Model-Driven Telemetry

Model-Driven Telemetry

Model-driven telemetry provides a mechanism to stream YANG-modelled data to a data collector. This module describes model-driven telemetry and provides sample telemetry remote procedure calls (RPCs).

Prerequisites for 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

Device Management Interface 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.
gnmib	GNMI protocol server.

NETCONF-Specific Prerequisites

- Knowledge of NETCONF and how to use it, including:
  - Establishing a NETCONF session.
  - Sending and receiving hello and capabilities messages.
  - Sending and receiving YANG XML RPCs over the established NETCONF session. For more information, see the *Configure NETCONF/YANG and Validate Example for Cisco IOS XE 16.x Platforms* document.

Enabling and Validating NETCONF

The NETCONF functionality can be verified by creating an SSH connection to the device using a valid username and password and receiving a hello message, which contains the capability of the device:

```
Device:~ USER1$ ssh -s cisco1@172.16.167.175 -p 830 netconf
cisco1@172.16.167.175's password: cisco1

<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability
.
.
.
</capabilities>
<session-id>2870</session-id></hello>]]>]]>

Use < ^C > to exit
```

NETCONF is ready to use, when a successful reply is received in response to your hello message.

RESTCONF-Specific Prerequisites

- Knowledge of RESTCONF and how to use it (when creating a subscription using RESTCONF).
- RESTCONF must be configured on the device.
- RESTCONF must send correctly-formed Uniform Resource Identifiers (URIs) that adhere to RESTCONF *RFC 8040*.

Enabling and Validating RESTCONF

Validate RESTCONF using appropriate credentials and the following URI:

```
Operation: GET
Headers:
" Accept: application/yang-data.collection+json, application/yang-data+json,
application/yang-data.errors+json
" Content-Type: application/yang-data+json
Returned Output (omitted for breverity):
{
    "ietf-restconf:data": {
        "ietf-yang-library:modules-state": {
            "module": [
                {
                    "name": "ATM-FORUM-TC-MIB",
                    "revision": "",
                    "schema":
"https://10.85.116.28:443/restconf/tailf/modules/ATM-FORUM-TC-MIB",
                    "namespace": "urn:ietf:params:xml:ns:yang:smiv2:ATM-FORUM-TC-MIB"
                },
                {
                    "name": "ATM-MIB",
                    "revision": "1998-10-19",
                    "schema":
"https://10.85.116.28:443/restconf/tailf/modules/ATM-MIB/1998-10-19",
                    "namespace": "urn:ietf:params:xml:ns:yang:smiv2:ATM-MIB"
                },
                {
                    "name": "ATM-TC-MIB",
```

```
                    "revision": "1998-10-19",
                    "schema": "https://10.85.116.28:443/restconf/tailf/
..
<snip>
..
}
```

RESTCONF is validated successfully when you receive the above reply with all device capabilities.

gRPC-Specific Prerequisites

- Set up a gRPC collector that understands key-value Google Protocol Buffers (GPB) encoding.

Restrictions for 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:

  - Cisco-IOS-XE-wireless-access-point-oper
  - Cisco-IOS-XE-wireless-ap-global-oper
  - Cisco-IOS-XE-wireless-awips-oper
  - Cisco-IOS-XE-wireless-client-global-oper
  - Cisco-IOS-XE-wireless-client-oper
  - Cisco-IOS-XE-wireless-general-cfg
  - Cisco-IOS-XE-wireless-general-oper
  - Cisco-IOS-XE-wireless-mesh-cfg
  - Cisco-IOS-XE-wireless-mesh-oper
  - Cisco-IOS-XE-wireless-mobility-oper
  - Cisco-IOS-XE-wireless-rfid-oper
  - Cisco-IOS-XE-wireless-rrm-emul-oper
  - Cisco-IOS-XE-wireless-rrm-global-oper
  - Cisco-IOS-XE-wireless-rrm-oper
  - Cisco-IOS-XE-wireless-site-cfg
  - bootcamp-test-autonomous
  - 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.

gRPC-Specific Restrictions

- Transport Layer Security-based (TLS-based) authentication between a device and receiver is not supported.

  TLS-based authentication is supported in Cisco IOS XE Amsterdam 17.1.1 and later releases.

yang-push-Specific Restriction

- Subscription quality of service (QoS) is not supported.

Information About Model-Driven Telemetry

The following sections provide information about the various aspects of model-driven telemetry.

Model-Driven Telemetry Overview

Telemetry is an automated communications process by which measurements and other data are collected at remote or inaccessible points and transmitted to the receiving equipment for monitoring. Model-driven telemetry provides a mechanism to stream YANG-modeled data to a data collector.

Applications can subscribe to specific data items they need, by using standards-based YANG data models over NETCONF, RESTCONF, or gRPC Network Management Interface (gNMI) protocols. Subscriptions can also be created by using CLIs if it is a configured subscription.

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

Telemetry Roles

In systems that use telemetry, different roles are involved. In this document the following telemetry roles are described:

- Publisher: Network element that sends the telemetry data.
- Receiver: Receives the telemetry data. This is 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. This is also called the management agent or management entity.
- Subscriber: Network element that creates subscriptions. Technically, while this does not have to be the receiver too, in this document, both are the same.

Subscription Overview

Subscriptions are items that create associations between telemetry roles, and define the data that is sent between them.

Specifically, a subscription is used to define the set of data that is requested as part of the telemetry data; when the data is required, how the data is to be formatted, and, when not implicit, who (which receivers) should receive the data.

Even though the maximum number of supported subscriptions is platform-dependent, currently 100 subscriptions are supported. The subscriptions can be either configured or dynamic, and use any combination of transport protocols. If too many subscriptions are operating at the same time to allow all the valid configured subscriptions to be active, the removal of an active subscription will cause one of the inactive but valid configured subscriptions to be attempted. Periodic triggered subscriptions (100 centiseconds is the default minimum) and on-change triggered subscriptions are supported.

NETCONF and other northbound programmable interfaces (such as RESTCONF or gNMI) are supported to configure subscriptions.

Two types of subscriptions are used in telemetry on Cisco IOS XE systems: dynamic and configured subscriptions.

Because dynamic subscriptions are created by clients (the subscriber) that connect into the publisher, they are considered dial-in. Configured subscriptions cause the publisher to initiate connections to receivers, and as a result, they are considered dial-out.

Dial-In and Dial-Out Model-Driven Telemetry

The two flavors of model-driven telemetry are, dial-in and dial-out.

Table 2: Dial-in and Dial-Out Model-Driven Telemetry

Dial-In (Dynamic)	Dial-Out (Static or Configured)
Telemetry updates are sent to the initiator or subscriber.	Telemetry updates are sent to the specified receiver or collector.
Life of the subscription is tied to the connection (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 reinitiated after a reload, because established connections or sessions are killed during 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 successful establishment of a subscription.	Subscription ID is fixed and configured on the device as part of the configuration.

Data Source Specifications

Sources of telemetry data in a subscription are specified by the use of a stream and a filter. The term 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 set of events from a stream are filtered. Different filter types are used for different stream types.

Cisco IOS XE 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 is stream-dependent. Some streams support making data available only when there a change happens, or after an event within the stream. Other streams make data available when there is a change or at a defined time period.

The result of the *when* specification is a series of update notifications that carry the telemetry data of interest. How the data is sent is dependent 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 is set by the controller, and for dynamic subscriptions is set by the publisher.

Controllers must limit the values they use for configured subscriptions in the range 0 to 2147483647 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.

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.

The following table describes the supported streams and encodings along with the combinations that are supported. While streams-as-inputs is intended to be independent of the protocols-as-outputs, not all combinations 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	No

RPC Support in Telemetry

You can send and receive YANG XML remote procedure calls (RPCs) in established NETCONF sessions.

The <establish-subscription> and <delete-subscription> RPCs are supported for telemetry.

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

The following table displays the response and reason for the response in an <rpc-reply> message:

Result String	RPC	Cause
ok	<establish-subscription> <delete-subscription>	Success
error-no-such-subscription	<delete-subscription>	The specified subscription does not exist.
error-no-such-option	<establish-subscription>	The requested subscription is not supported.
error-insufficient-resources	<establish-subscription>	A subscription cannot be created because of the following reasons: • There are too many subscriptions. • The amount of data requested is too large. • The interval for a periodic subscription is too small.
error-other	<establish-subscription>	Some other error.

Service gNMI

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

```
service gNMI{
  .
  .
  .
   rpc Subscribe(stream SubscribeRequest)
       returns (stream SubscribeResponse);
```

The <subscribe RPC> is used by a management agent to request a dynamic subscription. This RPC contains a set of messages. The following section describes the messages supported by the <subscribe RPC>

SubscribeRequest Message

This message is sent by a client to request updates from the target for a specified set of paths. The following is a 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>. The following is a message definition:

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

Note Only Notification update is supported.

SubscriptionList Message

This message is used to indicate a set of paths for which common subscription behavior are required. Within the specification of the SubscriptionList message, the client can identify one or more subscriptions to a given prefix in the model. The following is a 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; // only JSON_IETF supported in R16.12
  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 “” and “openconfig”), elem (supported element name is prefix-free), and target are supported.

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. The following is a 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, Uint64 sample_interval, and Uint64 heartbeat_interval (only if the value is set to 0) 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. The following is a 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. The following is a SubscriptionMode message definition:

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

Note Only SAMPLE and ON_CHANGE (from Cisco IOS XE Bengaluru 17.6.1) 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 YANG Models, on page 28.

Notifications Message

This message delivers telemetry data from the subscription target to the collector. The following 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, and update are supported.

Dynamic Subscription Management

This section describes how to create and delete dynamic subscriptions.

Creating Dynamic Subscriptions for NETCONF Dial-In

Dynamic subscriptions are created by subscribers who connect to the publisher and call for subscription creation using a mechanism within that connection, usually, an RPC. The lifetime of the subscription is limited to the lifetime of the connection between the subscriber and the publisher, and telemetry data is sent only to that subscriber. These subscriptions do not persist if either the publisher or the subscriber is rebooted. You can create dynamic subscriptions by using the in-band <establish-subscription> RPC. The

<establish-subscription> 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

The following 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 Subscription

The following 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>
```

Deleting Dynamic Subscriptions

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

For gNMI 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 clear CLI, will cause all subscriptions specified in the subscribe request to be killed.

Introduced in Cisco IOS XE Gibraltar 16.10.1, the <delete-subscription> RPC can be issued only by a subscriber, and it deletes only the subscriptions owned by that subscriber.

In Cisco IOS XE Gibraltar 16.11.1 and later releases, you can use the **clear telemetry ietf subscription** command to delete a dynamic subscription. Introduced in Cisco IOS XE Gibraltar 16.11.1, the <kill-subscription> RPC deletes dynamic subscription, the same way as the **clear telemetry ietf subscription** command.

A subscription is also deleted when the parent NETCONF session is torn down or disconnected. 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. 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 following sample output shows 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
```

The following example shows how to delete a dynamic subscription:

```
Device# clear telemetry ietf subscription 2147483648
```

Deleting Subscriptions Using NETCONF <delete-Subscription> RPC

The following example shows how to delete a subscription using NETCONF:

```
<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

The following examples show 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">
<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">
      <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">
```

```
<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">
      <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>
```

Configured Subscription Management

This section describes how to create, modifiy, and delete configured subscriptions.

Creating Configured Subscriptions

Configured subscriptions are 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 reboots of the publisher.

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 dial-out subscriptions are configured on the device by the following methods:

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

This section displays sample RPCs to create configured subscriptions.

Periodic Subscription

The following 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
```

The following 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>
```

The following 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 Subscription

The following 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>false</no-synch-on-start>
     <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"
        }
 }
 ]
}
}
```

gNMI Dial-In Subscription

The following is a sample gNMI dial-in subscription:

```
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
```

```
>'
```

Modifying Configured Subscriptions

There are two ways to modify configured subscriptions:

- Management protocol configuration operations, such as NETCONF <edit-config> RPC
- CLI (same process as creating a subscription)

Subscription receivers are identified by the address and port number. Receivers cannot be modified. To change the characteristics (protocol, profile, and so on) of a receiver, it must be deleted first and a new receiver created.

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 RPCs, only through the configuration interface.

Deleting Subscriptions Using the CLI

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

Deleting Subscriptions Using NETCONF

The following 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>
```

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 the introduction of the FQDN Support for gRPC Subscriptions feature, along with IP addresses, Fully Qualified Domain Names (FQDNs) can also be used for gRPC subscriptions.

In a telemetry subscription, receiver details can now be specified either as part of the subscription, or they can be configured independently; where the receiver has a name and this name is used to specify the receiver when configuring the subscription. In both the cases, it is possible to specify the same receiver name for multiple subscriptions.

This feature cannot be disabled.

Named Receivers

With 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.

The advantages of using named receivers are as follows:

- Capable of supporting different types of receivers.
- Better state and diagnostics information.
- Hostname can be used instead of an IP address to specify the host for protocol receivers.
- Parameters of a receiver that is used by multiple subscriptions can be changed at a single place.

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

- cloud-native: Cloud native protocol
- cntp-tcp: Civil Network Time Protocol (CNTP) TCP protocol
- cntp-tls: CNTP TLS protocol
- grpc-tcp: gRPC TCP protocol
- grpc-tls: gRPC TLS protocol
- native: Native protocol
- tls-native: Native TLS protocol

Named Protocol Receivers

Named protocol receivers are used to specify telemetry transports that use protocols. 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. 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.

You can configure a named protocol receiver by using the CLI or YANG models.

Configuring the Named Protocol Receiver Using 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:

- Name, which is the index in the list
- Protocol
- Profile
- Hostname
- Port number

The following 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>receiver1</name>
     <protocol>tls-native</protocol>
     <profile>tls-trustpoint</profile>
     <host>
      <hostname>rcvr.test.com</hostname>
     </host>
     <port>45000</port>
    </mdt-named-protocol-rcvr>
   </mdt-named-protocol-rcvrs>
  </mdt-config-data>
 </config>
</edit-config>
```

Subscription Configuration Using Named Receivers

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 configuration is part of the named receiver configuration. However; named protocol receivers still use the source VRF and source address of the subscriptions as part of the connection resolution process.

The only supported name 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 that 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.

Configuring a Named-Receiver Subscription Configuration Using YANG Model

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

The following 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>
```

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 are 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.

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.

The following 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
```

The following is sample output from the **show telemetry receiver name** command:

```
Device# show telemetry receiver name receiver1

  Name: receiver1
  Profile: tls-trustpoint
  State: Valid
  Last State Change: 08/12/20 19:55:54
  Explanation:
  Type: protocol
```

```
  Protocol: tls-native
  Host: rcvr.test.com
  Port: 45000
```

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.

The following 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>receiver1</name>
    <profile>tls-trustpoint</profile>
    <params>
     <protocol>tls-native</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>
```

Subscription Receiver Operation and Operational States

Subscription receivers are the subscription-related objects that connects 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 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 the table below.

Table 4: 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 the following 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.

Subscription Receiver Connections

This section provides information on how subscription receivers use connections.

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 Cisco proprietary 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.

Table 5: Telemetry Connection States

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

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

Table 6: 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.
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 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 the 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.

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

- Destination unreachable.
- No listener at the remote host port.
- Listener at the remote host port is of the wrong type.
- Authentication failures.

Note When a connection setup is in progress, any subscription receiver using this connection is 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. The following 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.

Table 7: 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 them	No limit; continuously requests re-resolution when connection retries fail. (14 seconds per try.)
• cloud-native • cntp-tcp • cntp-tls • native • tls-native		5, 10, 15, 20, 25, and 30 seconds.

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 8: 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.
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.
Subscription receiver is connected, but no updates are received.	**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.

show telemetry internal 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.

Displaying On-Change Subscription YANG Models

The Cisco-IOS-XE-mdt-capabilities-oper.YANG model can be queried to display information about the models that support on-change subscriptions and their transports.

Subscription Monitoring

Subscriptions of all types can be monitored by using CLIs and management protocol operations.

CLI

Use the **show telemetry ietf subscription** command to display information about telemetry subscriptions. The following 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:
```

NETCONF

The following 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">
<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">
      <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>
```

Streams

A stream defines 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.

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 module (from the YANG model Cisco-IOS-XE-mdt-oper.yang) in the *mdt-streams* container.

The following 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">
<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">
      <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.

The yang-push Stream

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 for this stream can be 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*.

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*.

IETF Draft Compliance

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*

Note The following 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

X-Path Filter for yang-push

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}]
  ```

  The following is an example of an 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 using [] between the keys, and encapsulate the string with ". The following is an example of an XPath expression:

  ```
  filter xpath
  /environment-ios-xe-oper:environment-sensors/environment-sensor[location=\"Switch\ 1\"]
  [name=\"Inlet\ Temp\ Sens\"]/current-reading
  ```

- 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.

XPath Expressions Supported on Cisco Catalyst 9800 Wireless Controllers

In Cisco IOS XE Bengaluru, 17.4.1, the following set of OpenConfig XPath expressions are supported on the Cisco Catalyst 9800 Series Wireless Controllers.

Ensure that you run the following RPC using any of the programmability interfaces, such as NETCONF, RESTCONF, or gNMI protocol, to enable telemetry subscription:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <provision-aps xmlns="http://openconfig.net/yang/wifi/ap-manager">
        <provision-ap>
          <mac>eth_mac_of_the_AP</mac>
          <config>
            <mac>eth_mac_of_the_AP</mac>
            <hostname>AP_NAME</hostname>
          </config>
```

```
        </provision-ap>
      </provision-aps>
    </config>
  </edit-config>
</rpc>
```

All of the XPath expressions listed below are a part of the *openconfig-access-points* YANG model, except the last one, which is a part of the *openconfig-ap-manager* YANG model. For the telemetry operation to work correctly, ensure that configurations are done based on the OpenConfig model.

- /access-points/access-point/radios/radio/state
- /access-points/access-point/radios/radio/neighbors/neighbor
- /access-points/access-point/radios/radio/neighbors/neighbor/state
- /access-points/access-point/ssids/ssid/bssids/bssid/state/counters
- /access-points/access-point/ssids/ssid/clients/client/state/counters
- /access-points/access-point/ssids/ssid/clients/client/client-rf/state
- /access-points/access-point/ssids/ssid/clients/client/client-connection/state
- /access-points/access-point/system/aaa/server-groups/server-group/servers/server/radius/state
- /joined-aps/joined-ap/state/opstate

When you subscribe to an XPath, you receive data for the subscribed XPath and all the XPaths under it in the hierarchy. For example, subscribing to */access-points/access-point/radios/radio/state* delivers data for all the leaves associated with it, as well as the subcontainers under it.

If you require only a subset of information, set filters in the XPath expressions to limit the updates. To filter the data of a specific access point (AP), use a key after the node. For example, to receive data for an AP with hostname ‘my_hostname’, use the subscription *XPath: access-point[hostname=’my_hostname’]*. Note that the data updates will contain data objects from all the leaves, and not just from the limited subset that is defined.

Scale Information

The following tables show the minimum recommended intervals for each of the gathering points under three different scale scenarios.

Scenario1: Full Scale with four SSIDs

Table 9: Setup

APs	2,000
Clients	30,000
SSIDs per AP	4
BSSIDs per AP	8
Physical neighbors per AP	12
Neighbors per AP	96

Table 10: Recommended Intervals

Gathering Point	Records	Recommended Interval (Seconds) One Collector	Recommended Interval (Seconds) Two Collectors
Joined	2000	30	60
AAA	2000	30	60
Radio	4000	30	60
Client RF	30,000	30	60
Client CNTR	30,000	30	60
Client CONN	30,000	60	120
BSSID	16,000	90	180
Neighbor	192,000	180	360

Scenario2: Full Scale with six SSIDs

Table 11: Setup

APs	2,000
Clients	30,000
SSIDs per AP	6
BSSIDs per AP	12
Physical neighbors per AP	12
Neighbors per AP	144

Table 12: Recommended Intervals

Gathering point	Records	Recommended Interval (Seconds) One Collector	Recommended Interval (Seconds) Two Collectors
Joined	2000	30	60
AAA	2000	30	60
Radio	4000	30	60
Client RF	30,000	30	60
Client CNTR	30,000	30	60

Gathering point	Records	Recommended Interval (Seconds) One Collector	Recommended Interval (Seconds) Two Collectors
Client CONN	30,000	60	120
BSSID	24,000	120	240
Neighbor	288,000	240	420

Scenario3: Reduced Scale with six SSIDs

Table 13: Setup

APs	1,000
Clients	15,000
SSIDs per AP	6
BSSIDs per AP	12
Physical neighbors per AP	12
Neighbors per AP	144

Table 14: Recommended Intervals

Gathering Point	Records	Recommended Interval (Seconds) One Collector	Recommended Interval (Seconds) Two Collectors
Joined	1000	NA	30
AAA	1000	NA	30
Radio	2000	NA	30
Client RF	15,000	NA	30
Client CNTR	15,000	NA	30
Client CONN	15,000	NA	30
BSSID	12,000	NA	120
Neighbor	144,000	NA	180

XPath Values and Corresponding Rates on Cisco Catalyst 9800 Wireless Controllers

In the Cisco-IOS-XE-wireless-mesh-rpc, following are the permitted values and corresponding rates for XPath `/exec-linktest-ap/data-rate-idx`:

```
ewlc-mesh-linktest-rate-idx-1 1 Mbps
ewlc-mesh-linktest-rate-idx-2 2 Mbps
ewlc-mesh-linktest-rate-idx-3 5 Mbps
```

```
ewlc-mesh-linktest-rate-idx-4 6 Mbps
ewlc-mesh-linktest-rate-idx-5 9 Mbps
ewlc-mesh-linktest-rate-idx-6 11 Mbps
ewlc-mesh-linktest-rate-idx-7 12 Mbps
ewlc-mesh-linktest-rate-idx-8 18 Mbps
ewlc-mesh-linktest-rate-idx-9 24 Mbps
ewlc-mesh-linktest-rate-idx-10 36 Mbps
ewlc-mesh-linktest-rate-idx-11 48 Mbps
ewlc-mesh-linktest-rate-idx-12 54 Mbps
ewlc-mesh-linktest-rate-idx-13 108 Mbps
ewlc-mesh-linktest-rate-idx-14 m0
ewlc-mesh-linktest-rate-idx-15 m1
ewlc-mesh-linktest-rate-idx-16 m2
ewlc-mesh-linktest-rate-idx-17 m3
ewlc-mesh-linktest-rate-idx-18 m4
ewlc-mesh-linktest-rate-idx-19 m5
ewlc-mesh-linktest-rate-idx-20 m6
ewlc-mesh-linktest-rate-idx-21 m7
ewlc-mesh-linktest-rate-idx-22 m8
ewlc-mesh-linktest-rate-idx-23 m9
ewlc-mesh-linktest-rate-idx-24 m10
ewlc-mesh-linktest-rate-idx-25 m11
ewlc-mesh-linktest-rate-idx-26 m12
ewlc-mesh-linktest-rate-idx-27 m13
ewlc-mesh-linktest-rate-idx-28 m14
ewlc-mesh-linktest-rate-idx-295 m15
```

Periodic Publication for yang-push

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

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 to data (the initial sychronization 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.

The 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. The following guideline applies to the XPath expression:

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

TLDP On-Change Notifications

Targeted Label Discovery Protocol (T-LDP) is an LDP session between label-switched routers (LSRs) that are not directly connected. The TLDP On-Change Notifications feature notifies users when TLDP sessions come up or go down and when TLDP is configured or disabled. TLDP must be enabled for the notifications to work.

Event-based notifications are generated in the following two scenarios:

- Configured events are generated when TLDP is configured and removed from a device. Notifications are also generated when a TLDP session comes up and goes down.
- Notifications are also generated when a TLDP session comes up and goes down.

Transport Protocol

The protocol that is used for the connection between 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.

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

Supported transport protocols are gNMI, gRPC, and NETCONF.

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.

Three update notification formats are used when using NETCONF as the transport protocol:

- 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.

The yang-push Format

When the *yang-push* source stream is sent over NETCONF as a 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.

The 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. As in prior releases, gRPC over TLS can be configured only with server validation.

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

This feature 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.

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. After switchover, you must destroy and re-establish all the sessions that use crypto, including NETCONF sessions that carry telemetry subscriptions. 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.

Note 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.

Pubd Restartability

In Cisco IOS XE Cupertino 17.9.1, the pubd process is restartable on all platforms. Prior to this release, pubd was restartable only on certain platforms. On other platforms, to restart the pubd process, the whole device had to be restarted.

Pubd can be restarted by removing and re-adding the NETCONF-YANG or gNXI configuration, as applicable. Note that this will also restart the other NETCONF-YANG or gNXI processes.

Sample Model-Driven Telemetry RPCs

The following section provides a list of sample RPCs, and describes how to configure subscriptions.

Managing Configured Subscriptions

Note Currently, you can only use the gRPC protocol for managing configured subscriptions.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **telemetry ietf subscription** *id*
4. **stream yang-push**
5. **filter xpath** *path*

6. **update-policy** {**on-change** | **periodic**} *period*
7. **encoding encode-kvgpb**
8. **source-vrf** *vrf-id*
9. **source-address** *source-address*
10. **receiver ip address** *ip-address receiver-port* **protocol** *protocol* **profile** *name*
11. **end**

DETAILED STEPS

	Command or Action	Purpose
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 kvGPB encoding.
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:**	Configures the source address.

	Command or Action	Purpose
	`Device(config-mdt-subs)# source-vrf 192.0.2.1`	
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.

Configuring On-Change gRPC Subscriptions

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **telemetry ietf subscription** *id*
4. **stream yang-push**
5. **filter xpath** *path*
6. **update-policy** {**on-change** | **periodic** *period*}
7. **encoding encode-kvgpb**
8. **receiver ip address** *ip-address receiver-port* **protocol** *protocol* **profile** *name*
9. **end**

DETAILED STEPS

	Command or Action	Purpose
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.

	Command or Action	Purpose
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.

Receiving a Response Code

When a subscription is successfully created, the device responds with a subscription result of notif-bis:ok and a subscription ID. The following 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. The following 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, and does not exist, the periodic update will be empty and will have a self-closing *datastore-contents-xml* tag. The following 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>
```

Retrieving Subscription Details

You can retrieve the list of current subscriptions by sending a <get> RPC to the Cisco-IOS-XE-mdt-oper model. You can also use the **show telemetry ietf subscription** command to display the list of current subscriptions.

The following 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">
        <mdt-subscriptions/>
      </mdt-oper-data>
    </filter>
  </get>
</rpc>
```

The following 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">
      <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>
```

The following 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
```

The following 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:
```

The following 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:
```

The following sample RPC shows how to retrieve subscription details uisng RESTCONF:

```
Subscription details can also be retrieved through a RESTCONF GET request to the
Cisco-IOS-XE-mdt-oper database:
URI:
https://10.85.116.28:443/restconf/data/Cisco-IOS-XE-mdt-oper: 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:mdt-subscriptions": [
    {
      "subscription-id": 101,
      "base": {
        "stream": "yang-push",
        "encoding": "encode-kvgpb",
        "source-vrf": "",
        "no-synch-on-start": 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": "5.28.35.35",
          "port": 57555,
          "protocol": "grpc-tcp",
          "state": "rcvr-state-connecting",
          "comments": "Connection retries in progress",
          "profile": ""
        }
      ]
    }
  ]
}
```

Configuring Named Protocol Receiver Using the CLI

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **telemetry receiver protocol** *receiver-name*
4. **protocol** {**cloud-native** | **cntp-tcp** | **cntp-tls profile** *profile-name* | **grpc-tcp** | **grpc-tls profile** *profile-name* | **native** | **tls-native profile** *profile-name*}
5. **host** {**ip** *ip-address* | **name** *hostname*} *receiver-port*
6. **end**

DETAILED STEPS

	Command or Action	Purpose
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.

Subscription Configuration Using Named Receivers Using CLI

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **telemetry ietf subscription** *id*
4. **receiver-type protocol** }
5. **receiver name** *name*
6. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable**	Enables privileged EXEC mode.

	Command or Action	Purpose
	Example: `Device> enable`	• 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.

Additional References for Model-Driven Telemetry

Related Documents

Related Topic	Document Title
YANG Explorer	https://github.com/CiscoDevNet/yang-explorer

Standards and RFCs

Standard/RFC	Title
Custom Subscription to Event Notifications draft-ietf-netconf-subscribed-notifications-03	https://tools.ietf.org/id/draft-ietf-netconf-subscribed-notifications-03.txt
NETCONF Support for Event Notifications	draft-ietf-netconf-netconf-event-notifications-01
RFC 5277	NETCONF Event Notifications
RFC 6241	Network Configuration Protocol (NETCONF)
RFC 7950	The YANG 1.1 Data Modeling Language
RFC 8040	RESTCONF Protocol

Standard/RFC	Title
Subscribing to Event Notifications	draft-ietf-netconf-rfc5277bis-01
Subscribing to YANG Datastore Push Updates	draft-ietf-netconf-yang-push-04
Subscribing to YANG datastore push updates draft-ietf-netconf-yang-push-07	https://tools.ietf.org/id/draft-ietf-netconf-yang-push-07.txt

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

Feature Information for Model-Driven Telemetry

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 15: Feature Information for Model-Driven Telemetry

Feature Name	Release	Feature Information
Model-Driven Telemetry NETCONF Dial-In	Cisco IOS XE Everest 16.6.1	Model-driven telemetry allows network devices to continuously stream real time configuration and operating state information to subscribers. • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9500 Series Switches
	Cisco IOS XE Everest 16.6.2	• Cisco Catalyst 9400 Series Switches
	Cisco IOS XE Fuji 16.7.1	• Cisco 4000 Series Integrated Services Routers • Cisco ASR 1000 Series Aggregation Services Routers (ASR1001-HX, ASR1001-X, ASR1002-HX, ASR1002-X)
	Cisco IOS XE Fuji 16.8.1	• Cisco 1000 Series Integrated Services Routers • Cisco ASR 1000 RP2 and RP3 Series Aggregation Services Routers
	Cisco IOS XE Fuji 16.8.1a	• Cisco Catalyst 9500-High Performance Series Switches
	Cisco IOS XE Fuji 16.9.1	• Cisco ASR 900 Series Aggregation Services Routers • Cisco ASR 920 Series Aggregation Services Router • Cisco cBR-8 Converged Broadband Router • Cisco Network Convergence System 4200 Series

Feature Name	Release	Feature Information
	Cisco IOS XE Gibraltar 16.9.2	• Cisco Catalyst 9200 and 9200L Series Switches • Cisco Catalyst 9300L SKUs
	Cisco IOS XE Gibraltar 16.10.1	• Cisco Cloud Services Router 1000v • Cisco Network Convergence System 520 Series
	Cisco IOS XE Gibraltar 16.11.1	• Cisco Catalyst 9600 Series Switches

Feature Name	Release	Feature Information
Model-Driven Telemetry gNMI Dial-In	Cisco IOS XE Gibraltar 16.12.1	Telemetry updates that are sent to the initiator/subscriber are called Dial-in. This feature was implemented on the following platforms: • Cisco Catalyst 9200 and 9200L Series Switches • Cisco Catalyst 9300 and 9300L Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 and 9500-High Performance Series Switches • Cisco Catalyst 9600 Series Switches • Cisco cBR-8 Converged Broadband Router
	Cisco IOS XE Amsterdam 17.1.1	• Cisco ASR 900 Series Aggregation Services Routers • Cisco ASR 920 Series Aggregated Services Router • Cisco Network Convergence System 520 Series • Cisco Network Convergence System 4200 Series
	Cisco IOS XE Amsterdam 17.2.1	Cisco ASR 1000 Series Aggregation Services Routers

Feature Name	Release	Feature Information
Model-Driven Telemetry gRPC Dial-Out	Cisco IOS XE Gibraltar 16.10.1	Configured subscriptions cause the publisher to initiate connections to receivers, and these connections are considered as dial-out. This feature was implemented on the following platforms: • Cisco 1000 Series Integrated Services Routers • Cisco 4000 Series Integrated Services Routers • Cisco ASR 1000 Series Aggregation Services Routers • Cisco ASR 900 Series Aggregation Services Routers • Cisco ASR 920 Series Aggregated Services Router • Cisco Catalyst 9200 and 9200L Series Switches • Cisco Catalyst 9300 and 9300L Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 and 9500-High Performance Series Switches • Cisco cBR-8 Converged Broadband Router • Cisco Cloud Services Router 1000V Series • Cisco Network Convergence System 520 Series • Cisco Network Convergence System 4200 Series
	Cisco IOS XE Gibraltar 16.11.1	• Cisco Catalyst 9600 Series Switches

Feature Name	Release	Feature Information
Model-Driven Telemetry: Kill Subscription	Cisco IOS XE Gibraltar 16.11.1	To delete dynamic subscriptions, you can use the CLI and the kill-subscription RPC. • Cisco ASR 900 Series Aggregation Services Routers • Cisco ASR 920 Series Aggregated Services Router (RSP2) • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches • Cisco Catalyst 9200 and 9200L Series Switches • Cisco Catalyst 9300 and 9300L Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 and 9500-High Performance Series Switches • Cisco Network Convergence System 520 Series • Cisco Network Convergence System 4200 Series

Feature Name	Release	Feature Information
TLDP On-Change Notifications	Cisco IOS XE Amsterdam 17.2.1	The TLDP On-Change Notifications feature notifies users when TLDP sessions come up or go down and when TLDP is configured or disabled. This feature was implemented on the following platforms: • Cisco 4000 Series Integrated Services Routers • Cisco Catalyst 9200 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 Series Switches

Feature Name	Release	Feature Information
TLS for gRPC Dial-Out	Cisco IOS XE Amsterdam 17.1.1	Transport-Layer Security is supported for gRPC dial-out. This feature is supported on the following platforms: • Cisco 1000 Series Integrated Services Routers • Cisco 4000 Series Integrated Services Routers • Cisco ASR 1000 Series Aggregation Services Routers • Cisco ASR 900 Series Aggregation Services Routers • Cisco ASR 920 Series Aggregated Services Router • Cisco Catalyst 9200 and 9200L Series Switches • Cisco Catalyst 9300 and 9300L Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 and 9500-High Performance Series Switches • Cisco Catalyst 9600 Series Switches • Cisco Catalyst 9800-40 Series Wireless Controller • Cisco Catalyst 9800-80 Series Wireless Controller • Cisco cBR-8 Converged Broadband Router • Cisco Cloud Services Router 1000V Series • Cisco Network Convergence System 520 Series • Cisco Network Convergence System 4200 Series

Feature Name	Release	Feature Information
FQDN Support for gRPC Subscriptions	Cisco IOS XE Bengaluru 17.6.1	With the introduction of the FQDN Support for gRPC Subscriptions feature, along with IP addresses, FQDN can also be used for gRPC subscriptions. • Cisco 1000 Series Integrated Services Routers • Cisco 4000 Series Integrated Services Routers • Cisco ASR 1000 Series Aggregation Services Routers • Cisco ASR 900 Series Aggregation Services Routers • Cisco ASR 920 Series Aggregated Services Router • Cisco Catalyst 9200 and 9200L Series Switches • Cisco Catalyst 9300 and 9300L Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 and 9500-High Performance Series Switches • Cisco Catalyst 9600 Series Switches • Cisco Catalyst 9800-40 Series Wireless Controller • Cisco Catalyst 9800-80 Series Wireless Controller • Cisco cBR-8 Converged Broadband Router • Cisco Cloud Services Router 1000V Series • Cisco Network Convergence System 520 Series • Cisco Network Convergence System 4200 Series

Feature Name	Release	Feature Information
Leaf-Level Filtering	Cisco IOS XE Cupertino 17.7.1	The Leaf-Level Filtering for Telemetry feature allows filtering below the gatherpoint level for the optimized code paths. • Cisco Catalyst 9300 and 9300L Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 and 9500-High Performance Series Switches • Cisco Catalyst 9600 Series Switches
Mutual Authentication for gRPC Telemetry	Cisco IOS XE Cupertino 17.9.1	A new gRPC TLS profile that contains a pair of trustpoints was added to the telemetry configuration, so that a client ID certificate can be specified for mutual authentication. This new profile can be used instead of the trustpoint containing the server CA certificate when configuring the receiver profile. The trustpoint containing the server CA certificate is now configured as part of the gRPC TLS profile. This feature is supported on the following platforms: • Cisco Catalyst 9800-CL Series Wireless Controller • Cisco Catalyst 9800-40 Series Wireless Controller • Cisco Catalyst 9800-80 Series Wireless Controller

Feature Name	Release	Feature Information
Pubd Restartability	Cisco IOS XE Cupertino 17.9.1	

Feature Name	Release	Feature Information
		The pubd process is made restartable from this release onwards. This feature is supported on the following platforms: • Cisco 1000 Series Integrated Services Routers • Cisco 4000 Series Integrated Services Routers • Cisco ASR 1000 Series Aggregation Services Routers • Cisco ASR 900 Series Aggregation Services Routers • Cisco ASR 920 Series Aggregated Services Router • Cisco Catalyst 9200 and 9200L Series Switches • Cisco Catalyst 9300 and 9300L Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 and 9500-High Performance Series Switches • Cisco Catalyst 9600 Series Switches • Cisco Catalyst 9800-CL Series Wireless Controller • Cisco Catalyst 9800-40 Series Wireless Controller • Cisco Catalyst 9800-80 Series Wireless Controller • Cisco cBR-8 Converged Broadband Router • Cisco Cloud Services Router 1000V Series • Cisco Network Convergence System 520 Series

Feature Name	Release	Feature Information
		• Cisco Network Convergence System 4200 Series