = 1; [ deprecated ]
  string origin = 2;
   repeated PathElem elem = 3;
   optional string target = 4;

}
```

Note

Origin (supported values are “” and “openconfig”), elem (supported element name is prefix-free), and target are supported.

SubscriptionMode message

This message informs the target about how to trigger notifications updates. This is a sample SubscriptionMode message definition:=,

```
enum SubscriptionMode {
   TARGET_DEFINED = 0;
   ON_CHANGE       = 1;
   SAMPLE          = 2;
}
```

Note

Only ON_CHANGE and SAMPLE are supported.

ON_CHANGE support is limited to certain model paths. To check whether a path supports ON_CHANGE, query the path in the Cisco-IOS-XE-MDT-capabilities-oper model. For more information about the model, see the section, Displaying on-change subscription capabilities.

Notifications message

This message delivers telemetry data from the subscription target to the collector. This is a Notifications message definition.

```
message Notification {
  int64 timestamp = 1;
   Path prefix = 2;
  string alias = 3;
  repeated Update update = 4;
  repeated Path delete = 5;
  bool atomic = 6;
}
```

Note

Timestamp, prefix, update, and delete (mainly used for on-change subscriptions) are supported.

Configured subscriptions

Configured subscriptions are telemetry or event subscriptions created by management operations on the publisher by controllers, and explicitly include the specification of the receiver of the telemetry data defined by a subscription. These subscriptions persist across device reboots and transport session interruptions.

Configured subscriptions can be configured with multiple receivers, however; only the first valid receiver is used. Connection to other receivers is not attempted, if a receiver is already connected, or is in the process of being connected. If that receiver is deleted, another receiver is connected.

Configured subscriptions are dial-out subscriptions and these subscriptions are configured on the device by

- using configuration CLIs to change to device configuration through console or VTY.
- using NETCONF or RESTCONF to configure the desired subscription.

  Use either the Cisco-IOS-XE-mdt-cfg.yang model or the ietf-event-notifications.yang model to configure subscriptions.

Creating configured subscriptions

This section describes the sample RPCs to create various types of configured subscriptions.

Periodic subscriptions

This example shows how to configure gRPC as the transport protocol for configured subscriptions using the CLI:

```
telemetry ietf subscription 101
```

```
  encoding encode-kvgpb
  filter xpath /memory-ios-xe-oper:memory-statistics/memory-statistic
  stream yang-push
  update-policy periodic 6000
  source-vrf Mgmt-intf
  receiver ip address 10.28.35.45 57555 protocol grpc-tcp
```

This sample RPC shows how to create a periodic subscription using NETCONF that sends telemetry updates to the receiver every 60 seconds.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><edit-config>
 <target>
  <running/>
 </target>
 <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
   <mdt-subscription>
    <subscription-id>200</subscription-id>
    <base>
     <stream>yang-push</stream>
     <encoding>encode-kvgpb</encoding>
     <period>6000</period>
     <xpath>/memory-ios-xe-oper:memory-statistics/memory-statistic</xpath>
    </base>
    <mdt-receivers>
     <address>10.22.23.48</address>
     <port>57555</port>
     <protocol>grpc-tcp</protocol>
    </mdt-receivers>
   </mdt-subscription>
  </mdt-config-data>
 </config>
</edit-config>
</rpc>
```

This sample RPC creates a periodic subscription using RESTCONF.

```
URI:https://10.85.116.28:443/restconf/data/Cisco-IOS-XE-mdt-cfg:mdt-config-data
Headers:
application/yang-data.collection+json, application/yang-data+json, application/yang-data.errors+json
Content-Type:
application/yang-data+json
BODY:
{
"mdt-config-data": {
 "mdt-subscription":[
 {
  "subscription-id": "102",
  "base": {
   "stream": "yang-push",
   "encoding": "encode-kvgpb",
                    "period": "6000",
   "xpath": "/memory-ios-xe-oper:memory-statistics/memory-statistic"
  }
        "mdt-receivers": {
            "address": "10.22.23.48"
            "port": "57555"
        }
 }
 ]
}
}
```

On-change subscriptions

This sample RPC shows how to create an on-change subscription using NETCONF that sends updates only when there is a change in the target database.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><edit-config>
 <target>
  <running/>
 </target>
 <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
   <mdt-subscription>
    <subscription-id>200</subscription-id>
    <base>
     <stream>yang-push</stream>
     <encoding>encode-kvgpb</encoding>
     <no-synch-on-start-v2>false</no-synch-on-start-v2>
     <xpath>/cdp-ios-xe-oper:cdp-neighbor-details/cdp-neighbor-detail</xpath>
    </base>
    <mdt-receivers>
     <address>10.22.23.48</address>
     <port>57555</port>
     <protocol>grpc-tcp</protocol>
    </mdt-receivers>
   </mdt-subscription>
  </mdt-config-data>
 </config>
</edit-config>
</rpc>
```

The following sample RPC shows how to create an on-change subscription using RESTCONF:

```
URI:
https://10.85.116.28:443/restconf/data/Cisco-IOS-XE-mdt-cfg:mdt-config-data
Headers:
application/yang-data.collection+json, application/yang-data+json, application/yang-data.errors+json
Content-Type:
application/yang-data+json
BODY:
{
"mdt-config-data": {
 "mdt-subscription":[
 {
  "subscription-id": "102",
  "base": {
   "stream": "yang-push",
   "encoding": "encode-kvgpb",
                     "dampening period": "0",
   "xpath": "/cdp-ios-xe-oper:cdp-neighbor-details/cdp
                           -neighbor-detail "
  }
        "mdt-receivers": {
            "address": "10.22.23.48"
            "port": "57555"
        }
 }
 ]
}
}
```

Configuring on-change gRPC subscriptions

Perform this task to configure an on-change gRPC subscription.

Step 1 **enable**

Example:

```
Device> enable
```

Enables privileged EXEC mode.

- Enter your password if prompted.

Step 2 **configure terminal**

Example:

```
Device# configure terminal
```

Enters global configuration mode.

Step 3 **telemetry ietf subscription** *id*

Example:

```
Device(config)# telemetry ietf subscription 8
```

Creates a telemetry subscription and enters telemetry-subscription mode.

Step 4 **stream yang-push**

Example:

```
Device(config-mdt-subs)# stream yang-push
```

Configures a stream for the subscription.

Step 5 **filter xpath** *path*

Example:

```
Device(config-mdt-subs)# filter xpath
/iosxe-oper:ios-oper-db/hwidb-table
```

Specifies the XPath filter for the subscription.

Step 6 **update-policy** {**on-change** | **periodic** *period*}

Example:

```
Device(config-mdt-subs)# update-policy on-change
```

Configures an on-change update policy for the subscription.

Step 7 **encoding encode-kvgpb**

Example:

```
Device(config-mdt-subs)# encoding encode-kvgpb
```

Specifies kvGPB encoding.

Step 8 **receiver ip address** *ip-address receiver-port* **protocol** *protocol* **profile** *name*

Example:

```
Device(config-mdt-subs)# receiver ip address 10.22.22.45 45000 protocol
grpc_tls profile secure_profile
```

Configures the receiver IP address, protocol, and profile for notifications.

Step 9 **end**

Example:

```
Device(config-mdt-subs)# end
```

Exits telemetry-subscription configuration mode and returns to privileged EXEC mode.

Modifying configured subscriptions

You can modify subscriptions by using the

- management protocol configuration operations, such as NETCONF edit-config RPC, or
- CLI (same process as creating a subscription)

Use named receivers to modify subscriptions. For more information, see Named receivers for configured subscriptions, on page 29 section.

If a valid receiver configuration on a valid subscription is in the disconnected state, and the management wants to force a new attempt at setting up the connection to the receiver, it must rewrite the receiver with the exact same characteristics.

Deleting configured subscriptions

You can use the CLI or management operation to delete configured subscriptions. The **no telemetry ietf subscription** command removes the configured subscriptions. Note that configured subscriptions cannot be deleted using the kill-subscription or delete-subscription RPCs; they can only be removed through the configuration interface.

Deleting subscriptions using NETCONF

This sample RPC shows how to delete a configured subscription.

```
<edit-config>
   <target>
     <running/>
   </target>
   <config>
     <mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
       <mdt-subscription  operation="delete">
       <subscription-id>102</subscription-id>
      </mdt-subscription>
     </mdt-config-data>
   </config>
 </edit-config>
```

Deleting subscriptions using the CLI

This example shows how to delete a subscription.

```
Device# configure terminal
Device(config)# no telemetry ietf subscription 101
```

```
Device(config)# end
```

Managing configured subscriptions

Perform this task to manage a configured subscription.

Note

gRPC can be used as a transport for subscriptions, but cannot be used to manage subscriptions.

Step 1 **enable**

Example:

```
Device> enable
```

Enables privileged EXEC mode.

- Enter your password if prompted.

Step 2 **configure terminal**

Example:

```
Device# configure terminal
```

Enters global configuration mode.

Step 3 **telemetry ietf subscription** *id*

Example:

```
Device(config)# telemetry ietf subscription 101
```

Creates a telemetry subscription and enters telemetry-subscription mode.

Step 4 **stream yang-push**

Example:

```
Device(config-mdt-subs)# stream yang-push
```

Configures a stream for the subscription.

Step 5 **filter xpath** *path*

Example:

```
Device(config-mdt-subs)# filter xpath
/memory-ios-xe-oper:memory-statistics/memory-statistic
```

Specifies the XPath filter for the subscription.

Step 6 **update-policy** {**on-change** | **periodic**} *period*

Example:

```
Device(config-mdt-subs)# update-policy periodic 6000
```

Configures a periodic update policy for the subscription.

Step 7 **encoding encode-kvgpb**

Example:

```
Device(config-mdt-subs)# encoding encode-kvgpb
```

Specifies key-value Google Protocol Buffers (kvGPB) encoding.

- kvGPB is a data encoding format that uses GPB technology to represent data as key-value pair. This encoding organizes data as key-value pairs, where each key maps to a corresponding value.

Step 8 **source-vrf** *vrf-id*

Example:

```
Device(config-mdt-subs)# source-address Mgmt-intf
```

Configures the source VRF instance.

Step 9 **source-address** *source-address*

Example:

```
Device(config-mdt-subs)# source-vrf 192.0.2.1
```

Configures the source address.

Step 10 **receiver ip address** *ip-address receiver-port* **protocol** *protocol* **profile** *name*

Example:

```
Device(config-mdt-subs)# receiver ip address 10.28.35.45
57555 protocol grpc-tcp
```

Configures the receiver IP address, protocol, and profile for notifications.

Step 11 **end**

Example:

```
Device(config-mdt-subs)# end
```

Exits telemetry-subscription configuration mode and returns to privileged EXEC mode.

Retrieving subscription details

get RPC

This section describes how to retrieve the list of current subscriptions and how to monitor subscriptions.

To retrieve the list of current subscriptions send a get RPC to the Cisco-IOS-XE-mdt-oper-v2. This is a sample get RPC message.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter>
      <mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
        <mdt-subscriptions/>
      </mdt-oper-data>
    </filter>
  </get>
</rpc>
```

This is a sample RPC reply.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <data>
    <mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
      <mdt-subscriptions>
        <subscription-id>2147485164</subscription-id>
        <base>
          <stream>yang-push</stream>
          <encoding>encode-xml</encoding>
          <period>100</period>
          <xpath>/ios:native/router/ios-rip:rip/ios-rip:version</xpath>
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <updates-in>0</updates-in>
        <updates-dampened>0</updates-dampened>
        <updates-dropped>0</updates-dropped>
      </mdt-subscriptions>
    </mdt-oper-data>
  </data>
</rpc-reply>
```

Subscription details can also be retrieved through a RESTCONF GET request to the Cisco-IOS-XE-mdt-oper-v2 database. This sample RPC shows how to retrieve subscription details using RESTCONF.

```
URI:
https://10.85.116.28:443/restconf/data/Cisco-IOS-XE-mdt-oper-v2: mdt-oper-data/mdt-subscriptions
Headers:
application/yang-data.collection+json, application/yang-data+json, application/yang-data.errors+json
Content-Type:
application/yang-data+json
Returned output:
{
  "Cisco-IOS-XE-mdt-oper-v2:mdt-subscriptions": [
    {
      "subscription-id": 101,
      "base": {
        "stream": "yang-push",
        "encoding": "encode-kvgpb",
        "source-vrf": "",
        "no-synch-on-start-v2": false,
        "xpath": "/iosxe-oper:ios-oper-db/hwidb-table"
      },
      "type": "sub-type-static",
      "state": "sub-state-valid",
      "comments": "",
      "updates-in": "0",
      "updates-dampened": "0",
      "updates-dropped": "0",
      "mdt-receivers": [
        {
          "address": "10.28.35.35",
          "port": 57555,
          "protocol": "grpc-tcp",
          "state": "rcvr-state-connecting",
          "comments": "Connection retries in progress",
          "profile": ""
        }
      ]
```

```
    }
  ]
}
```

show telemetry ietf subscription command

You can also use the **show telemetry ietf subscription** command to display the list of current subscriptions.

This is sample output from the **show telemetry ietf subscription dynamic brief** command.

```
Device# show telemetry ietf subscription dynamic brief

Telemetry subscription brief

  ID               Type        State       Filter type
  ------------------------------------------------------
  2147483667       Dynamic     Valid       xpath
  2147483668       Dynamic     Valid       xpath
  2147483669       Dynamic     Valid       xpath
```

This is sample output from the **show telemetry ietf subscription** *subscription-ID* **detail** command.

```
Device# show telemetry ietf subscription 2147483667 detail

Telemetry subscription detail:

  Subscription ID: 2147483667
  State: Valid
  Stream: yang-push
  Encoding: encode-xml
  Filter:
    Filter type: xpath
    XPath: /mdt-oper:mdt-oper-data/mdt-subscriptions
  Update policy:
    Update Trigger: periodic
    Period: 1000
  Notes:
```

This is sample output from the **show telemetry ietf subscription all detail** command.

```
Device# show telemetry ietf subscription all detail

Telemetry subscription detail:

  Subscription ID: 101
  Type: Configured
  State: Valid
  Stream: yang-push
  Encoding: encode-kvgpb
  Filter:
    Filter type: xpath
    XPath: /iosxe-oper:ios-oper-db/hwidb-table
  Update policy:
    Update Trigger: on-change
    Synch on start: Yes
    Dampening period: 0
  Notes:
```

Subscription monitoring

The process of managing telemetry subscriptions between publishers and subscribers is called subscription monitoring. It involves tracking the status, health, and performance of telemetry data streams. Subscriptions of all types can be monitored by using CLIs and management protocol operations.

Monitoring through models

This is a sample NETCONF message that displays information about telemetry subscriptions.

```
<get>
<filter>
<mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
<mdt-subscriptions/>
</mdt-oper-data>
</filter>
</get>
```

```
* Enter a NETCONF operation, end with an empty line
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <data>
    <mdt-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
      <mdt-subscriptions>
        <subscription-id>101</subscription-id>
        <base>
          <stream>yang-push</stream>
          <encoding>encode-kvgpb</encoding>
          <source-vrf>RED</source-vrf>
          <period>10000</period>
          <xpath>/ios:native/interface/Loopback[name="1"]</xpath>
        </base>
        <type>sub-type-static</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
          <address>5.22.22.45</address>
          <port>57500</port>
          <protocol>grpc-tcp</protocol>
          <state>rcvr-state-connecting</state>
          <comments/>
          <profile/>
          <last-state-change-time>1970-01-01T00:00:00+00:00</last-state-change-time>
        </mdt-receivers>
        <last-state-change-time>1970-01-01T00:00:00+00:00</last-state-change-time>
      </mdt-subscriptions>
      <mdt-subscriptions>
        <subscription-id>2147483648</subscription-id>
        <base>
          <stream>yang-push</stream>
          <encoding>encode-xml</encoding>
          <source-vrf/>
          <period>1000</period>
          <xpath>/if:interfaces-state/interface[name="GigabitEthernet0/0"]/oper-status</xpath>
        </base>
        <type>sub-type-dynamic</type>
        <state>sub-state-valid</state>
        <comments/>
        <mdt-receivers>
          <address>5.22.22.45</address>
          <port>51259</port>
```

```
          <protocol>netconf</protocol>
          <state>rcvr-state-connected</state>
          <comments/>
          <profile/>
          <last-state-change-time>1970-01-01T00:00:00+00:00</last-state-change-time>
        </mdt-receivers>
        <last-state-change-time>1970-01-01T00:00:00+00:00</last-state-change-time>
      </mdt-subscriptions>
    </mdt-oper-data>
  </data>
</rpc-reply>
```

Monitoring through CLIs

Use the **show telemetry ietf subscription** command to display information about telemetry subscriptions. This is sample output from the command.

```
Device# show telemetry ietf subscription 2147483667 detail

Telemetry subscription detail:

  Subscription ID: 2147483667
  State: Valid
  Stream: yang-push
  Encoding: encode-xml
  Filter:
    Filter type: xpath
    XPath: /mdt-oper:mdt-oper-data/mdt-subscriptions
  Update policy:
    Update Trigger: periodic
    Period: 1000
  Notes:
```

Named receivers for configured subscriptions

With fully qualified domain name (FQDN) support, a new method of configuring receivers is introduced, called the named-receiver configuration. Named receivers are top-level configuration entities that can exist independent of subscriptions. Named receivers are identified by a name. The name is an arbitrary string, and is the index or key of the named receiver records in the system. The named receiver configuration contains all configurations associated with the receiver that is not subscription-dependent.

Named receivers can

- support different types of receivers,
- provide better state and diagnostics information,
- use a host name instead of an IP address to specify the host for protocol receivers, and
- change the parameters of a receiver that is used by multiple subscriptions at a single place.

Only protocol-type named receivers are supported, and these are

- grpc-tcp: gRPC TCP protocol
- grpc-tls: gRPC TLS protocol

Displaying named receiver state using YANG models

The state of the named receivers can be retrieved using the Cisco-IOS-XE-mdt-oper-v2 YANG model. The mdt-oper-v2-data container contains an mdt-named-receivers list that contains the operational state of all named receivers.

This is a sample NETCONF reply to retrieve the state of named receivers.

```
<get>
 <filter>
  <mdt-oper-v2-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
   <mdt-named-receivers/>
  </mdt-oper-v2-data>
 </filter>
</get>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
 <data>
  <mdt-oper-v2-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-oper-v2">
   <mdt-named-receivers>
    <name>my-receiver</name>
    <profile>tls-profile</profile>
    <params>
     <protocol>grpc-tls</protocol>
<host>
      <hostname>rcvr.test.com </hostname>
     </host>
     <port>45000</port>
    </params>
    <state>named-rcvr-state-valid</state>
    <last-state-change-time>2020-…:00</last-state-change-time>
   </mdt-named-receivers>
  </mdt-oper-v2-data>
 </data>
</rpc-reply>
```

Displaying named receiver state using the CLI

To view the state of named receivers of all types, use the **show telemetry receiver** command. The **all** keyword displays information about all named receivers in a brief format, and the **name** keyword displays detailed information about the specified named receiver.

This is sample output from the **show telemetry receiver all** command.

```
Device# show telemetry receiver all

Telemetry receivers

  Name       <…>     Type        Profile             State     Explanation
  -----------<…>-------------------------------------------------------
  receiver1  <…>     protocol    tls-trustpoint      Valid
```

This is sample output from the **show telemetry receiver name** command.

```
Device# show telemetry receiver name my-receiver

  Name: my-receiver
  Profile: tls-profile
  State: Valid
```

```
  State Description:
  Last State Change: 09/18/24 15:50:54
  Type: protocol
  Protocol: grpc-tls
  Host: collector.cisco.com
```

Named protocol receivers

Named protocol receivers specify the telemetry transports that use protocols. Named protocol receivers refer to explicitly-defined, reusable receiver entities that represent destinations for pushed notifications or telemetry data. In addition to the name that identifies a receiver, named protocol receivers also use a host specification. The host specification takes a hostname or IP address, and a destination port number. Instead of specifying the receiver connection details directly within each subscription, named protocol receivers allow these details to be defined once and referenced by name in multiple subscriptions. Secure protocol transports also use a profile string.

Note

When a valid named protocol receiver is created, it is not automatically connected to the receiver. The named protocol receiver must be requested by at least one subscription to create a connection to the receiver.

Configuring named protocol receivers using YANG models

You can configure named protocol receivers through the CLI or YANG models. The YANG model, Cisco-IOS-XE-mdt-cfg, contains the named protocol receiver. The container mdt-named-protocol-rcvrs inside the top level mdt-config-data container has a list of mdt-named-protocol-rcvr structures.

This group has five members:

- hostname
- name, which is the list key
- port number
- protocol
- profile

This is a sample NETCONF RPC that shows how to create a named protocol receiver.

```
<edit-config>
 <target>
  <running/>
 </target>
 <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
   <mdt-named-protocol-rcvrs>
    <mdt-named-protocol-rcvr>
     <name>my-receiver</name>
     <protocol>grpc-tls</protocol>
     <profile>tls-profile</profile>
     <host>
      <hostname>collector.cisco.com</hostname>
     </host>
     <port>57500</port>
    </mdt-named-protocol-rcvr>
   </mdt-named-protocol-rcvrs>
```

```
  </mdt-config-data>
 </config>
</edit-config>
```

Configuring named protocol receivers

Perform this task to configure a named protocol receiver.

Step 1 **enable**

Example:

```
Device> enable
```

Enables privileged EXEC mode.

- Enter your password if prompted.

Step 2 **configure terminal**

Example:

```
Device# configure terminal
```

Enters global configuration mode.

Step 3 **telemetry receiver protocol** *receiver-name*

Example:

```
Device(config)# telemetry receiver protocol receiver1
```

Configures a named protocol receiver, and enters telemetry protocol-receiver configuration mode.

Step 4 **protocol** {**cloud-native** | **cntp-tcp** | **cntp-tls profile** *profile-name* | **grpc-tcp** | **grpc-tls profile** *profile-name* | **native** | **tls-native profile** *profile-name*}

Example:

```
Device(config-mdt-protocol-receiver)# protocol grpc-tcp
```

Configures a protocol for the named protocol receiver connection.

Step 5 **host** {**ip** *ip-address* | **name** *hostname*} *receiver-port*

Example:

```
Device(config-mdt-protocol-receiver)# host name rcvr.test.com 45000
```

Configures the name protocol receiver hostname.

Step 6 **end**

Example:

```
Device(config-mdt-protocol-receiver)# end
```

Exits telemetry protocol-receiver configuration mode and returns to privileged EXEC mode.

Named-receiver subscriptions

To use a named receiver with a subscription, both the receiver type and receiver name must be specified. No additional receiver configuration is required, since all receiver-specific information is part of the named receiver configuration. However; named protocol

receivers still use the source virtual routing and forwarding (VRF) instance and source address of the subscriptions as part of the connection resolution process.

The only supported named receiver type is *protocol*.

Subscriptions can use either named receivers or legacy receivers, but cannot use both. If the legacy receiver is configured, setting the subscription receiver type and a named-receiver name is blocked. Similarly, if a subscription receiver type or a named receiver is specified, you cannot configure legacy receivers.

Note

Subscriptions use only one receiver, even if more than one receiver is configured.

Subscriptions using legacy receivers and subscriptions using named receivers are permitted to use the same connection; however, it is not recommended.

Named receiver operation and operational state

Named receiver objects and subscription receiver objects (that refer to the named receiver) have two different operational states. The operational states can be valid or invalid. The most common reason for a named receiver to be invalid is incomplete configuration; however, it could also be due to other reasons. The operational state view of a named receiver has a field that provides a text explanation on why the receiver is invalid. When the receiver state is valid, this field is empty.

Configuring named-receiver subscriptions configuration using YANG models

When named receivers are used, the only value supported for rcvr-type is rcvr-type-protocol, and when legacy receivers are used, the value is the default rcvr-type-unspecified.

This is a sample NETCONF RPC that shows how to create a subscription using a named protocol-receiver.

```
<edit-config>
 <target>
  <running/>
 </target>
 <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
   <mdt-subscription>
    <subscription-id>1</subscription-id>
    <base>
     <rcvr-type>rcvr-type-protocol</rcvr-type>
    </base>
    <mdt-receiver-names>
     <mdt-receiver-name>
      <name>receiver1</name>
     </mdt-receiver-name>
    </mdt-receiver-names>
   </mdt-subscription>
  </mdt-config-data>
 </config>
</edit-config>
```

Configuring named-receiver subscriptions using the CLI

Perform this task to configure named receiver subscriptions,

Step 1 **enable**

Example:

```
Device> enable
```

Enables privileged EXEC mode.

- Enter your password if prompted.

Step 2 **configure terminal**

Example:

```
Device# configure terminal
```

Enters global configuration mode.

Step 3 **telemetry ietf subscription** *id*

Example:

```
Device(config)# telemetry ietf subscription 101
```

Creates a telemetry subscription and enters telemetry-subscription mode.

Step 4 **receiver-type protocol**

Example:

```
Device(config-mdt-subs)# receiver-type protocol
```

Configures a protocol-type receiver.

Step 5 **receiver name** *name*

Example:

```
Device(config-mdt-subs)# receiver name receiver1
```

Configures a name for the receiver for notifications.

Step 6 **end**

Example:

```
Device(config-mdt-subs)# end
```

Exits telemetry telemetry-subscription mode and returns to privileged EXEC mode.

Troubleshooting named receiver connections

When a subscription is set up, one of the common problems is that no telemetry update messages are received. Possible reasons could be that there are no events to send, or the subscription is not valid. This section describes how to troubleshoot some of the common problems that occur in named receiver connections.

The logs from the telemetry process, and the output of some of the **show** commands provide information that can be used for troubleshooting the named receiver configuration.

Table 4: Troubleshooting named receiver connections

Problem	How to Check/ Symptom	What to do
Subscription is not valid.	**show telemetry ietf subscription** *id* **details**	Fix the subscription configuration.

Problem	How to Check/ Symptom	What to do
Subscription receiver is not valid.	**show telemetry ietf subscription** *id* **receiver**	Fix the named receiver configuration.
Subscription receiver's connection parameters cannot be resolved.	**show telemetry ietf subscription** *id* **receiver** Subscription receiver state appears to never leave the resolving state.	Verify the receiver, the network configuration, or the interface state.
Subscription receiver connection does not come up.	**show telemetry ietf subscription** *id* **receiver** Subscription receiver state constantly changes from resolving to connecting.	Verify that the resolved connection is valid, and the receiver or collector is reachable and able to accept inbound connections using the specified transport.
Subscription receiver connections are rejected.	**show telemetry ietf subscription** *id* **receiver** Subscription receiver state constantly changes through all states except disconnected.	Verify that the collector is of the correct type, and that the configured authentication and authorization is valid.
Subscription receiver is connected, but no updates are received.	**show telemetry internal subscription** *id* **stats** Message drop count is incrementing, but the records sent is not.	Verify that the collector is able to keep up with the flow of update notifications.
	show telemetry internal subscription **No change in the count.**	If the subscription is on-change, ensure that there really have been no events. If the subscription is periodic, ensure that the update period is small, that the time is specified in hundredths of a second.

This section provides more details about the commands that provide internal information to troubleshoot issues.

show telemetry connection: This command takes an optional connection index value. When no index is specified, it displays the basic connection parameter information for all connections that are being used. When a connection index is specified in the command, it shows low-level details about the connection. The command output is transport-specific, and might not be available for all transports. The output from this command is subject to change.

show telemetry internal diagnostics: This command attempts to dump all telemetry logs and operational state. When reporting problems, it may be helpful to use this command as close to the problem time as possible and provide the output of the **show running-config | section telemetry** command as well.

Subscription receivers

Subscription receivers are the subscription-related objects that connect to the actual subscription receiver or collector. While the mechanism needed to reach the collector is specific to the receiver type, a connection is the entity that is used to allow the subscription to reach its receiver or collector.

Subscription receiver state is based on its ability to request and use the connection to the receiver and has a number of states that are associated with the control of other resources required to allow the subscription to send updates to the receiver or collector.

Subscription receivers use connections through telemetry or telemetry protocols

Subscription receiver states

The operational state of a subscription receiver consists of the configured name (that is the index of the connection), the state of the receiver, an explanation or note about the state, and the time of the last state change. The explanation string is not always used.

The possible states of a subscription receiver are shown in this table.

Table 5: Subscription receiver states

Subscription receiver state		**Description**
CLI value	**YANG value**	
Disconnected	rcvr-state-disconnected	The receiver is disconnected and no attempt is made to reconnect it.
Resolving	rcvr-state-resolving	Resolving the connection parameters required to reach the receiver.
Transport requested	rcvr-state-transport-requested	A request for a connection to reach the receiver was using the connection parameters determined from the resolving state.
Connecting	rcvr-state-connecting	Resources needed to connect the subscription to the receiver are being allocated.
Connected	rcvr-state-connected	The subscription is connected to the receiver, and updates can flow to the receiver.
Disconnecting	rcvr-state-disconnecting	Resources used on the connection are being re-allocated.

The YANG value rcvr-state-invalid is used only by legacy receivers. Subscription receivers that are invalid cannot be connected, so the subscription receiver state is set to disconnected when it is invalid. The explanation string provides the distinction between invalid subscription receivers and disconnected subscription receivers.

A subscription receiver may be disconnected due to these reasons:

- Another receiver on the subscription is not disconnected.
- Connection setup failed permanently.
- Named receiver does not exist.
- Named receiver is not the type specified in the subscription.
- Named receiver is not valid.
- Subscription is invalid.
- The requested connection is in use by a different receiver.

Telemetry connections

Telemetry connections represent the transport instances used by subscriptions to reach the receivers and are purely operational. Telemetry connections are identified by an integer index value. Other information about the connections is specific to the type of connection, which is based on the type of receiver that the subscription is configured to use.

For the secure dial-out transports, the host part of the configured named receiver must match the distinguished name (DN) of the certificate provided by the receiver, when the connection is set up. For this reason, it is not permitted to have more than one receiver using the same connection.

While all the states discussed in this section are available to all types of connections, not all have to be used.

The possible states of a telemetry connection are shown in this table.

Telemetry connection states

Table 6: Connection states and description

Connection state		**Description**
CLI value	**YANG value**	
Pending	con-state-pending	The connection has been created, but not yet initiated.
Connecting	con-state-connecting	A request to set up the connection is in progress.
Active	con-state-active	The connection is up and is available for use by subscription receivers.
Disconnecting	con-state -disconnecting	The connection has been torn down and is waiting to be released by subscription receivers.

Additional operational state associated with a connection includes the identity of the remote receiver (the peer, when available), and the time of the last state change.

Telemetry protocol connections

Telemetry protocol connections are the communication links and mechanisms established between telemetry data sources and telemetry collectors or management systems to transport telemetry data. These connections use specific protocols to deliver network performance, state, and event data for monitoring and analysis.

This section discusses protocol type connections and how these are used by subscription receivers that are assigned to named protocol receivers.

This table displays the protocol-connection parameters.

Table 7: Parameters of a protocol-type connection

Parameter	**Origin**	**Comments**
Destination IP address	Named receiver host	Because hosts use domain names, domain name resolution may be required.
Destination port number	Named receiver port	Must be explicitly configured.

Parameter	Origin	Comments
Source VRF	Subscription, if specified	Default VRF is used, if not specified. Otherwise the VRF name is resolved to an internal identifier.
Source IP address	Subscription, if specified	If not specified, the source IP address is determined based on the VRF and destination IP address.

Some of these parameters are based on the configuration of the subscription receiver's parent subscription.

When resolving the connection parameters from the configuration, the VRF is determined first, followed by the destination IP address, and finally the source IP address, if an order is not specified. If a given step in the resolution fails non-permanently, there are infinite retries at 5 second intervals.

A connection is instantiated as soon as it is requested. That is, as soon as the first subscription receiver goes from the resolving state to the transport requested state, a connection instance with the parameters that were resolved previously by the subscription receiver is created.

If the requested connection is successfully setup and used by telemetry, the connection state changes to connected, which means that a connection exists between the Cisco IOS XE device and the receiver device. To reallocate the resources used by the receiver, the subscription receivers that want to use these resources are informed that the connection is set up. These subscription receivers then transition to the connecting state to set up the resources required to connect the subscription to the receiver. Once these resources are in place, the subscription receiver's state changes to connected, and update notifications are received by the receiver.

These are some of the reasons why a telemetry connection cannot become active:

- authentication failures,
- destination unreachable,
- listener at the remote host port is of the wrong type, and
- no listener at the remote host port.

Note

When a connection setup is in progress, any subscription receiver using this connection will be in the connecting state, because it has successfully resolved the parameters needed to initiate the connection setup.

The action taken when a connection setup fails is specific to the protocol. This table shows the retry behaviors for connections within a single setup request and for re-resolution requests when the connection setup request fails. This behavior is the same for connections requested by the legacy receivers as well.

Retry intervals for protocols

Table 8: Protocol-specific retry intervals

Protocol	Connection retries	Re-resolution requests
• grpc-tcp • grpc-tls	5 retries at 1, 3, 4, and 7 seconds in between each try	No limit; continuously requests re-resolution when connection retries fail. 14 seconds per try.

Protocol	Connection retries	Re-resolution requests
• cloud-native • cntp-tcp • cntp-tls • native • tls-native		5, 10, 15, 20, 25, and 30 seconds.

High Availability in telemetry

Dynamic telemetry connections are established over a NETCONF session through SSH to the active switch or a member in a switch stack, or the active route processor in a high-availability-capable device. You must also re-create all the dynamic subscriptions after a switchover. gNMI dial-in subscriptions also work the same as a NETCONF session through SSH.

gRPC dial-out subscriptions are configured on the device as part of the running configuration of the active switch or member of the stack. When switchover occurs, the existing connections to the telemetry receivers are torn down and reconnected (as long as there is still a route to the receiver). Subscriptions need not be reconfigured.

In the event of a device reload, subscription configurations must be synchronized to the start-up configuration of a device. This ensures that after the device reboots, subscription configurations remain intact on the device. When the necessary processes are up and running, the device attempts to connect to the telemetry receiver and resume normal operations.

FQDN support for gRPC subscriptions

gRPC telemetry subscriptions are configuration-based, which means that users must specify the receiving host and other subscription parameters as part of the device configuration. This receiver configuration is used to determine the connection details for sending telemetry updates. With FQDN support, along with IP addresses, Fully Qualified Domain Names (FQDNs) can also be used for gRPC subscriptions. FQDN support is enabled by default.

To use FQDN, a subscription must use a named receiver, as described earlier. This example shows an FQDN used for gRPC subscriptions.

```
telemetry receiver protocol my-receiver
 host name receiver.cisco.com 12345
 protocol grpc-tcp

telemetry ietf subscription 101

 encoding encode-kvgpb
 filter xpath /mdt-oper-v2-data/mdt-subscriptions

 stream yang-push
 update-policy periodic 1000

 receiver name my-receiver

telemetry ietf subscription 102

 encoding encode-kvgpb
 filter xpath /mdt-oper-v2-data/mdt-connections
```

```
stream yang-push
update-policy periodic 1000

receiver name my-receiver
```