モデル駆動型テレメトリ

- テレメトリについて （1 ページ）
- テレメトリのライセンス要件 （3 ページ）
- Guidelines and Limitations, on page 4
- CLI を使用したテレメトリの構成 （9 ページ）
- NX-API を使用したテレメトリの構成 （31 ページ）
- クラウド スケール ソフトウェア テレメトリ （46 ページ）
- テレメトリ パス ラベル （47 ページ）
- ネイティブ データ送信元パス （65 ページ）
- ストリーミング Syslog （80 ページ）
- その他の参考資料 （86 ページ）

テレメトリについて

分析やトラブルシューティングのためのデータ収集は、ネットワークの健全性をモニタリングする上で常に重要な要素であり続けています。

Cisco NX-OS は、ネットワークからデータを収集するための、SNMP、CLI や Syslog といった複数のメカニズムを提供します。これらのメカニズムには、自動化や拡張に対する制約があります。ネットワーク要素からのデータの最初の要求がクライアントから出された場合、プルモデルの使用が制限されることもその制約の 1 つです。プル モデルは、ネットワーク内に複数のネットワーク管理ステーション（NMS）がある場合は拡張しません。このモデルを使用すると、クライアントが要求した場合に限り、サーバーがデータを送信します。このような要求を開始するには、手動による介入を続けて行う必要があります。このような手動による介入を続けると、プル モデルの効率が失われます。

プッシュ モデルは、ネットワークからデータを継続的にストリーミングし、クライアントに通知します。テレメトリはプッシュ モデルをイネーブルにし、モニタリング データにほぼリアルタイムでアクセスできるようにします。

テレメトリ コンポーネントとプロセス

テレメトリは、次の 4 つの主要な要素で構成されます。

- **データ収集**：テレメトリ データは、識別名（DN）パスを使用して指定されたオブジェクトモデルのブランチにあるデータ管理エンジン（DME）データベースから収集されます。データは定期的に取得されるか（頻度ベース）、指定したパスのオブジェクトで変更があった場合にのみ取得できます（イベント ベース）。NX-API を使用して、頻度ベースのデータを収集できます。

- **データ エンコーディング**：テレメトリ エンコーダが、収集されたデータを目的の形式で転送できるようにカプセル化します。

  NX-OS は、テレメトリ データを Google Protocol Buffers（GPB）および JSON 形式でエンコードします。

- **データ トランスポート**：NX-OS は、JSON エンコードに HTTP を使用してテレメトリ データを転送し、GPB エンコードに Google リモート プロシージャ コール (gRPC) プロトコルを使用します。gRPC レシーバーは、4 MB を超えるメッセージサイズをサポートします。（証明書が構成されている場合は、HTTPS を使用したテレメトリ データもサポートされます。）

  Cisco NX-OS リリース 9.2(1) 以降、テレメトリは IPv6 接続先および IPv4 接続先へのストリーミングをサポートするようになりました。

  次のコマンドを使用して、JSON または GPB のデータグラム ソケットを使用してデータをストリーミングするように UDP トランスポートを構成します。

```
destination-group num
  ip address xxx.xxx.xxx.xxx port xxxx protocol UDP encoding {JSON | GPB }
```

  IPv6 接続先の例:

```
destination-group 100
  ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB
```

  UDP テレメトリには次のヘッダーがあります。

```
typedef enum tm_encode_ {
  TM_ENCODE_DUMMY,
  TM_ENCODE_GPB,
  TM_ENCODE_JSON,
  TM_ENCODE_XML,
  TM_ENCODE_MAX,
} tm_encode_type_t;

typedef struct tm_pak_hdr_ {
  uint8_t version; /* 1 */
  uint8_t encoding;
  uint16_t msg_size;
  uint8_t secure;
  uint8_t padding;
}__attribute__ ((packed, aligned (1))) tm_pak_hdr_t;
```

  次のいずれかの方法で、ペイロードの最初の 6 バイトを使用して、UDP を使用してテレメトリ データを処理します。

- 受信側が複数のエンドポイントから異なるタイプのデータを受信することになっている場合は、ヘッダーの情報を読んで、データのデコードに使用するデコーダー（JSON または GPB）を決定します。
- 1 つのデコーダー（JSON または GPB）が必要で、もう 1 つのデコーダーは必要ない場合は、ヘッダーを削除します。

- **テレメトリ レシーバー**：テレメトリ レシーバーは、テレメトリ データを保存するリモート管理システムです。

GPB エンコーダーは、汎用キーと値の形式でデータを格納します。また、データを GPB 形式に変換するには、コンパイルされた `.proto` ファイル形式のメタデータが GPB エンコーダに必要です。

データ ストリームを正しく受信してデコードするには、受信側でエンコードとトランスポート サービスを記述した `.proto` ファイルが必要です。エンコードは、バイナリ ストリームをキー値の文字列のペアにデコードします。

GPB エンコーディングと gRPC トランスポートを記述する telemetry `.proto` ファイルは、Cisco の GitLab で入手できます。 https://github.com/CiscoDevNet/nx-telemetry-proto

テレメトリ プロセスの高可用性

テレメトリ プロセスの高可用性は、次の動作でサポートされています。

- **[システムのリロード（System Reload）]** — システムのリロード中に、テレメトリ構成とストリーミング サービスが復元されます。
- **[スーパーバイザ フェールオーバー（Supervisor Failover）]** — テレメトリはホット スタンバイではありませんが、テレメトリ構成とストリーミング サービスは、新しい現用系スーパーバイザが実行されているときに復元されます。
- **[プロセスの再起動（Process Restart）]** — なんらかの理由でテレメトリ プロセスがフリーズまたは再起動した場合、テレメトリが再開されると、構成およびストリーミング サービスが復元されます。

テレメトリのライセンス要件

製品	ライセンス要件
Cisco NX-OS	テレメトリにはライセンスは必要ありません。ライセンス パッケージに含まれていない機能は Cisco NX-OS イメージにバンドルされており、無料で提供されます。NX-OS ライセンス方式の詳細については、『*Cisco NX-OS Licensing Guide*』を参照してください。

Guidelines and Limitations

Telemetry has the following configuration guidelines and limitations:

- For information about supported platforms, see the Nexus Switch Platform Matrix.
- Cisco NX-OS releases that support the data management engine (DME) Native Model support Telemetry.
- Support is in place for the following:
  - DME data collection
  - NX-API data sources
  - Google protocol buffer (GPB) encoding over Google Remote Procedure Call (gRPC) transport
  - JSON encoding over HTTP
- The smallest sending interval (cadence) supported is five seconds for a depth of 0. The minimum cadence values for depth values greater than 0 depends on the size of the data being streamed out. Configuring any cadences below the minimum value may result in undesirable system behavior.
- Telemetry supports up to five remote management receivers (destinations). Configuring more than five remote receivers may result in undesirable system behavior.
- Telemetry can consume up to 20% of the CPU resource.

Configuration Commands After Downgrading to an Older Release

After a downgrade to an older release, some configuration commands or command options can fail because the older release may not support them. When downgrading to an older release, unconfigure and reconfigure the telemetry feature after the new image comes up. This sequence avoids the failure of unsupported commands or command options.

The following example shows this procedure:

- Copy the telemetry configuration to a file:

```
switch# show running-config | section telemetry
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch# show running-config | section telemetry > telemetry_running_config
switch# show file bootflash:telemetry_running_config
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
```

```
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch#
```

- Execute the downgrade operation. When the image comes up and the switch is ready, copy the telemetry configurations back to the switch.

```
switch# copy telemetry_running_config running-config echo-commands
`switch# config terminal`
`switch(config)# feature telemetry`
`switch(config)# telemetry`
`switch(config-telemetry)# destination-group 100`
`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB `
`switch(conf-tm-dest)# sensor-group 100`
`switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0`
`switch(conf-tm-sensor)# subscription 600`
`switch(conf-tm-sub)# dst-grp 100`
`switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000`
`switch(conf-tm-sub)# end`
Copy complete, now saving to disk (please wait)...
Copy complete.
switch#
```

gRPC Error Behavior

The switch client disables the connection to the gRPC receiver if the gRPC receiver sends 20 errors. Unconfigure then reconfigure the receiver's IP address under the destination group to enable the gRPC receiver. Errors include:

- The gRPC client sends the wrong certificate for secure connections.
- The gRPC receiver takes too long to handle client messages and incurs a timeout. Avoid timeouts by processing messages using a separate message processing thread.

Support for gRPC Chunking

Starting with Release 9.2(1), support for gRPC chunking has been added. For streaming to occur successfully, you must enable chunking if gRPC has to send an amount of data greater than 12 MB to the receiver.

The gRPC user must do the gRPC chunking. The gRPC client side does the fragmentation, and the gRPC server side does the reassembly. Telemetry is still bound to memory and data can be dropped if the memory size is more than the allowed limit of 12 MB for telemetry. In order to support chunking, use the telemetry `.proto` file that is available at Cisco's GibLab, which has been updated for gRPC chunking, as described in テレメトリ コンポーネントとプロセス, on page 1.

The chunking size is from 64 through 4096 bytes.

Following shows a configuration example through the NX-API CLI:

```
feature telemetry
!
telemetry
  destination-group 1
    ip address 171.68.197.40 port 50051 protocol gRPC encoding GPB
    use-chunking size 4096
```

```
  destination-group 2
    ip address 10.155.0.15 port 50001 protocol gRPC encoding GPB
    use-chunking size 64
  sensor-group 1
    path sys/intf depth unbounded
  sensor-group 2
    path sys/intf depth unbounded
  subscription 1
    dst-grp 1
    snsr-grp 1 sample-interval 10000
  subscription 2
    dst-grp 2
    snsr-grp 2 sample-interval 15000
```

Following shows a configuration example through the NX-API REST:

```
{
    "telemetryDestGrpOptChunking": {
        "attributes": {
            "chunkSize": "2048",
            "dn": "sys/tm/dest-1/chunking"
        }
    }
}
```

The following error message appears on systems that do not support gRPC chunking, such as the Cisco MDS series switches:

```
MDS-9706-86(conf-tm-dest)# use-chunking size 200
ERROR: Operation failed: [chunking support not available]
```

NX-API Sensor Path Limitations

NX-API can collect and stream switch information not yet in the DME using **show** commands. However, using the NX-API instead of streaming data from the DME has inherent scale limitations as outlined:

- The switch backend dynamically processes NX-API calls such as **show** commands,
- NX-API spawns several processes that can consume up to a maximum of 20% of the CPU.
- NX-API data translates from the CLI to XML to JSON.

The following is a suggested user flow to help limit excessive NX-API sensor path bandwidth consumption:

1. Check whether the **show** command has NX-API support. You can confirm whether NX-API supports the command from the VSH with the pipe option: `show <command> | json` or `show <command> | json pretty`.

Note Avoid commands that take the switch more than 30 seconds to return JSON output.

2. Refine the **show** command to include any filters or options.
    - Avoid enumerating the same command for individual outputs; for example, **show vlan id 100** , **show vlan id 101** , and so on. Instead, use the CLI range options; for example, **show vlan id 100-110,204** , whenever possible to improve performance.

If only the summary or counter is needed, then avoid dumping a whole show command output to limit the bandwidth and data storage that is required for data collection.

3. Configure telemetry with sensor groups that use NX-API as their data sources. Add the **show** commands as sensor paths
4. Configure telemetry with a cadence of five times the processing time of the respective **show** command to limit CPI usage.
5. Receive and process the streamed NX-API output as part of the existing DME collection.

Telemetry VRF Support

Telemetry VRF support allows you to specify a transport VRF, which means that the telemetry data stream can egress through front-panel ports and avoid possible competition between SSH or NGINX control sessions.

You can use the **use-vrf** *vrf-name* command to specify the transport VRF.

The following example specifies the transport VRF:

The following is an example of use-vrf as a POST payload:

```
{
              "telemetryDestProfile": {
                "attributes": {
                  "adminSt": "enabled"
                },
                "children": [
                  {
                    "telemetryDestOptVrf": {
                      "attributes": {
                        "name": "default"
                      }
                    }
                  }
                ]
              }
            }
```

Certificate Trustpoint Support

Beginning in NX-OS release 10.1(1), the **trustpoint** keyword is added in the existing global level command.

The following is the command syntax:

```
switch(config-telemetry)# certificate ?
trustpoint    specify trustpoint label
WORD          .pem certificate filename (Max Size 256)
switch(config-telemetry)# certificate trustpoint
WORD          trustpoint label name (Max Size 256)
switch(config-telemetry)# certificate trustpoint trustpoint1 ?
WORD  Hostname associated with certificate (Max Size 256)
switch(config-telemetry)#certificate trustpoint trustpoint1 foo.test.google.fr
```

Destination Hostname Support

Beginning in NX-OS release 10.1(1), the **host** keyword is added in destination-group command.

The following is the example for the destination hostname support:

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ?
certificate Specify certificate
host Specify destination host
ip Set destination IPv4 address
ipv6 Set destination IPv6 address
...
switch(conf-tm-dest)# host ?
A.B.C.D|A:B::C:D|WORD  IPv4 or IPv6 address or DNS name of destination
switch(conf-tm-dest)#

switch(conf-tm-dest)# host abc port 11111 ?
protocol  Set transport protocol
switch(conf-tm-dest)# host abc port 11111 protocol ?
HTTP
UDP
gRPC
switch(conf-tm-dest)# host abc port 11111 protocol gRPC ?
encoding  Set encoding format
switch(conf-tm-dest)# host abc port 11111 protocol gRPC encoding ?
Form-data    Set encoding to Form-data only
GPB          Set encoding to GPB only
GPB-compact  Set encoding to Compact-GPB only
JSON         Set encoding to JSON
XML          Set encoding to XML
switch(conf-tm-dest)# host ip address 1.1.1.1 port 2222 protocol HTTP encoding JSON
<CR>
```

Support for Node ID

Beginning in NX-OS release 10.1(1), you can configure a custom Node ID string for a telemetry receiver through the **use-nodeid** command. By default, the host name is used, but support for a node ID enables you to set or change the identifier for the `node_id_str` of the telemetry receiver data.

You can assign the node ID through the telemetry destination profile, by using the **usenode-id** command. This command is optional.

The following example shows configuring the node ID.

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(conf-tm-dest-profile)# use-nodeid test-srvr-10
switch(conf-tm-dest-profile)#
```

The following example shows a telemetry notification on the receiver after the node ID is configured.

```
Telemetry receiver:
================================
node_id_str: "test-srvr-10"
subscription_id_str: "1"
encoding_path: "sys/ch/psuslot-1/psu"
collection_id: 3896
msg_timestamp: 1559669946501
```

Use the **use-nodeid** sub-command under the **host** command. The destination level **use-nodeid** configuration preceeds the global level configuration.

The following example shows the command syntax:

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# host 172.19.216.78 port 18112 protocol http enc json
switch(conf-tm-dest-host)# use-nodeid ?
WORD  Node ID (Max Size 128)
switch(conf-tm-dest-host)# use-nodeid session_1:18112
```

The following example shows the output from the Telemetry receiver:

```
>> Message size 923
Telemetry msg received @  23:41:38 UTC
    Msg Size: 11
    node_id_str    : session_1:18112
    collection_id : 3118
    data_source    : DME
    encoding_path : sys/ch/psuslot-1/psu
    collection_start_time    : 1598485314721
    collection_end_time    : 1598485314721
    data           :
```

Support for Streaming of YANG Models

Beginning in NX-OS release 9.2(1), telemetry supports the YANG ("Yet Another Next Generation") data modeling language. Telemetry supports data streaming for both device YANG and OpenConfig YANG.

For more information on the YANG data modeling language, see Infrastructure Overview and RESTConf Agent.

Support for Proxy

Beginning in NX-OS release 10.1(1), the **proxy** command is included in the host command. The following is the command syntax:

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# host 172.19.216.78 port 18112 protocol http enc json
switch(conf-tm-dest-host)# proxy ?
    A.B.C.D|A:B::C:D|WORD  IPv4 or IPv6 address or DNS name of proxy server
    <1-65535>  Proxy port number, Default value is 8080
username  Set proxy authentication username
password  Set proxy authentication password
```

gRPC Asynchronous Mode

The gRPC asynchronous mode is available only under the **host** command. In normal stream condition, this mode allows the receivers to stream data in **mdtDialout** call without exiting or receiving **WriteDone()** call.

The following is the command syntax:

```
nxosv-1(config-telemetry)# destination-group 1
nxosv-1(conf-tm-dest)# host 172.22.244.130 port 50007 ?
nxosv-1(conf-tm-dest-host)# grpc-async ?
```

CLI を使用したテレメトリの構成

NX-OS CLI を使用したテレメトリの構成

次の手順では、ストリーミング テレメトリを有効にし、データ ストリームの送信元と接続先を構成します。

手順の概要

1. **configure terminal**
2. **feature telemetry**
3. **feature nxapi**
4. **nxapi use-vrf management**
5. **telemetry**
6. （任意） **certificate** *certificate_path host_URL*
7. **sensor-group** *sgrp_id*
8. **path** *sensor_path* **depth unbounded** [**filter-condition** *filter*] [**alias** *path_alias*]
9. **destination-group** *dgrp_id*
10. （任意） **ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
11. （任意） **ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
12. *ip_version* **address** *ip_address* **port** *portnum*
13. （任意） **use-chunking size** *chunking_size*
14. **subscription** *sub_id*
15. **snsr-grp** *sgrp_id* **sample-interval** *interval*
16. **dst-grp** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		グローバル構成モードを開始します。

ステップ 2	**feature telemetry**	ストリーミング テレメトリ機能を有効にします。
ステップ 3	**feature nxapi**	NX-API を有効にします。

ステップ 4	**nxapi use-vrf management**
	例：
	`switch(config)#`
	`switch(config)# nxapi use-vrf management`
	`switch(config)#`
		NX-API 通信に使用する VRF 管理を有効にします。
		（注） ACL はネットスタックパケットのみをフィルタ処理できるため、10.2（3）F より前のリリースでは次の意味の警告が表示されます：
		「警告：構成された管理ACLは、HTTP サービスでは有効になりません。iptables を使用してアクセスを制限してください」。

	コマンドまたはアクション	目的
		（注） 10.2(3)F 以降、ACL は、管理 vrf に着信する netstack パケットと kstack パケットの両方をフィルタリングできます。次の意味の警告が表示されます： 「警告：非管理 VRF で構成された ACL は、その VRF の HTTP サービスでは有効になりません」。

ステップ 5	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		ストリーミング テレメトリの構成モードに入ります。

ステップ 6	（任意） **certificate** *certificate_path host_URL* 例： `switch(config-telemetry)# certificate /bootflash/server.key localhost`	既存の SSL/TLS 証明書を使用します。 EOR デバイスの場合、証明書もスタンバイ SUP にコピーする必要があります。

ステップ 7	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 100`
	`switch(conf-tm-sensor)#`
		ID *srgp_id* を持つセンサー グループを作成し、センサー グループ構成モードを開始します。
		現在は、数字の ID 値のみサポートされています。センサー グループでは、テレメトリ レポートのモニタリング対象ノードを定義します。

ステップ 8	**path** *sensor_path* **depth unbounded** [**filter-condition** *filter*] [**alias** *path_alias*]
	例：
	• 次のコマンドは、NX-API ではなく、DME または YANG に適用されます：
	`switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition eq(l2BD.operSt, "down")`
	以下の構文を使用し、状態ベースのフィルタリングを使用して、**operSt** が **up** から **down** に変化したときにのみトリガーするようにします。MO が変化しても通知しません。
	`switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition and(updated(l2BD.operSt),eq(l2BD.operSt,"down"))`
		ここでの制限なしとは、出力に子管理対象オブジェクト（MO）を含めることを意味します。したがって、POLL テレメトリ ストリームの場合、そのパスと EVENT のすべての子 MO が子 MO で行われた変更を取得します。
		（注） これは、データ送信元 DME パスにのみ適用されます。
		センサー グループにセンサー パスを追加します。
		• Cisco NX-OS 9.3(5) リリース以降では、**alias** キーワードが導入されています。
		• **depth** 設定では、センサー パスの取得レベルを指定します。**0 - 32**、**unbounded** の深さ設定がサポートされています。

	コマンドまたはアクション	目的

	UTR 側のパスを区別するには、次の構文を使用します。
	`switch(conf-tm-sensor)# path sys/ch/ftslot-1/ft alias ft_1`
	• 次のコマンドは、DME ではなく、NX-API または YANG に適用されます：
	`switch(conf-tm-sensor)# path "show interface" depth 0`
	• 次のコマンドは、デバイス YANG に適用されます：
	`switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items`
	• 次のコマンドは、OpenConfig YANG に適用されます：
	`switch(conf-tm-sensor)# path openconfig-bgp:bgp`
	`switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items alias bgp_alias`
	• 次のコマンドは、NX-API に適用されます：
	`switch(conf-tm-sensor)# path "show interface" depth 0 alias sh_int_alias`
	• 次のコマンドは、OpenConfigに適用されます：
	`switch(conf-tm-sensor)# path openconfig-bgp:bgp alias oc_bgp_alias`
		（注） **depth 0** デフォルトの深さです。
		NX-APIベースのセンサーパスは、**depth 0** のみを使用できます。
		イベント収集のパスがサブスクライブされている場合、深さは 0 とバウンドなしのみをサポートします。その他の値は 0 として扱われます。
		• オプションの **filter-condition** パラメータを指定して、イベントベースのサブスクリプション用の特定のフィルタを作成できます。
		状態ベースのフィルタ処理の場合、フィルタ処理は、状態が変化したときと、指定された状態でイベントが発生したときの両方を返します。つまり、**eq(l2Bd.operSt, "down")** の DN **sys/bd/bd-[vlan]** のフィルタ条件は、operSt が変更されたとき、および operSt が **down** である間に DN のプロパティが変更されたとき（VLAN が動作上 **down** である間に **no shutdown** コマンドが発行された場合など）にトリガーされます。
		• YANG モデルの場合、センサー パスのフォーマットは *module_name* : *YANG_path* です。*module_name* は YANG モデル ファイルの名前です。次に例を示します。
		• デバイス YANG の場合：
		Cisco-NX-OS-device:System/bgp-items/inst-items
		• OpenConfig YANG の場合：
		openconfig-bgp:bgp
		（注） **depth**、**filter-condition**、および **query-condition** パラメータは、現在 YANG ではサポートされていません。
		openconfig YANG モデルの場合は、https://github.com/YangModels/yang/tree/master/vendor/cisco/nx に移動して、最新リリースの適切なフォルダに移動します。

	コマンドまたはアクション	目的
		特定のモデルをインストールする代わりに、すべての OpenConfig モデルを含む openconfig-all RPM をインストールできます。 次に例を示します。 **install add mtx-openconfig-bgp-1.0.0.0.0-7.0.3.IHD8.1.lib32_n9000.rpm activate**

ステップ 9	**destination-group** *dgrp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 100`
	`switch(conf-tm-dest)#`
		接続先グループを作成して、接続先グループ構成モードを開始します。
		現在、*dgrp_id* は、数字の ID 値のみをサポートしています。

ステップ 10	（任意）**ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	例：
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50007 protocol HTTP encoding JSON`
		エンコードされたテレメトリ データを受信する IPv4 IP アドレスとポートを指定します。
		（注） gRPC はデフォルトのトランスポートプロトコルです。
		GPB がデフォルトのエンコーディングです。

ステップ 11	（任意）**ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	例：
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8001 protocol HTTP encoding JSON`
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8002 protocol UDP encoding JSON`
		エンコードされたテレメトリ データを受信する IPv6 IP アドレスとポートを指定します。
		（注） gRPC はデフォルトのトランスポートプロトコルです。
		GPB がデフォルトのエンコーディングです。

ステップ 12	*ip_version* **address** *ip_address* **port** *portnum*
	例：
	• IPv4 の場合：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50003`
	• IPv6 の場合：
	`switch(conf-tm-dest)# ipv6 address 10:10::1 port 8000`
		発信データの接続先プロファイルを作成します。ここで *ip_version* は、ip（IPv4 の場合）または ipv6（IPv6 の場合）のいずれかです。
		接続先グループがサブスクリプションにリンクされている場合、テレメトリ データは、このプロファイルで指定されている IP アドレスとポートに送信されます。

ステップ 13	（任意）**use-chunking size** *chunking_size* 例：	gRPC チャンクを有効にして、チャンク サイズを 64～4096 バイトに設定します。詳細については、

	コマンドまたはアクション	目的
	`switch(conf-tm-dest)# use-chunking size 64`	「gRPCチャンクのサポート」セクションを参照してください。

ステップ 14	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 100`
	`switch(conf-tm-sub)#`
		ID を持つサブスクリプション ノードを作成し、サブスクリプション構成モードを開始します。
		現在、*sub_id* は、数字の ID 値のみをサポートしています。
		（注） DNにサブスクライブする場合は、イベントが確実にストリーミングされるように、その DN が REST を使用して DME でサポートされているかどうかを確認します。

ステップ 15	**snsr-grp** *sgrp_id* **sample-interval** *interval* 例： `switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000`	ID *sgrp_id* のセンサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。 間隔の値が 0 の場合、イベント ベースのサブスクリプションが作成され、テレメトリ データは、指定された MO での変更時にのみ送信されます。0 より大きい間隔値の場合、テレメトリ データが指定された間隔で定期的に送信される頻度に基いたサブスクリプションが作成されます。たとえば、間隔値が 15000 の場合、テレメトリ データは 15 秒ごとに送信されます。
ステップ 16	**dst-grp** *dgrp_id* 例： `switch(conf-tm-sub)# dst-grp 100`	ID *dgrp_id* を持つ接続先グループをこのサブスクリプションにリンクします。

YANG パスの頻度の設定

YANG パスの頻度は、合計ストリーミング時間よりも長くする必要があります。合計ストリーミング時間と頻度が正しく構成されていない場合、テレメトリ データの収集にストリーミング間隔よりも長くかかることがあります。この状況では、次のことがわかります。

- テレメトリ データが受信側へのストリーミングよりも速く蓄積されるため、徐々に満たされるキュー。
- 現在の間隔からではない古いテレメトリ データ。

合計ストリーミング時間よりも大きい値に頻度を構成します。

手順の概要

1. **show telemetry control database sensor-groups**
2. **sensor group** *number*
3. **subscription** *number*
4. **snsr-grp** *number* **sample-interval** *milliseconds*
5. **show system resources**

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**show telemetry control database sensor-groups**
	例：
	`switch# show telemetry control database sensor-groups`
	`Sensor Group Database size = 2`
	`----------`
	`Row ID Sensor Group ID Sensor Group type Sampling interval(ms) Linked subscriptions SubID`
	`----------`
	`1 2 Timer /YANG 5000 /Running 1 1`
	`Collection Time in ms (Cur/Min/Max): 2444/2294/2460`
	`Encoding Time in ms (Cur/Min/Max): 56/55/57`
	`Transport Time in ms (Cur/Min/Max): 0/0/1`
	`Streaming Time in ms (Cur/Min/Max): 2515/2356/28403`
	`Collection Statistics:`
	`collection_id_dropped = 0`
	`last_collection_id_dropped = 0`
	`drop_count = 0`
	`2 1 Timer /YANG 5000 /Running 1 1`
	`Collection Time in ms (Cur/Min/Max): 144/142/1471`
	`Encoding Time in ms (Cur/Min/Max): 0/0/1`
	`Transport Time in ms (Cur/Min/Max): 0/0/0`
	`Streaming Time in ms (Cur/Min/Max): 149/147/23548`
	`Collection Statistics:`
	`collection_id_dropped = 0`
	`last_collection_id_dropped = 0`
	`drop_count = 0`
	`switch#`
	`telemetry`
	`destination-group 1`
	`ip address 192.0.2.1 port 9000 protocol HTTP encoding JSON`
	`sensor-group 1`
	`data-source YANG`
	`path /Cisco-NX-OS-device:System/procsys-items`
		合計ストリーミング時間を計算します。
		合計ストリーミング時間は、各センサーグループの個々の現在のストリーミング時間の合計です。個々のストリーミング時間は、ミリ秒単位のストリーミング時間（Cur）に表示されます。この例では、合計ストリーミング時間は2.664秒（2515ミリ秒+149ミリ秒）です。
		構成された頻度をセンサーグループの合計ストリーミング時間と比較します。
		頻度はサンプル間隔で表示されます。この例では、合計ストリーミング時間（2.664秒）がケイデンス（デフォルトの5.000秒）よりも短いため、頻度は正しく構成されています。

	コマンドまたはアクション	目的
	depth unbounded sensor-group 2 data-source YANG path /Cisco-NX-OS-device:System/intf-items/phys-items depth unbounded subscription 1 dst-grp 1 snsr-grp 1 sample-interval 5000 **snsr-grp 2 sample-interval 5000**	
ステップ 2	**sensor group** *number* 例： switch(config-telemetry)# **sensor group*1***	合計ストリーミング時間がその頻度以上の場合、間隔を設定したいセンサーグループを入力します。
ステップ 3	**subscription** *number* 例： switch(conf-tm-sensor)# **subscription *100***	センサーグループのサブスクリプションを編集します。
ステップ 4	**snsr-grp** *number* **sample-interval** *milliseconds* 例： switch(conf-tm-sub)# **snsr-grp *number* sample-interval *5000***	適切なセンサーグループについて、サンプル間隔を合計ストリーミング時間よりも大きい値に設定します。 この例では、サンプル間隔は 5.000 秒に設定されています。これは、2.664 秒の合計ストリーミング時間よりも長いため、有効です。
ステップ 5	**show system resources** 例： switch# **show system resources** Load average: 1 minute: 0.38 5 minutes: 0.43 15 minutes: 0.43 Processes: 555 total, 3 running CPU states : **24.17% user**, 4.32% kernel, 71.50% idle CPU0 states: 0.00% user, 2.12% kernel, 97.87% idle CPU1 states: 86.00% user, 11.00% kernel, 3.00% idle CPU2 states: 8.08% user, 3.03% kernel, 88.88% idle CPU3 states: 0.00% user, 1.02% kernel, 98.97% idle Memory usage: 16400084K total, 5861652K used, 10538432K free Current memory status: OK	CPUの使用状況を確認してください。 この例に示すように、CPUユーザー状態が高い使用率を示している場合、頻度とストリーミング値が正しく構成されていません。この手順を繰り返して、頻度を正しく設定します。

CLI を使用したテレメトリの構成例

次の手順では、GPB エンコーディングを使用して 10 秒のリズムで単一のテレメトリ DME ストリームを構成する方法について説明します。

```
switch# configure terminal
switch(config)# feature telemetry
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(config-tm-dest)# ip address 171.70.59.62 port 50051 protocol gRPC encoding GPB
switch(config-tm-dest)# exit
switch(config-telemetry)# sensor group sg1
switch(config-tm-sensor)# data-source DME
switch(config-tm-dest)# path interface depth unbounded query-condition keep-data-type
switch(config-tm-dest)# subscription 1
switch(config-tm-dest)# dst-grp 1
switch(config-tm-dest)# snsr grp 1 sample interval 10000
```

この例では、sys/bgp ルート MO のデータを宛先 IP 1.2.3.4 ポート 50003 に 5 秒ごとにストリーミングするサブスクリプションを作成します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

次に、sys/intf のデータを 5 秒ごとに、宛先 IP 1.2.3.4 ポート 50003 にストリーミングし、test.pem を使用して検証された GPB エンコーディングを使用してストリームを暗号化するサブスクリプションの作成例を示します。

```
switch(config)# telemetry
switch(config-telemetry)# certificate /bootflash/test.pem foo.test.google.fr
switch(conf-tm-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
switch(config-dest)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

この例では、sys/cdp のデータを接続先 IP 1.2.3.4 ポート 50004 に 15 秒ごとにストリーミングするサブスクリプションを作成します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000
switch(conf-tm-sub)# dst-grp 100
```

この例では、750 秒ごとに **show** コマンド データのケイデンス ベースのコレクションを作成します。

```
switch(config)# telemetry
```

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ip address 172.27.247.72 port 60001 protocol gRPC encoding GPB
switch(conf-tm-dest)# sensor-group 1
switch(conf-tm-sensor# data-source NX-API
switch(conf-tm-sensor)# path "show system resources" depth 0
switch(conf-tm-sensor)# path "show version" depth 0
switch(conf-tm-sensor)# path "show environment power" depth 0
switch(conf-tm-sensor)# path "show environment fan" depth 0
switch(conf-tm-sensor)# path "show environment temperature" depth 0
switch(conf-tm-sensor)# path "show process cpu" depth 0
switch(conf-tm-sensor)# path "show nve peers" depth 0
switch(conf-tm-sensor)# path "show nve vni" depth 0
switch(conf-tm-sensor)# path "show nve vni 4002 counters" depth 0
switch(conf-tm-sensor)# path "show int nve 1 counters" depth 0
switch(conf-tm-sensor)# path "show policy-map vlan" depth 0
switch(conf-tm-sensor)# path "show ip access-list test" depth 0
switch(conf-tm-sensor)# path "show system internal access-list resource utilization"
depth 0
switch(conf-tm-sensor)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-dest)# snsr-grp 1 sample-interval 750000
```

この例では、sys/fm のイベント ベースのサブスクリプションを作成します。sys/fm MO に変更がある場合にのみ、データは接続先にストリーミングされます。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
```

動作中に、サンプル間隔を変更することで、センサー グループを周波数ベースからイベント ベースに変更したり、イベント ベースから周波数ベースに変更したりできます。この例では、センサー グループを前の例から頻度ベースに変更します。次のコマンドの後、テレメトリ アプリケーションは 7 秒ごとに sys/fm データの接続先へのストリーミングを開始します。

```
switch(config)# telemetry
switch(config-telemetry)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
```

複数のセンサー グループと接続先を 1 つのサブスクリプションにリンクできます。この例のサブスクリプションは、イーサネット ポート 1/1 のデータを 4 つの異なる接続先に 10 秒ごとにストリーミングします。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-sensor)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001 protocol HTTP encoding JSON
```

```
switch(conf-tm-dest)# ip address 1.4.8.2 port 60003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 10000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200
```

次に、センサー グループに複数のパスを含め、接続先グループに複数の接続先プロファイルを含め、サブスクリプションを複数のセンサー グループと宛先グループにリンクできる例を表示します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# path sys/epId-1 depth 0
switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0

switch(config-telemetry)# sensor-group 200
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# path sys/ipv4 depth 0

switch(config-telemetry)# sensor-group 300
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# path sys/bgp depth 0

switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 4.3.2.5 port 50005

switch(conf-tm-dest)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001

switch(conf-tm-dest)# destination-group 300
switch(conf-tm-dest)# ip address 1.2.3.4 port 60003

switch(conf-tm-dest)# subscription 600
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 200 sample-interval 20000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200

switch(conf-tm-dest)# subscription 900
switch(conf-tm-sub)# snsr-grp 200 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 300 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 300
```

この例に示すように、**show running-config telemetry** コマンドを使用してテレメトリ構成を確認できます。

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# end
switch# show run telemetry

!Command: show running-config telemetry
!Time: Thu Oct 13 21:10:12 2016
```

```
version 7.0(3)I5(1)
feature telemetry

telemetry
destination-group 100
ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
```

テレメトリの構成と統計情報の表示

次の NX-OS CLI **show** コマンドを使用して、テレメトリの構成、統計情報、エラー、およびセッション情報を表示します。

show telemetry yang direct-path cisco-nxos-device

このコマンドは、他のパスよりもパフォーマンスが向上するように直接エンコードされたYANGパスを表示します。

```
switch# show telemetry yang direct-path cisco-nxos-device
) Cisco-NX-OS-device:System/lldp-items
2) Cisco-NX-OS-device:System/acl-items
3) Cisco-NX-OS-device:System/mac-items
4) Cisco-NX-OS-device:System/intf-items
5) Cisco-NX-OS-device:System/procsys-items/sysload-items
6) Cisco-NX-OS-device:System/ospf-items
7) Cisco-NX-OS-device:System/procsys-items
8) Cisco-NX-OS-device:System/ipqos-items/queuing-items/policy-items/out-items
9) Cisco-NX-OS-device:System/mac-items/static-items
10) Cisco-NX-OS-device:System/ch-items
11) Cisco-NX-OS-device:System/cdp-items
12) Cisco-NX-OS-device:System/bd-items
13) Cisco-NX-OS-device:System/eps-items
14) Cisco-NX-OS-device:System/ipv6-items
```

show telemetry control database

次に、テレメトリの構成を反映している内部データベースのコマンドを表示します。

```
switch# show telemetry control database ?
  <CR>
  >                   Redirect it to a file
  >>                  Redirect it to a file in append mode
  destination-groups  Show destination-groups
  destinations        Show destinations
  sensor-groups       Show sensor-groups
  sensor-paths        Show sensor-paths
  subscriptions       Show subscriptions
  |                   Pipe command output to filter

switch# show telemetry control database

Subscription Database size = 1

------------------------------------------------------------------------------
Subscription ID      Data Collector Type
------------------------------------------------------------------------------
100                  DME NX-API
```

```
Sensor Group Database size = 1

------------------------------------------------------------------------------
Sensor Group ID  Sensor Group type  Sampling interval(ms)  Linked subscriptions
------------------------------------------------------------------------------
100              Timer             10000(Running)         1

Sensor Path Database size = 1

------------------------------------------------------------------------------
Subscribed Query Filter  Linked Groups  Sec Groups  Retrieve level  Sensor Path
------------------------------------------------------------------------------
No                       1              0           Full            sys/fm

Destination group Database size = 2

------------------------------------------------------------------------------
Destination Group ID  Refcount
------------------------------------------------------------------------------
100                   1

Destination Database size = 2

------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
------------------------------------------------------------------------------
192.168.20.111       12345      JSON       HTTP       1
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry control database sensor-paths

このコマンドは、テレメトリ設定のセンサーパスの詳細を表示します。これには、エンコーディング、収集、トランスポート、およびストリーミングのカウンタが含まれます。

```
switch(conf-tm-sub)# show telemetry control database sensor-paths
Sensor Path Database size = 4
------------------------------------------------------------------------------------------------------
Row ID     Subscribed Linked Groups  Sec Groups  Retrieve level  Path(GroupId) : Query
: Filter
------------------------------------------------------------------------------------------------------
1          No         1              0           Full            sys/cdp(1) : NA : NA

GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 65785/65785/65785
Collection Time in ms (Cur/Min/Max): 10/10/55
Encoding Time in ms (Cur/Min/Max): 8/8/9
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 18/18/65

2          No         1              0           Self            show module(2) : NA :
NA
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 1107/1106/1107
Collection Time in ms (Cur/Min/Max): 603/603/802
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/1
Streaming Time in ms (Cur/Min/Max): 605/605/803

3          No         1              0           Full            sys/bgp(1) : NA : NA
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
```

```
Collection Time in ms (Cur/Min/Max): 0/0/44
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 1/1/44

4          No         1              0           Self            show version(2) : NA :
 NA
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 2442/2441/2442
Collection Time in ms (Cur/Min/Max): 1703/1703/1903
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 1703/1703/1904

switch(conf-tm-sub)#
```

show telemetry control stats

このコマンドは、テレメトリの構成についての内部データベースの統計を表示します。

```
switch# show telemetry control stats
show telemetry control stats entered

------------------------------------------------------------------------------
Error Description                                               Error Count
------------------------------------------------------------------------------
Chunk allocation failures                                       0
Sensor path Database chunk creation failures                    0
Sensor Group Database chunk creation failures                   0
Destination Database chunk creation failures                    0
Destination Group Database chunk creation failures              0
Subscription Database chunk creation failures                   0
Sensor path Database creation failures                          0
Sensor Group Database creation failures                         0
Destination Database creation failures                          0
Destination Group Database creation failures                    0
Subscription Database creation failures                         0
Sensor path Database insert failures                            0
Sensor Group Database insert failures                           0
Destination Database insert failures                            0
Destination Group Database insert failures                      0
Subscription insert to Subscription Database failures           0
Sensor path Database delete failures                            0
Sensor Group Database delete failures                           0
Destination Database delete failures                            0
Destination Group Database delete failures                      0
Delete Subscription from Subscription Database failures         0
Sensor path delete in use                                       0
Sensor Group delete in use                                      0
Destination delete in use                                       0
Destination Group delete in use                                 0
Delete destination(in use) failure count                        0
Failed to get encode callback                                   0
Sensor path Sensor Group list creation failures                 0
Sensor path prop list creation failures                         0
Sensor path sec Sensor path list creation failures              0
Sensor path sec Sensor Group list creation failures             0
Sensor Group Sensor path list creation failures                 0
Sensor Group Sensor subs list creation failures                 0
Destination Group subs list creation failures                   0
Destination Group Destinations list creation failures           0
Destination Destination Groups list creation failures           0
Subscription Sensor Group list creation failures                0
```

```
Subscription Destination Groups list creation failures       0
Sensor Group Sensor path list delete failures                0
Sensor Group Subscriptions list delete failures              0
Destination Group Subscriptions list delete failures         0
Destination Group Destinations list delete failures          0
Subscription Sensor Groups list delete failures              0
Subscription Destination Groups list delete failures         0
Destination Destination Groups list delete failures          0
Failed to delete Destination from Destination Group          0
Failed to delete Destination Group from Subscription         0
Failed to delete Sensor Group from Subscription              0
Failed to delete Sensor path from Sensor Group               0
Failed to get encode callback                                0
Failed to get transport callback                             0
switch#  Destination Database size = 1

------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
------------------------------------------------------------------------------
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry data collector brief

このコマンドは、データ収集に関する簡略化した統計情報を表示します。

```
switch# show telemetry data collector brief

-----------------------------------------------------------------------
Collector Type       Successful Collections     Failed Collections
-----------------------------------------------------------------------
DME                   143                        0
```

show telemetry data collector details

このコマンドは、すべてのセンサーパスの詳細を含む、データ収集に関する詳細な統計情報を表示します。

```
switch# show telemetry data collector details

------------------------------------------------------------------------------
Succ Collections      Failed Collections     Sensor Path
------------------------------------------------------------------------------
150                   0                      sys/fm
```

show telemetry event collector errors

このコマンドは、イベント収集に関するエラー統計情報を表示します。

```
switch# show telemetry event collector errors

------------------------------------------------------------------------------
Error Description                                Error Count
------------------------------------------------------------------------------
APIC-Cookie Generation Failures                  - 0
Authentication Failures                          - 0
Authentication Refresh Failures                  - 0
```

```
Authentication Refresh Timer Start Failures        - 0
Connection Timer Start Failures                    - 0
Connection Attempts                                - 3
Dme Event Subscription Init Failures               - 0
Event Data Enqueue Failures                        - 0
Event Subscription Failures                        - 0
Event Subscription Refresh Failures                - 0
Pending Subscription List Create Failures          - 0
Subscription Hash Table Create Failures            - 0
Subscription Hash Table Destroy Failures           - 0
Subscription Hash Table Insert Failures            - 0
Subscription Hash Table Remove Failures            - 0
Subscription Refresh Timer Start Failures          - 0
Websocket Connect Failures                         - 0
```

show telemetry event collector stats

このコマンドは、すべてのセンサーパスの内訳を含むイベント収集に関する統計情報を表示します。

```
switch# show telemetry event collector stats

------------------------------------------------------------------------------
Collection Count  Latest Collection Time    Sensor Path
------------------------------------------------------------------------------
```

show telemetry control pipeline stats

このコマンドは、テレメトリ パイプラインの統計情報を表示します。

```
switch# show telemetry pipeline stats
Main Statistics:
    Timers:
        Errors:
            Start Fail        =     0

    Data Collector:
        Errors:
            Node Create Fail  =     0

    Event Collector:
        Errors:
            Node Create Fail  =     0    Node Add Fail     =     0
            Invalid Data      =     0



Queue Statistics:
    Request Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
```

```
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

    Data Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0
```

show telemetry transport

次に、構成されているすべての転送セッションの例を表示します。

```
switch# show telemetry transport

Session Id      IP Address      Port       Encoding   Transport  Status
--------------------------------------------------------------------------------
0               192.168.20.123   50001      GPB        gRPC       Connected
```

表 *1 : show telemetry transport* の構文の説明

構文	説明
show	実行中のシステム情報を表示します。
telemetry	テレメトリ情報を表示します
トランスポート	テレメトリのトランスポート情報を表示します。
session_id	（オプション）セッション ID
stats	（オプション）すべてのテレメトリ統計情報を表示します。
errors	（オプション）すべてのテレメトリ エラー情報を表示します。
読み取り専用	（オプション）
TABLE_transport_info	（オプション）トランスポート情報

構文	説明
session_idx	（オプション）セッション ID
ip_address	（オプション）トランスポート IP アドレス
port	（任意）トランスポート ポート
dest_info	（オプション）接続先情報
encoding_type	（オプション）エンコーディング タイプ
transport_type	（オプション）トランスポート タイプ
transport_status	（オプション）トランスポート ステータス
transport_security_cert_fname	（オプション）トランスポート セキュリティ ファイル名
transport_last_connected	（オプション）最後に接続されたトランスポート
transport_last_disconnected	（オプション）この宛先構成が最後に削除された時刻
transport_errors_count	（オプション）トランスポート エラー数
transport_last_tx_error	（オプション）トランスポートの最後の送信エラー
transport_statistics	（オプション）トランスポート統計情報
t_session_id	（オプション）トランスポート セッション ID
connect_statistics	（オプション）接続統計情報
connect_count	（オプション）接続数
last_connected	（オプション）最終接続のタイムスタンプ
Disconnect_count	（オプション）切断数
last_disconnected	（オプション）この宛先構成が最後に削除された時刻
trans_statistics	（オプション）トランスポート統計情報
compression	（オプション）圧縮ステータス
source_interface_name	（オプション）送信元インターフェイス名
source_interface_ip	（オプション）送信元インターフェイス IP

構文	説明
transmit_count	（オプション）送信数
last_tx_time	（オプション）最終送信時刻
min_tx_time	（オプション）最小送信時間
max_tx_time	（オプション）最大送信時間
avg_tx_time	（オプション）平均送信時間
cur_tx_time	（オプション）現在の送信時間
transport_errors	（オプション）トランスポート エラー
connect_errors	（オプション）接続エラー
connect_errors_count	（オプション）接続エラー数
trans_errors	（オプション）トランスポート エラー
trans_errors_count	（オプション）トランスポート エラー数
last_tx_error	（オプション）最後のトランスポートエラー
last_tx_return_code	（オプション）最後の送信戻りコード
transport_retry_stats	（オプション）再試行統計情報
ts_event_retry_bytes	（オプション）イベント再試行バッファ サイズ
ts_timer_retry_bytes	（オプション）タイマー再試行バッファ サイズ
ts_event_retry_size	（オプション）イベント再試行メッセージ数
ts_timer_retry_size	（オプション）タイマー再試行メッセージ数
ts_retries_sent	（オプション）送信された再試行回数
ts_retries_dropped	（オプション）ドロップされた再試行回数
event_retry_bytes	（オプション）イベント再試行バッファ サイズ
timer_retry_bytes	（オプション）タイマー再試行バッファ サイズ
retries_sent	（オプション）送信された再試行回数

構文	説明
retries_dropped	（オプション）ドロップされた再試行回数
retry_buffer_size	（オプション）再試行バッファ サイズ

show telemetry transport <session-id>

次のコマンドでは、特定の転送セッションの詳細なセッション情報が表示されます。

```
switch# show telemetry transport 0

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           gRPC
Status:              Disconnected
Last Connected:      Fri Sep 02 11:45:57.505 UTC

Tx Error Count:      224
Last Tx Error:       Fri Sep 02 12:23:49.555 UTC

switch# show telemetry transport 1

Session Id:          1
IP Address:Port      10.30.218.56:51235 Encoding:           JSON
Transport:           HTTP
Status:              Disconnected
Last Connected:      Never

Tx Error Count:      3
Last Tx Error:       Wed Apr 19 15:56:51.617 PDT
```

次に、IPv6 エントリの出力例を示します。

```
switch# show telemetry transport 0
Session Id: 0
IP Address:Port [10:10::1]:8000
Transport: GRPC
Status: Idle
Last Connected: Never
Last Disconnected: Never
Tx Error Count: 0
Last Tx Error: None
Event Retry Queue Bytes: 0
Event Retry Queue Size: 0
Timer Retry Queue Bytes: 0
Timer Retry Queue Size: 0
Sent Retry Messages: 0
Dropped Retry Messages: 0
```

show telemetry transport <session-id> stats

次に、特定の転送セッションの詳細のコマンドを示します。

```
switch# show telemetry transport 0 stats

Session Id:          0
```

```
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           GRPC
Status:              Connected
Last Connected:      Mon May 01 11:29:46.912 PST
Last Disconnected:   Never
Tx Error Count:      0
Last Tx Error:       None
```

show telemetry transport <session-id> errors

次のコマンドでは、特定の転送セッションの詳細なエラーの統計情報が表示されます。

```
switch# show telemetry transport 0 errors

Session Id:                    0
Connection Stats
   Connection Count            1
   Last Connected:             Mon May 01 11:29:46.912 PST
   Disconnect Count            0
   Last Disconnected:          Never
Transmission Stats
   Transmit Count:             1225
   Last TX time:               Tue May 02 11:40:03.531 PST
   Min Tx Time:                7                   ms
   Max Tx Time:                1760                ms
   Avg Tx Time:                500                 ms
```

show telemetry control databases sensor-paths

次の構成手順により、次の **show telemetry control databases sensor-paths** コマンド出力が得られます。

```
feature telemetry

telemetry
  destination-group 1
    ip address 172.25.238.13 port 50600 protocol gRPC encoding GPB
  sensor-group 1
    path sys/cdp depth unbounded
    path sys/intf depth unbounded
    path sys/mac depth 0
  subscription 1
    dst-grp 1
    snsr-grp 1 sample-interval 1000
```

コマンド出力。

```
switch# show telemetry control databases sensor-paths

Sensor Path Database size = 3
------------------------------------------------------------------------------
--------------------
Row ID     Subscribed Linked Groups  Sec Groups  Retrieve level  Path(GroupId) :
Query : Filter
------------------------------------------------------------------------------
--------------------
1          No         1              0           Full            sys/cdp(1) : NA
: NA
GPB Encoded Data size in bytes (Cur/Min/Max): 30489/30489/30489
```

```
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 6/5/54
Encoding Time in ms (Cur/Min/Max): 5/5/6
Transport Time in ms (Cur/Min/Max): 1027/55/1045
Streaming Time in ms (Cur/Min/Max): 48402/5/48402

2          No         1              0           Full            sys/intf(1) : N
A : NA
GPB Encoded Data size in bytes (Cur/Min/Max): 539466/539466/539466
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 66/64/114
Encoding Time in ms (Cur/Min/Max): 91/90/92
Transport Time in ms (Cur/Min/Max): 4065/4014/5334
Streaming Time in ms (Cur/Min/Max): 48365/64/48365

3          No         1              0           Self            sys/mac(1) : NA
: NA
GPB Encoded Data size in bytes (Cur/Min/Max): 247/247/247
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 1/1/47
Encoding Time in ms (Cur/Min/Max): 1/1/1
Transport Time in ms (Cur/Min/Max): 4/1/6
Streaming Time in ms (Cur/Min/Max): 47369/1/47369
```

show telemetry transport sessions

次のコマンドは、すべてのトランスポートセッションをループし、1つのコマンドで情報を出力します。

```
switch# show telemetry transport sessions
switch# show telemetry transport stats
switch# show telemetry transport errors
switch# show telemetry transport all
```

次に、テレメトリ トランスポート セッションの例を示します。

```
switch# show telemetry transport sessions
Session Id:          0
IP Address:Port      172.27.254.13:50004
Transport:           GRPC
Status:              Transmit Error
SSL Certificate:     trustpoint1
Last Connected:      Never
Last Disconnected:   Never
Tx Error Count:      2
Last Tx Error:       Wed Aug 19 23:32:21.749 UTC
…
Session Id:          4
IP Address:Port      172.27.254.13:50006
Transport:           UDP
```

テレメトリ エフェメラル イベント

エフェメラル イベントをサポートするために、新しいセンサー パス クエリ条件が追加されました。アカウンティング ログの外部イベント ストリーミングを有効にするには、次のクエリ条件を使用します。

```
sensor-group 1
path sys/accounting/log query-condition
query-target=subtree&complete-mo=yes&notify-interval=1
```

エフェメラル イベントをサポートするその他のセンサー パスは次のとおりです。

```
sys/pim/inst/routedb-route, sys/pim/pimifdb-adj, sys/pim/pimifdb-prop
sys/igmp/igmpifdb-prop, sys/igmp/inst/routedb, sys/igmpsnoop/inst/dom/db-exptrack,
sys/igmpsnoop/inst/dom/db-group, sys/igmpsnoop/inst/dom/db-mrouter
sys/igmpsnoop/inst/dom/db-querier, sys/igmpsnoop/inst/dom/db-snoop
```

テレメトリ ログとトレース情報の表示

ログとトレース情報を表示するには、次の NX-OS CLI コマンドを使用します。

テクニカル サポート テレメトリを表示

この NX-OS CLI コマンドは、テクニカル サポート ログからテレメトリ ログの内容を収集します。この例では、コマンド出力がブートフラッシュのファイルにリダイレクトされます。

```
switch# show tech-support telemetry > bootflash:tmst.log
```

NX-API を使用したテレメトリの構成

Configuring Telemetry Using the NX-API

In the object model of the switch DME, the configuration of the telemetry feature is defined in a hierarchical structure of objects as shown in the section "Telemetry Model in the DME." Following are the main objects to be configured:

- **fmEntity** — Contains the NX-API and Telemetry feature states.
  - **fmNxapi** — Contains the NX-API state.
  - **fmTelemetry** — Contains the Telemetry feature state.
- **telemetryEntity** — Contains the telemetry feature configuration.
  - **telemetrySensorGroup** — Contains the definitions of one or more sensor paths or nodes to be monitored for telemetry. The telemetry entity can contain one or more sensor groups.
    - **telemetryRtSensorGroupRel** — Associates the sensor group with a telemetry subscription.
    - **telemetrySensorPath** — A path to be monitored. The sensor group can contain multiple objects of this type.
  - **telemetryDestGroup** — Contains the definitions of one or more destinations to receive telemetry data. The telemetry entity can contain one or more destination groups.
    - **telemetryRtDestGroupRel** — Associates the destination group with a telemetry subscription.

- **telemetryDest** — A destination address. The destination group can contain multiple objects of this type.

- **telemetrySubscription** — Specifies how and when the telemetry data from one or more sensor groups is sent to one or more destination groups.
  - **telemetryRsDestGroupRel** — Associates the telemetry subscription with a destination group.
  - **telemetryRsSensorGroupRel** — Associates the telemetry subscription with a sensor group.
- **telemetryCertificate** — Associates the telemetry subscription with a certificate and hostname.

To configure the telemetry feature using the NX-API, you must construct a JSON representation of the telemetry object structure and push it to the DME with an HTTP or HTTPS POST operation.

Note For detailed instructions on using the NX-API, see the *Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference*.

Before you begin

Your switch must be configured to run the NX-API from the CLI:

```
switch(config)# feature nxapi

nxapi use-vrf vrf_name
nxapi http port port_number
```

Procedure

	Command or Action	Purpose
ステップ 1	Enable the telemetry feature. **Example:** `{ "fmEntity" : { "children" : [{ "fmTelemetry" : { "attributes" : { "adminSt" : "enabled" } } } ] } }`	The root element is **fmTelemetry** and the base path for this element is `sys/fm`. Configure the **adminSt** attribute as `enabled`.
ステップ 2	Create the root level of the JSON payload to describe the telemetry configuration. **Example:**	The root element is **telemetryEntity** and the base path for this element is `sys/tm`. Configure the **dn** attribute as `sys/tm`.

	Command or Action	Purpose

	<pre>{
	    "telemetryEntity": {
	        "attributes": {
	            "dn": "sys/tm"
	        },
	    }
	}</pre>	

ステップ 3	Create a sensor group to contain the defined sensor paths.
	Example:
	<pre>"telemetrySensorGroup": {
	    "attributes": {
	        "id": "10",
	        "rn": "sensor-10"
	    }, "children": [{
	    }]
	}</pre>
		A telemetry sensor group is defined in an object of class **telemetrySensorGroup**. Configure the following attributes of the object:
		• **id** — An identifier for the sensor group. Currently only numeric ID values are supported.
		• **rn** — The relative name of the sensor group object in the format: **sensor-***id*.
		• **dataSrc** — Selects the data source from **DEFAULT**, **DME**, **YANG**, or **NX-API**.
		Children of the sensor group object include sensor paths and one or more relation objects (**telemetryRtSensorGroupRel**) to associate the sensor group with a telemetry subscription.

ステップ 4	(Optional) Add an SSL/TLS certificate and a host.
	Example:
	<pre>{
	    "telemetryCertificate": {
	        "attributes": {
	            "filename": "root.pem"
	             "hostname": "c.com"
	         }
	    }
	}</pre>
		The **telemetryCertificate** defines the location of the SSL/TLS certificate with the telemetry subscription/destination.

ステップ 5	Define a telemetry destination group.
	Example:
	<pre>{
	      "telemetryDestGroup": {
	        "attributes": {
	          "id": "20"
	        }
	      }
	}</pre>
		A telemetry destination group is defined in **telemetryEntity**. Configure the id attribute.

ステップ 6	Define a telemetry destination profile.
	Example:
	<pre>{
	   "telemetryDestProfile": {
	      "attributes": {</pre>
		A telemetry destination profile is defined in **telemetryDestProfile**.
		• Configure the **adminSt** attribute as `enabled`.
		• Under **telemetryDestOptSourceInterface**, configure the **name** attribute with an interface name to stream

	Command or Action	Purpose

	<pre> "adminSt": "enabled"
	       },
	       "children": [
	          {
	             "telemetryDestOptSourceInterface":
	{
	                "attributes": {
	                   "name": "lo0"
	                }
	             }
	          }
	       ]
	    }
	}</pre>
		data from the configured interface to a destination with the source IP address.

ステップ 7	Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.
	Example:
	<pre>{
	    "telemetryDest": {
	        "attributes": {
	             "addr": "1.2.3.4",
	             "enc": "GPB",
	             "port": "50001",
	             "proto": "gRPC",
	             "rn": "addr-[1.2.3.4]-port-50001"
	        }
	    }
	}</pre>
		A telemetry destination is defined in an object of class **telemetryDest**. Configure the following attributes of the object:
		• **addr** — The IP address of the destination.
		• **port** — The port number of the destination.
		• **rn** — The relative name of the destination object in the format: **path-[***path***]**.
		• **enc** — The encoding type of the telemetry data to be sent. NX-OS supports:
		  • Google protocol buffers (GPB) for gRPC.
		  • JSON for C.
		• **proto** — The transport protocol type of the telemetry data to be sent. NX-OS supports:
		  • gRPC
		  • HTTP
		• Supported encoded types are:
		  • HTTP/JSON YES
		  • HTTP/Form-data YES Only supported for Bin Logging.
		  • GRPC/GPB-Compact YES Native Data Source Only.
		  • GRPC/GPB YES
		  • UDP/GPB YES
		  • UDP/JSON YES

	Command or Action	Purpose
ステップ 8	Enable gRPC chunking and set the chunking size, between 64 and 4096 bytes. **Example:** `{ "telemetryDestGrpOptChunking": { "attributes": { "chunkSize": "2048", "dn": "sys/tm/dest-1/chunking" } } }`	See Guidelines and Limitations section for more information.
ステップ 9	Create a telemetry subscription to configure the telemetry behavior. **Example:** `"telemetrySubscription": { "attributes": { "id": "30", "rn": "subs-30" }, "children": [{ }] }`	A telemetry subscription is defined in an object of class **telemetrySubscription**. Configure the following attributes of the object: • **id** — An identifier for the subscription. Currently only numeric ID values are supported. • **rn** — The relative name of the subscription object in the format: **subs-***id*. Children of the subscription object include relation objects for sensor groups (**telemetryRsSensorGroupRel**) and destination groups (**telemetryRsDestGroupRel**).
ステップ 10	Add the sensor group object as a child object to the **telemetrySubscription** element under the root element (**telemetryEntity**). **Example:** `{ "telemetrySubscription": { "attributes": { "id": "30" } "children": [{ "telemetryRsSensorGroupRel": { "attributes": { "sampleIntvl": "5000", "tDn": "sys/tm/sensor-10" } } } ] } }`	
ステップ 11	Create a relation object as a child object of the subscription to associate the subscription to the telemetry sensor group and to specify the data sampling behavior. **Example:**	The relation object is of class **telemetryRsSensorGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object:

	Command or Action	Purpose

	<pre>"telemetryRsSensorGroupRel": {
	    "attributes": {
	        "rType": "mo",
	        "rn":
	"rssensorGroupRel-[sys/tm/sensor-10]",
	        "sampleIntvl": "5000",
	        "tCl": "telemetrySensorGroup",
	        "tDn": "sys/tm/sensor-10",
	        "tType": "mo"
	    }
	}</pre>
		• **rn** — The relative name of the relation object in the format: **rssensorGroupRel-[sys/tm/***sensor-group-id***]**.
		• **sampleIntvl** — The data sampling period in milliseconds. An interval value of 0 creates an event-based subscription, in which telemetry data is sent only upon changes under the specified MO. An interval value greater than 0 creates a frequency-based subscription, in which telemetry data is sent periodically at the specified interval. For example, an interval value of 15000 results in the sending of telemetry data every 15 seconds.
		• **tCl** — The class of the target (sensor group) object, which is **telemetrySensorGroup**.
		• **tDn** — The distinguished name of the target (sensor group) object, which is **sys/tm/***sensor-group-id*.
		• **rType** — The relation type, which is **mo** for managed object.
		• **tType** — The target type, which is **mo** for managed object.

ステップ **12**	Define one or more sensor paths or nodes to be monitored for telemetry.
	Example:
	Single sensor path
	<pre>{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	                    }
	    }
	}</pre>**Example:**
	Multiple sensor paths
	<pre>{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",</pre>
		A sensor path is defined in an object of class **telemetrySensorPath**. Configure the following attributes of the object:
		• **path** — The path to be monitored.
		• **rn** — The relative name of the path object in the format: **path-[***path***]**
		• **depth** — The retrieval level for the sensor path. A depth setting of **0** retrieves only the root MO properties.
		• **filterCondition** — (Optional) Creates a specific filter for event-based subscriptions. The DME provides the filter expressions. For more information about filtering, see the Cisco APIC REST API Usage Guidelines on composing queries. You can find it at the following Cisco APIC documents landing page:

	Command or Action	Purpose

	<pre> "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	},
	{
	     "telemetrySensorPath": {
	         "attributes": {
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/dhcp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}</pre> **Example:** Single sensor path filtering for BGP disable events: <pre>{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition":
	"eq(fmBgp.operSt.\"disabled\")",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}</pre>	

ステップ **13**	Add sensor paths as child objects to the sensor group object (**telemetrySensorGroup**).	
ステップ **14**	Add destinations as child objects to the destination group object (**telemetryDestGroup**).	
ステップ **15**	Add the destination group object as a child object to the root element (**telemetryEntity**).	
ステップ **16**	Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription. **Example:**	The relation object is of class **telemetryRtSensorGroupRel** and is a child object of **telemetrySensorGroup**. Configure the following attributes of the relation object:

	Command or Action	Purpose

	<pre>"telemetryRtSensorGroupRel": {
	    "attributes": {
	        "rn":
	"rtsensorGroupRel-[sys/tm/subs-30]",
	        "tCl": "telemetrySubscription",
	        "tDn": "sys/tm/subs-30"
	    }
	}</pre>
		• **rn** — The relative name of the relation object in the format: **rtsensorGroupRel-[sys/tm/***subscription-id***]**.
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

ステップ **17**	Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.
	Example:
	<pre>"telemetryRtDestGroupRel": {
	    "attributes": {
	        "rn": "rtdestGroupRel-[sys/tm/subs-30]",
	
	        "tCl": "telemetrySubscription",
	        "tDn": "sys/tm/subs-30"
	    }
	}</pre>
		The relation object is of class **telemetryRtDestGroupRel** and is a child object of **telemetryDestGroup**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rtdestGroupRel-[sys/tm/***subscription-id***]**.
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

ステップ **18**	Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group.
	Example:
	<pre>"telemetryRsDestGroupRel": {
	    "attributes": {
	        "rType": "mo",
	        "rn": "rsdestGroupRel-[sys/tm/dest-20]",
	
	        "tCl": "telemetryDestGroup",
	        "tDn": "sys/tm/dest-20",
	        "tType": "mo"
	    }
	}</pre>
		The relation object is of class **telemetryRsDestGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rsdestGroupRel-[sys/tm/***destination-group-id***]**.
		• **tCl** — The class of the target (destination group) object, which is **telemetryDestGroup**.
		• **tDn** — The distinguished name of the target (destination group) object, which is **sys/tm/***destination-group-id*.
		• **rType** — The relation type, which is **mo** for managed object.
		• **tType** — The target type, which is **mo** for managed object.

ステップ **19**	Send the resulting JSON structure as an HTTP/HTTPS POST payload to the NX-API endpoint for telemetry configuration.	The base path for the telemetry entity is `sys/tm` and the NX-API endpoint is: `{{URL}}/api/node/mo/sys/tm.json`

Example

The following is an example of all the previous steps that are collected into one POST payload (note that some attributes may not match):

```
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50051",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "5000",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
```

```
  }
}
```

NX-API を使用したテレメトリの構成例

宛先へのストリーミング パス

この例では、パス sys/cdp および sys/ipv4 を接続先 1.2.3.4 ポート 50001 に 5 秒ごとにストリーミングするサブスクリプションを作成します。

```
POST https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
    "telemetryEntity": {
        "attributes": {
            "dn": "sys/tm"
        },
        "children": [{
            "telemetrySensorGroup": {
                "attributes": {
                    "id": "10",
                    "rn": "sensor-10"
                },  "children": [{
                    "telemetryRtSensorGroupRel": {
                        "attributes": {
                            "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
                            "tCl": "telemetrySubscription",
                            "tDn": "sys/tm/subs-30"
                        }
                    }
                }, {
                    "telemetrySensorPath": {
                        "attributes": {
                            "path": "sys/cdp",
                            "rn": "path-[sys/cdp]",
                            "excludeFilter": "",
                            "filterCondition": "",
                            "secondaryGroup": "0",
                            "secondaryPath": "",
                            "depth": "0"
                        }
                    }
                }, {
                    "telemetrySensorPath": {
                        "attributes": {
                            "path": "sys/ipv4",
                            "rn": "path-[sys/ipv4]",
                            "excludeFilter": "",
                            "filterCondition": "",
                            "secondaryGroup": "0",
                            "secondaryPath": "",
                            "depth": "0"
                        }
                    }
                }]
            }
        }, {
            "telemetryDestGroup": {
```

```
                "attributes": {
                    "id": "20",
                    "rn": "dest-20"
                },
                "children": [{
                    "telemetryRtDestGroupRel": {
                        "attributes": {
                            "rn": "rtdestGroupRel-[sys/tm/subs-30]",
                            "tCl": "telemetrySubscription",
                            "tDn": "sys/tm/subs-30"
                        }
                    }
                }, {
                    "telemetryDest": {
                        "attributes": {
                            "addr": "1.2.3.4",
                            "enc": "GPB",
                            "port": "50001",
                            "proto": "gRPC",
                            "rn": "addr-[1.2.3.4]-port-50001"
                        }
                    }
                }]
            }
        }, {
            "telemetrySubscription": {
                "attributes": {
                    "id": "30",
                    "rn": "subs-30"
                },
                "children": [{
                    "telemetryRsDestGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rsdestGroupRel-[sys/tm/dest-20]",
                            "tCl": "telemetryDestGroup",
                            "tDn": "sys/tm/dest-20",
                            "tType": "mo"
                        }
                    }
                }, {
                    "telemetryRsSensorGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
                            "sampleIntvl": "5000",
                            "tCl": "telemetrySensorGroup",
                            "tDn": "sys/tm/sensor-10",
                            "tType": "mo"
                        }
                    }
                }]
            }
        }]
    }
}
```

BGP 通知のフィルタ条件

次のペイロードの例では、`telemetrySensorPath` MO の `filterCondition` 属性に従って BFP 機能が無効になっているときにトリガーされる通知を有効にします。データは `10.30.217.80` ポート `50055` にストリーミングされます。

```
POST  https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50055",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "0",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
```

```
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
  }
}
```

テレメトリ構成のための Postman コレクションの使用

Postman コレクションの例は、テレメトリ機能の構成を開始する簡単な方法であり、1 つのペイロードですべてのテレメトリ CLI に相当するものを実行できます。好みのテキスト エディターを使用して前述のリンクのファイルを変更し、ペイロードをニーズに合わせて更新してから、Postman でコレクションを開いてコレクションを実行します。

DME のテレメトリ モデル

テレメトリ アプリケーションは、次の構造を持つ DME でモデル化されます。

```
model
|----package [name:telemetry]
    |  @name:telemetry
    |----objects
         |----mo [name:Entity]
              |    @name:Entity
              |      @label:Telemetry System
              |--property
              |    @name:adminSt
              |      @type:AdminState
              |
              |----mo [name:SensorGroup]
              |    |    @name:SensorGroup
              |    |      @label:Sensor Group
              |    |--property
              |    |    @name:id [key]
              |    |      @type:string:Basic
              |    |
              |    |----mo [name:SensorPath]
              |    |    |   @name:SensorPath
              |    |    |     @label:Sensor Path
              |    |    |--property
              |    |    |    @name:path [key]
              |    |    |      @type:string:Basic
              |    |    |    @name:filterCondition
              |    |    |      @type:string:Basic
              |    |    |    @name:excludeFilter
              |    |    |      @type:string:Basic
              |    |    |    @name:depth
              |    |    |      @type:RetrieveDepth
              |    |
              |----mo [name:DestGroup]
              |    |    @name:DestGroup
              |    |      @label:Destination Group
              |    |--property
```

```
|     |    @name:id
|     |      @type:string:Basic
|     |
|     |----mo [name:Dest]
|          |   @name:Dest
|          |     @label:Destination
|          |--property
|          |    @name:addr [key]
|          |      @type:address:Ip
|          |    @name:port [key]
|          |      @type:scalar:Uint16
|          |    @name:proto
|          |      @type:Protocol
|          |    @name:enc
|          |      @type:Encoding
|
|----mo [name:Subscription]
     |    @name:Subscription
     |      @label:Subscription
     |--property
     |    @name:id
     |      @type:scalar:Uint64
     |----reldef
     |    |  @name:SensorGroupRel
     |    |    @to:SensorGroup
     |    |    @cardinality:ntom
     |    |    @label:Link to sensorGroup entry
     |    |--property
     |         @name:sampleIntvl
     |           @type:scalar:Uint64
     |
     |----reldef
          |  @name:DestGroupRel
          |    @to:DestGroup
          |    @cardinality:ntom
          |    @label:Link to destGroup entry
```

マルチキャスト フロー パスの可視性

この機能は、Nexus 3548-XL スイッチで使用できる、必要なすべてのマルチキャスト状態をエクスポートする手段を提供します。エクスポートにより、各フローが送信元から各受信者までたどるパスの完全で信頼性の高いトレーサビリティが確保されます。

この機能は、DME ですべての適切な情報を公開することを目的としており、プッシュモデル（ソフトウェア テレメトリ）またはプルモデル（DME REST クエリ）のいずれかを介してコンシューマ/コントローラにアクセスできるようにします。

この機能の利点は次のとおりです。

- フロー パスの可視化
- 障害検出のためにフローの統計と状態のエクスポート
- ユーザーがフローパスに沿ったスイッチで適切なデバッグコマンドを実行できるようにすることによる根本原因の分析

MFDM はマルチキャスト FIB 分散管理の略で、上位レベルのコンポーネントからの情報を消費し、マルチキャスト機能ごとにインテリジェンスを構築してから、情報をコンシューマに伝達します。これは、機能が DME とともに実装されるコア コンポーネントです。MRIB によって提供される情報と MFIB によって収集された統計情報に基づいて、すべてのマルチキャスト状態を DME にパブリッシュします。

DME は、コンシューマ/コントローラが使用できるようにする必要があるすべての情報を保存するために使用されます。また、イベントベースの通知をサポートするため、オブジェクトが作成、削除、変更されるたびに、テレメトリへの適切な通知を生成します。

テレメトリ プロセスは、DME に保存されているすべてのデータをコンシューマにストリーミングし、データを適切な形式でフォーマットする役割を担います。

マルチキャスト フロー パスの可視性のための CLI

次に、マルチキャスト フロー パスの可視性の正確な機能を確認するために導入された CLI を示します。

- DME への情報のエクスポートを有効にする構成コマンド。この CLI は、システムに存在するすべてのルートに対してこの機能を有効にします。

```
switch(config)# multicast flow-path export
         switch(config)# sh system internal dme run all dn sys/mca/config
```

- MFDM と DME に存在する状態間の整合性チェックを実行する整合性チェッカーの show コマンド。このコマンドを使用すると、特に大規模なセットアップで不整合をすばやく検出できます。

```
switch# show forwarding distribution internal multicast consistency-checker flow-path
 route
Starting flow-path DME consistency-check for VRF: default
(0.0.0.0/0, 230.0.0.1/32).  Result: PASS
(10.0.0.10/32, 230.0.0.1/32).  Result: PASS
(0.0.0.0/0, 232.0.0.0/8).  Result: PASS
```

- グローバルな show コマンドは、この機能がシステムで有効になっているかどうかを確認するために使用します。

```
switch(config)# show forwarding distribution internal multicast global_state
 **** MFDM Flow PATH VISIBILITY INFO ****

 Multicast flow-path info export enabled: Y
 BE DME Handler: 0x117c3e6c
 PE DME Handler: 0x117b955c

switch(config)# show forwarding distribution internal multicast fpv CC
            PASS/FAIL (In case of fail, it will highlight the inconsistencies)
```

クラウド スケール ソフトウェア テレメトリ

クラウド スケール ソフトウェア テレメトリについて

NX-OS リリース 9.3(1) 以降、ソフトウェア テレメトリは、Tahoe ASIC を使用する Cisco Nexus クラウド スケール スイッチでサポートされます。このリリースで、サポートされているクラウド スケール スイッチは、ASIC と緊密に統合された TCP/IP サーバーをホストします。これにより、スイッチからのテレメトリ データのレポートをすばやく処理できます。サーバーは TCP ポート 7891 を使用します。テレメトリ クライアントはこのポートでサーバーに接続して、最大 10 ミリ秒でハードウェア カウンタ データを取得できます。

クラウド スケール ソフトウェア テレメトリには、独自のクライアント プログラムを作成したり、NX-OS リリース 9.3.1 以降にバンドルされているデフォルトのクライアント プログラムを使用したりする柔軟性があります。クライアント プログラムは、Python 2.7 以降、C、PHP など、TCP/IP をサポートする任意のプログラミング言語で作成できます。クライアント プログラムは、正しいメッセージ フォーマットで作成する必要があります。

NX-OS リリース 9.3(1) 以降、クラウド スケール ソフトウェア テレメトリ機能は NX-OS で使用できます。この機能はデフォルトで有効になっているため、NX-OS 9.3(1) 以降を実行しているサポート対象のスイッチでは、この機能を使用できます。

Cloud Scale ソフトウェア テレメトリ メッセージの形式

Cloud Scale テレメトリは、クライアントとスイッチ上の TCP/IP サーバー間のハンドシェイクで始まります。その間にクライアントは TCP ソケットを介して接続を開始します。クライアント メッセージは、32 ビット整数での 0 です。スイッチは、特定のファオーマットのカウンタ データを含むメッセージで応答します。

NX-OS リリース 9.3(1) では、次のメッセージ フォーマットがサポートされています。独自のクライアントプログラムを作成する場合は、クライアントが開始するメッセージがこのフォーマットに準拠していることを確認してください。

長さ	指定します。
4 バイト	ポート数、*N*

長さ	指定します。
56 バイト	各ポートのデータ、合計 56 * *N* バイト。 データの各 56 バイトチャンクは、次のもので構成されます。 • 24 バイトのインターフェイス名 • 8 バイトの送信（TX）パケット • 8 バイトの送信（TX）バイト • 8 バイトの受信（RX）パケット • 8 バイトの受信（RX）バイト

Guidelines and Limitations for Cloud Scale Software Telemetry

The following are the guidelines and limitations for the Cloud Scale software telemetry feature:

- For information about supported platforms for Cisco NX-OS prior to release 9.3(x), see the section for *Platform Support for Programmability Features* in that guide.
- For custom client telemetry programs, one message format is supported. Your client programs must comply with this format.

テレメトリ パス ラベル

テレメトリ パス ラベルについて

NX-OS リリース 9.3(1) 以降、モデル駆動型テレメトリはパス ラベルをサポートします。パス ラベルを使用すると、複数のソースからテレメトリ データを一度に簡単に収集できます。この機能では、収集するテレメトリ データのタイプを指定すると、テレメトリ機能によって複数のパスからそのデータが収集されます。次に、機能は情報を 1 つの統合された場所（パス ラベル）に返します。この機能により、次の作業が不要になるため、テレメトリの使用が簡素化されます。

- Cisco DME モデルに関する深く包括的な知識を持っています。
- 収集されるイベントの数と頻度のバランスを取りながら、複数のクエリを作成し、サブスクリプションに複数のパスを追加します。
- スイッチからテレメトリ情報の複数のチャンクを収集し、有用性を簡素化します。

パス ラベルは、モデル内の同じオブジェクト タイプの複数のインスタンスにわたり、カウンタまたはイベントを収集して返します。パス ラベルは、次のテレメトリ グループをサポートします。

- ファン、温度、電力、ストレージ、スーパーバイザ、ライン カードなどのシャーシ情報をモニタリングする環境。
- すべてのインターフェイス カウンターとステータスの変更をモニタリングするインターフェイス。

  このラベルは、**query-condition** コマンドを使用して返されるデータを絞り込むための定義済みのキーワード フィルタをサポートします。
- リソース。CPU 使用率やメモリ使用率などのシステム リソースをモニタリングします。
- VXLAN: VXLAN ピア、VXLAN カウンタ、VLAN カウンター、および BGP ピア データを含む VXLAN EVPN をモニタリングします。

データの投票またはイベントの受信

センサー グループのサンプル間隔によって、テレメトリ データがパス ラベルに送信される方法とタイミングが決まります。サンプル間隔は、テレメトリ データを定期的に投票するか、イベントが発生したときにテレメトリ データを収集するように構成できます。

- テレメトリのサンプル間隔がゼロ以外の値に設定されている場合、テレメトリは各サンプル間隔中に環境、インターフェイス、情報技術、および vxlan ラベルのデータを定期的に送信します。
- サンプル間隔がゼロに設定されている場合、環境、インターフェイス、情報技術、vxlan ラベルで動作状態の更新、および MO の作成と削除が発生するとテレメトリはイベント通知を送信します。

データの投票または受信イベントは相互に排他的です。パス ラベルごとに投票またはイベント駆動型テレメトリを構成できます。

パス ラベル注意事項と制約事項

テレメトリ パス ラベル機能には、次の注意事項と制約事項があります。

- この機能は、Cisco DME データ 送信元のみをサポートします。
- 同じセンサー グループ内の通常の DME パスとユーザビリティ パスを混在させて一致させることはできません。たとえば、`sys/intf` と `[インターフェイス（interface）]` を同じセンサー グループに構成することはできません。また、`sys/intf` と `[interface（インターフェイス）]` で同じセンサー グループを構成することはできません。この状況が発生した場合、NX-OS は構成を拒否します。
- `oper-speed` や `counters=[detailed]` などのユーザー フィルター キーワードは、`[インターフェイス（interface）]` パスに対してのみサポートされます。
- この機能は、`[深度（depth）]`や`[フィルター条件（filter-condition）]`などの他のセンサー パス オプションをサポートしていません。

- テレメトリ パス ラベルには、パス ラベルの使用に関する次の制限があります。
  - 大文字と小文字が区別されるため、小文字のプレフィックス **show** で開始する必要があります。

    例：**show version** は許可されます。ただし、**SHOW version** または version は使用できません。
  - 次の文字を含めることはできません。
    - ;
    - |
    - " " または ' '
  - 次の単語を含めることはできません。
    - telemetry
    - conf t
    - 設定

データまたはイベントをポーリングするためのインターフェイスパスの構成

インターフェイス パス ラベルは、すべてのインターフェイス カウンタとステータスの変更をモニタリングします。次のインターフェイス タイプをサポートします。

- 物理
- サブインターフェイス
- 管理
- ループバック
- VLAN
- ポート チャネル

インターフェイス パス ラベルを構成して、定期的にデータをポーリングするか、イベントを受信することができます。「データの投票またはイベントの受信 （48 ページ）」を参照してください。

（注） このモデルは、サブインターフェイス、ループバック、または VLAN のカウンタをサポートしていないため、ストリームアウトされません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path interface**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

ステップ 4	**path interface**
	例：
	`switch(conf-tm-sensor)# path interface`
	`switch(conf-tm-sensor)#`
		インターフェイス パス ラベルを構成して、複数の個々のインターフェイスに対して 1 つのテレメトリ データ クエリを送信できるようにします。ラベルは、複数のインターフェイスのクエリを 1 つに統合します。次に、テレメトリはデータを収集し、ラベルに返します。
		ポーリング間隔の設定方法に応じて、インターフェイスデータは定期的に、またはインターフェイスの状態が変化するたびに送信されます。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

	コマンドまたはアクション	目的

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定されたIPアドレスとポートにストリーミングします。

ステップ 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

非ゼロ カウンタのインターフェイス パスの構成

ゼロ以外の値を持つカウンターのみを返す事前定義されたキーワードフィルタを使用して、インターフェイス パス ラベルを構成できます。フィルタは `counters=[detailed]` です。

このフィルタを使用することにより、インターフェイス パスは使用可能なすべてのインターフェイス カウンターを収集し、収集したデータをフィルタ処理してから、結果を受信側に転送します。フィルタはオプションであり、使用しない場合、ゼロ値カウンターを含むすべてのカウンターがインターフェイス パスに表示されます。

（注） フィルタの使用は、概念的には **show interface mgmt0 counters detailed** と類似しています。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path interface query-condition counters=[detailed]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*

7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

ステップ 4	**path interface query-condition counters=[detailed]**
	例：
	`switch(conf-tm-sensor)# path interface query-condition counters=[detailed]`
	`switch(conf-tm-sensor)#`
		インターフェイス パス ラベルを構成し、すべてのインターフェイスからのゼロ以外のカウンターのみを照会します。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定されたIPアドレスとポートにストリーミングします。

ステップ 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプションサブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval* 例：	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチ

	コマンドまたはアクション	目的
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000` `switch(conf-tm-sub)#`	がテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

動作速度のインターフェイス パスの構成

指定された動作速度のインターフェイスのカウンタを返す定義済みのキーワードフィルタを使用して、インターフェイスパス ラベルを構成できます。フィルタは `oper-speed=[]` です。次の動作速度がサポートされています: auto、10M、100M、1G、10G、40G、200G、および 400G。

このフィルタを使用することにより、インターフェースパスは指定された速度のインターフェースのテレメトリ データを収集し、その結果を受信側に転送します。フィルタはオプションです。使用しない場合、動作速度に関係なく、すべてのインターフェイスのカウンタが表示されます。

フィルタは、複数の速度をコンマ区切りのリストとして受け入れることができます。たとえば、`oper-speed=[1G,10G]` は、1 および 10 Gbps で動作するインターフェイスのカウンタを取得します。区切り文字として空白を使用しないでください。

（注） インターフェイス タイプ サブインターフェイス、ループバック、および VLAN には動作速度プロパティがないため、フィルタはこれらのインターフェイス タイプをサポートしません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **snsr-group** *sgrp_id* **sample-interval** *interval*
4. **path interface query-condition oper-speed=[***speed***]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。

ステップ 4	**path interface query-condition oper-speed=[***speed***]**
	例：
	`switch(conf-tm-sensor)# path interface query-condition oper-speed=[1G,40G]`
	`switch(conf-tm-sensor)#`
		インターフェイス パス ラベルを設定し、指定された速度 (この例では 1 Gbps と 40 Gbps のみ) を実行しているインターフェイスからのカウンターを照会します。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定されたIPアドレスとポートにストリーミングします。

ステップ 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、イン

	コマンドまたはアクション	目的
		ターフェイスイベントが発生したときに送信するかを決定します。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

複数のクエリによるインターフェイス パスの構成

インターフェイス パス ラベルの同じクエリ条件に対して複数のフィルタを構成できます。その場合、使用する個々のフィルタは AND で結合されます。

クエリ条件の各フィルタは、コンマを使用して区切ります。query-condition には、任意の数のフィルタを指定できますが、追加するフィルタが多いほど、結果の焦点が絞られることに注意してください。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path interface query-condition counters=[detailed],oper-speed=[1G,40G]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id* 例：	テレメトリ データのセンサー グループを作成します。

	コマンドまたはアクション	目的
	`switch(config-telemetry)# sensor-group 6` `switch(conf-tm-sensor)#`	

ステップ 4	**path interface query-condition counters=[detailed],oper-speed=[1G,40G]**
	例：
	`switch(conf-tm-sensor)# path interface query-condition counters=[detailed],oper-speed=[1G,40G]`
	`switch(conf-tm-sensor)#`
		同じクエリで複数の条件を構成します。この例では、クエリは次の両方を実行します。
		• 1 Gbps で実行されているインターフェイスでゼロ以外のカウンターを収集して返します。
		• 40 Gbps で実行されているインターフェイスでゼロ以外のカウンターを収集して返します。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。

ステップ 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

データまたはイベントをポーリングするための環境パスの構成

環境パス ラベルは、ファン、温度、電源、ストレージ、スーパーバイザ、ラインカードなどのシャーシ情報をモニタリングします。テレメトリ データを定期的にポーリングするか、イベントが発生したときにデータを取得するように環境パスを構成できます。詳細については、データの投票またはイベントの受信 （48 ページ）を参照してください。

定期的なポーリングまたはイベントに基づいてシステム リソース情報を返すようにリソース パスを設定できます。このパスはフィルタリングをサポートしていません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path environment**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

ステップ 4	**path environment**
	例：
	`switch(conf-tm-sensor)# path environment`
	`switch(conf-tm-sensor)#`
		複数の個々の環境オブジェクトのテレメトリ データをラベルに送信できるようにする環境パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。
		サンプル間隔に応じて、環境データはポーリング間隔に基づいてストリーミングされるか、イベントが発生したときに送信されます。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

	コマンドまたはアクション	目的

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定されたIPアドレスとポートにストリーミングします。

ステップ 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプションサブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、環境イベントが発生したときに送信するかを決定します。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

イベントまたはデータをポーリングするためのリソース パスの構成

リソース パスは、CPU使用率やメモリ使用率などのシステム リソースをモニタリングします。このパスを構成して、テレメトリ データを定期的に収集するか、イベントが発生したときに収集できます。「データの投票またはイベントの受信 （48 ページ）」を参照してください。

このパスはフィルタリングをサポートしていません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path resources**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

ステップ 4	**path resources**
	例：
	`switch(conf-tm-sensor)# path resources`
	`switch(conf-tm-sensor)#`
		複数の個々のシステム リソースのテレメトリ データをラベルに送信できるようにするリソース パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。
		サンプル間隔に応じて、リソースデータはポーリング間隔に基づいてストリーミングされるか、システム メモリが「Not OK」に変更されたときに送信されます。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。

ステップ 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval* 例：	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチ

	コマンドまたはアクション	目的
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000` `switch(conf-tm-sub)#`	がテレメトリ データを定期的に送信するか、リソース イベントが発生したときに送信するかを決定します。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

イベントまたはデータをポーリングするための VXLAN パスの構成

vxlan パス ラベルは、VXLAN ピア、VXLAN カウンター、VLAN カウンター、BGP ピア データなど、スイッチの仮想拡張 LAN EVPN に関する情報を提供します。このパス ラベルを構成して、定期的に、またはイベントが発生したときにテレメトリ情報を収集できます。「データの投票またはイベントの受信 （48 ページ）」を参照してください。

このパスはフィルタリングをサポートしていません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **vxlan environment**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

	コマンドまたはアクション	目的

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

ステップ 4	**vxlan environment**
	例：
	`switch(conf-tm-sensor)# vxlan environment`
	`switch(conf-tm-sensor)#`
		複数の個々の VXLAN オブジェクトのテレメトリ データをラベルに送信できるようにする vxlan パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。サンプル間隔に応じて、VXLAN データはポーリング間隔に基づいてストリーミングされるか、イベントが発生したときに送信されます。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。

ステップ 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、VXLAN イベントが発生したときに送信するかを決定します。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

パス ラベル 構成 を確認

いつでも、パスラベルが構成されていることを確認し、実行中のテレメトリ構成を表示してその値を確認できます。

手順の概要

1. **show running-config-telemetry**

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**show running-config-telemetry**
	例：
	`switch(conf-tm-sensor)# show running-config telemetry`
	
	`!Command: show running-config telemetry`
	`!Running configuration last done at: Mon Jun 10 08:10:17 2019`
	`!Time: Mon Jun 10 08:10:17 2019`
	
	`version 9.3(1) Bios:version`
	`feature telemetry`
	
	`telemetry`
	`  destination-profile`
	`    use-nodeid tester`
	`    sensor-group 4`
	`      path interface query-condition and(counters=[detailed],oper-speed=[1G,10G])`
	`    sensor-group 6`
	`      path interface query-condition oper-speed=[1G,40G]`
	`    subscription 6`
	`      snsr-grp 6 sample-interval 6000`
	` nxosv2(conf-tm-sensor)#`
		テレメトリの現在の実行構成を表示します。
		この例では、センサー グループ 4 は、1 および 10 Gbps で実行されているインターフェイスからゼロ以外のカウンターを収集するように構成されています。センサー グループ 6 は、1 と 40 Gbps で実行されているインターフェイスからすべてのカウンターを収集するように構成されています。

パス ラベル情報の表示

パス ラベル表示コマンド

show telemetry usability コマンドを使用すると、クエリを発行したときにパス ラベルがたどる個々のパスを表示できます。

コマンド	表示内容
show telemetry usability {all | environment | interface | resources | vxlan}	すべてのパス ラベルのすべてのテレメトリ パス、または指定されたパス ラベルのすべてのテレメトリ パス。また、出力には、各パスが定期的なポーリングまたはイベントに基づいてテレメトリ データを報告するかどうかが示されます。 インターフェイス パス ラベルには、設定したキーワード フィルタまたはクエリ条件も含まれます。
show running-config telemetry	テレメトリと選択されたパス情報の実行構成。

コマンドの例

（注） **show telemetry usability all** コマンドは、このセクションに示されている個々のコマンドをすべて連結したものです。

show telemetry usability environment コマンドの例を次に示します。

```
switch# show telemetry usability environment
  1) label_name          : environment

     path_name           : sys/ch
     query_type          : poll
     query_condition     :
rsp-subtree=full&query-target=subtree&target-subtree-class=eqptPsuSlot,eqptFtSlot,eqptSupCSlot,eqptPsu,eqptFt,eqptSensor,eqptLCSlot


  2) label_name          : environment

     path_name           : sys/ch
     query_type          : event
     query_condition     :
[illegible]
switch#
```

show telemetry usability interface コマンドの出力を次に示します。

```
switch# show telemetry usability interface
  1) label_name          : interface

     path_name           : sys/intf
     query_type          : poll
     query_condition     :
query-target=children&query-target-filter=eq(l1PhysIf.adminSt,"up")&rsp-subtree=children&rsp-subtree-class=rmonEtherStats,rmonIfIn,rmonIfOut,rmonIfHCIn,rmonIfHCOut


  2) label_name          : interface

     path_name           : sys/mgmt-[mgmt0]
     query_type          : poll
     query_condition     :
```

query-target=subtree&query-target-filter=eq(mgmtMgmtIf.adminSt,"up")&rsp-subtree=full&rsp-subtree-class=rmonEtherStats,rmonIfIn,rmonIfOut,rmonIfHCIn,rmonIfHCOut

```
  3) label_name          : interface

     path_name           : sys/intf
     query_type          : event
     query_condition     :
[illegible]
ethpmEncRtdIf.operSt,"down")),and(updated(ethpmEncRtdIf.operSt),eq(ethpmEncRtdIf.operSt,"up"))))


  4) label_name          : interface

     path_name           : sys/mgmt-[mgmt0]
     query_type          : event
     query_condition     :
query-target=subtree&query-target-filter=or(or(deleted(),created()),or(and(updated(mgmtMgmtIf.operSt),eq(mgmtMgmtIf.operSt,"down")),and(updated(mgmtMgmtIf.operSt),eq(mgmtMgmtIf.operSt,"up"))))
switch#
```

show telemetry usability resources コマンドの例を次に示します。

```
switch# show telemetry usability resources
  1) label_name          : resources

     path_name           : sys/proc
     query_type          : poll
     query_condition     : rsp-subtree=full&rsp-foreign-subtree=ephemeral

  2) label_name          : resources

     path_name           : sys/procsys
     query_type          : poll
     query_condition     :
[illegible]


  3) label_name          : resources

     path_name           : sys/procsys/sysmem
     query_type          : event
     query_condition     :
query-target-filter=and(updated(procSysMem.memstatus),ne(procSysMem.memstatus,"OK"))

switch#
```

show telemetry usability vxlan コマンドの例を次に示します。

```
switch# show telemetry usability vxlan
  1) label_name          : vxlan

     path_name           : sys/bd
     query_type          : poll
     query_condition     : query-target=subtree&target-subtree-class=l2VlanStats

  2) label_name          : vxlan

     path_name           : sys/eps
     query_type          : poll
     query_condition     : rsp-subtree=full&rsp-foreign-subtree=ephemeral

  3) label_name          : vxlan

     path_name           : sys/eps
     query_type          : event
```

```
    query_condition     : query-target=subtree&target-subtree-class=nvoDyPeer

  4) label_name         : vxlan

    path_name           : sys/bgp
    query_type          : event
   query_condition    : query-target=subtree&query-target-filter=or(deleted(),created())


  5) label_name         : vxlan

    path_name           : sys/bgp
    query_type          : event
    query_condition     :
query-target=subtree&target-subtree-class=bgpDom,bgpPeer,bgpPeerAf,bgpDomAf,bgpPeerAfEntry,bgpOperRtctrlL3,bgpOperRtP,bgpOperRtEntry,bgpOperAfCtrl

switch#
```

ネイティブ データ送信元パス

ネイティブ データ送信元パスについて

NX-OSテレメトリは、特定のインフラストラクチャまたはデータベースに限定されないニュートラル データ 送信元であるネイティブ データ ソースをサポートします。代わりに、ネイティブ データ 送信元を使用すると、コンポーネントまたはアプリケーションをフックして、関連情報を発信テレメトリ ストリームに挿入できます。ネイティブ データ 送信元のパスはインフラストラクチャに属さないため、この機能は柔軟性を提供し、ネイティブアプリケーションはNX-OS テレメトリと対話できます。

ネイティブ データ 送信元 パスを使用すると、特定のセンサー パスに登録して、セレクトしたテレメトリ データを受信できます。この機能は NX-SDK と連携して、次のパスからのテレメトリ データのストリーミングをサポートします。

- IP ルートのテレメトリ データを送信する RIB パス。
- 静的および動的 MAC エントリのテレメトリ データを送信する MAC パス。
- IPv4 と IPv6 隣接のテレメトリ データを送信する隣接関係パス。

サブスクリプションを作成すると、選択したパスのすべてのテレメトリ データが基準値として受信者にストリーミングされます。基準値の後、イベント通知のみが受信者にストリーミングされます。

ネイティブ データ 送信元 パスのストリーミングは、次のエンコーディング タイプをサポートします：

- Google Protobuf（GPB）
- JavaScript Object Notation（JSON）
- コンパクト Google Protobuf (コンパクト GPB)

ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ

次の表は、各ソース パスについて、サブスクリプションが最初に作成されたとき（ベースライン）とイベント通知が発生したときにストリーミングされる情報を示しています。

Path Type	サブスクリプション ベースライン	イベント通知（**Event Notifications**）
RIB	全てのルートの送信	

Path Type	サブスクリプション ベースライン	イベント通知（Event Notifications）
		イベントの作成、更新、および削除に関するイベント通知を送信します。次の値は、RIB パスのテレメトリを介してエクスポートされます： • ネクスト ホップ ルーティング情報： • ネクスト ホップのアドレス • ネクスト ホップの発信インターフェイス • ネクスト ホップの VRF 名 • ネクスト ホップの所有者 • ネクスト ホップの優先度 • ネクスト ホップのメトリック • ネクスト ホップのタグ • ネクスト ホップのセグメント 識別子 • ネクスト ホップのトンネル 識別子 • ネクスト ホップのカプセル化タイプ • ネクスト ホップ タイプのフラグのビットごとの OR • レイヤ 3 のルーティング情報を検証する： • ルートの VRF 名 • ルート プレフィック

Path Type	サブスクリプション ベースライン	イベント通知（Event Notifications）
		ス アドレス • ルートのマスク長 • ルートのネクスト ホップ数 • イベントの種類 • ネクスト ホップ
MAC	静的およびダイナミック MAC エントリに対して DME から GETALL を実行します。	イベントの追加、更新および削除に関するイベント通知を送信します。次の値は、MAC パスのテレメトリを通じてエクスポートされます： • MAC アドレス（MAC address） • MAC アドレス タイプ • VLAN番号 • インターフェイス名 • イベント タイプ イベント通知では、静的エントリとダイナミック エントリの両方がサポートされています。

Path Type	サブスクリプション ベースライン	イベント通知（Event Notifications）
隣接	IPv4 および IPv6 隣接関係（アジャセンシー）を送信します。	イベントの追加、更新および削除に関するイベント通知を送信します。次の値は、隣接関係（アジャセンシー）パスのテレメトリを通じてエクスポートされます： • IP アドレス • MAC アドレス • インターフェイス名 • 物理インターフェイス名 • VRF 名 • プリファレンス • 隣接の送信元 • 隣接関係（アジャセンシー）のアドレス ファミリ • 隣接関係（アジャセンシー）のイベント タイプ

詳細については、Github https://github.com/CiscoDevNet/nx-telemetry-proto を参照してください。

注意事項と制約事項

ネイティブ データ 送信元 パス機能には、次の注意事項と制約事項があります。

- RIB、MAC、および隣接関係（アジャセンシー）のネイティブデータ送信元パスからのストリーミングの場合、センサー パス プロパティの更新は、**depth**、**query-condition**あるいは、**filter-condition**などのカスタム基準をサポートしません。

ルーティング情報のネイティブ データ送信元パスの構成

URIB に含まれるすべてのルートに関する情報を送信するルーティング情報のネイティブ データ送信元パスを構成できます。登録すると、基準値はすべてのルート情報を送信します。ベースラインの後、スイッチがサポートするルーティングプロトコルのルート更新と削除操作について通知が送信されます。RIB 通知で送信されるデータについては、ネイティブデータ送信元パス用にストリーミングされるテレメトリ データ （66 ページ） を参照してください。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path rib**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch(conf-tm-sub)# sensor-grp 6`
	`switch(conf-tm-sub)#`
		センサー グループを作成します。

ステップ 4	**data-source native**
	例：
	`switch(conf-tm-sensor)# data-source native`
	`switch(conf-tm-sensor)#`
		特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。

ステップ 5	**path rib**
	例：
	`nxosv2(conf-tm-sensor)# path rib`
	`nxosv2(conf-tm-sensor)#`
		ルートとルート アップデート情報をストリーミングする RIB パスを構成します。

	コマンドまたはアクション	目的

ステップ 6	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。

ステップ 8	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

MAC 情報のネイティブ データ送信元パスの構成

MAC テーブルのすべてのエントリに関する情報を送信する MAC 情報のネイティブ データ 送信元パスを構成できます。登録すると、基準値はすべての MAC 情報を送信します。基準値の後、MAC アドレスの追加、更新、および削除操作の通知が送信されます。MAC 通知で送信されるデータについては、ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ （66 ページ） を参照してください。

（注） 更新または削除イベントの場合、MAC 通知は、IP 隣接関係を持つ MAC アドレスに対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path mac**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch(conf-tm-sub)# sensor-grp 6`
	`switch(conf-tm-sub)#`
		センサー グループを作成します。

ステップ 4	**data-source native**
	例：
	`switch(conf-tm-sensor)# data-source native`
	`switch(conf-tm-sensor)#`
		特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。

	コマンドまたはアクション	目的

ステップ 5	**path mac**
	例：
	`nxosv2(conf-tm-sensor)# path mac`
	`nxosv2(conf-tm-sensor)#`
		MAC エントリおよび MAC 通知に関する情報をストリームする MAC パスを構成します。

ステップ 6	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。

ステップ 8	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプションサブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

すべての MAC 情報のネイティブ データ送信元パスの構成

レイヤ 3 およびレイヤ 2 から、MAC テーブルのすべてのエントリに関する情報を送信する MAC 情報のネイティブ データ 送信元 パスを構成できます。登録すると、基準値はすべての MAC情報を送信します。基準値の後、MACアドレスの追加、更新、および削除操作の通知が送信されます。MAC 通知で送信されるデータについては、ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ （66 ページ） を参照してください。

（注） 更新または削除イベントの場合、MAC 通知は、IP 隣接関係を持つ MAC アドレスに対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path mac-all**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id* 例：	センサー グループを作成します。

	コマンドまたはアクション	目的
	`switch(conf-tm-sub)# sensor-grp 6` `switch(conf-tm-sub)#`	

ステップ 4	**data-source native**
	例：
	`switch(conf-tm-sensor)# data-source native`
	`switch(conf-tm-sensor)#`
		特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。

ステップ 5	**path mac-all**
	例：
	`nxosv2(conf-tm-sensor)# path mac-all`
	`nxosv2(conf-tm-sensor)#`
		すべての MAC エントリおよび MAC 通知に関する情報をストリームする MAC パスを構成します。

ステップ 6	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。

ステップ 8	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

	コマンドまたはアクション	目的

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

IP 隣接のネイティブ データ パスの構成

スイッチのすべての IPv4 と IPv6 隣接に関する情報を送信する IP 隣接情報のネイティブ データ送信元パスを構成できます。登録すると、基準値はすべての隣接情報を送信します。基準値の後、隣接操作の追加、更新、および削除に関する通知が送信されます。隣接関係通知で送信されるデータについては、ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ （66 ページ） を参照してください。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path adjacency**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

	コマンドまたはアクション	目的

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch(conf-tm-sub)# sensor-grp 6`
	`switch(conf-tm-sub)#`
		センサー グループを作成します。

ステップ 4	**data-source native**
	例：
	`switch(conf-tm-sensor)# data-source native`
	`switch(conf-tm-sensor)#`
		ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。

ステップ 5	**path adjacency**
	例：
	`nxosv2(conf-tm-sensor)# path adjacency`
	`nxosv2(conf-tm-sensor)#`
		IPv4 と IPv6 隣接に関する情報をストリームする隣接パスを構成します。

ステップ 6	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。

ステップ 8	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、イン

	コマンドまたはアクション	目的
		ターフェイス イベントが発生したときに送信するかを決定します。

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

ネイティブ データ ソース パス情報の表示

NX-OS の **show telemetry event collector** コマンドを使用して、ネイティブ データ ソース パスの統計情報とカウンタ、またはエラーを表示できます。

統計情報の表示

show telemetry event collector stats コマンドを発行して、各ネイティブ データ ソース パスの統計情報とカウンタを表示できます。

RIB パスの統計情報の例：

```
switch# show telemetry event collector stats

------------------------------------------------------------------------------
Row ID         Collection Count  Latest Collection Time     Sensor Path(GroupId)
------------------------------------------------------------------------------
1              4                 Mon Jul 01 13:53:42.384 PST rib(1)
switch#
```

MAC パスの統計情報の例：

```
switch# show telemetry event collector stats

------------------------------------------------------------------------------
Row ID         Collection Count  Latest Collection Time     Sensor Path(GroupId)
------------------------------------------------------------------------------
1              3                 Mon Jul 01 14:01:32.161 PST mac(1)
switch#
```

隣接パスの統計情報の例：

```
switch# show telemetry event collector stats

------------------------------------------------------------------------------
Row ID         Collection Count  Latest Collection Time     Sensor Path(GroupId)
------------------------------------------------------------------------------
1              7                 Mon Jul 01 14:47:32.260 PST adjacency(1)
switch#
```

エラー カウンタの表示

show telemetry event collector stats コマンドを使用して、すべてのネイティブ データ ソース パスのエラーの合計を表示できます。

```
switch# show telemetry event collector errors

-------------------------------------------------------------------------------
-
Error Description                                    Error Count
-------------------------------------------------------------------------------
-
Dme Event Subscription Init Failures                 - 0
Event Data Enqueue Failures                          - 0
Event Subscription Failures                          - 0
Pending Subscription List Create Failures            - 0
Subscription Hash Table Create Failures              - 0
Subscription Hash Table Destroy Failures             - 0
Subscription Hash Table Insert Failures              - 0
Subscription Hash Table Remove Failures              - 0
switch#
```

ストリーミング Syslog

テレメトリ用のストリーミング Syslog について

Cisco NX-OS リリース 9.3(3) 以降、モデル駆動型テレメトリは、YANG をデータ ソースとして使用する syslogのストリーミングをサポートします。サブスクリプションを作成すると、すべての syslog が基準値として受信者にストリーミングされます。この機能は NX-SDK と連携して、次の syslog パスからのストリーミング syslog データをサポートします。

- Cisco-NX-OS-Syslog-oper:syslog
- Cisco-NX-OS-Syslog-oper:syslog/messages

基準値の後は、syslog イベント通知のみが受信者にストリーミングされます。syslog パスのストリーミングは、次のエンコーディング タイプをサポートします：

- Google Protobuf（GPB）
- JavaScript Object Notation（JSON）

Syslog 情報のための YANG データ ソース パスの構成

スイッチで生成されたすべての syslog に関する情報を送信する syslog の syslog パスを構成できます。サブスクライブすると、ベースラインはすべての既存の syslog 情報を送信します。ベースラインの後、通知は、スイッチで生成された新しい syslog に対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、**feature telemetry** コマンドで有効にします。

手順の概要

1. **configure terminal**

2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data source** *data-source-type*
5. **path Cisco-NX-OS-Syslog-oper:syslog/messages**
6. **destination-group** *grp_id*
7. ip address *ip_addr* **port** *port* **protocol** {**HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub-id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**configure terminal** 例： `switch# configure terminal`	グローバル コンフィギュレーション モードを開始します。
ステップ 2	**telemetry** 例： switch(config)# **telemetry**	テレメトリの構成モードに入ります。
ステップ 3	**sensor-group** *sgrp_id* 例： `switch(config-telemetry)# sensor-group 6`	センサー グループを作成します。
ステップ 4	**data source** *data-source-type* 例： `switch(config-tm-sensor)# data source YANG`	データソースを YANG に設定し、ネイティブ YANG ストリーミング モデルを使用して syslog をストリーミングできるようにします。
ステップ 5	**path Cisco-NX-OS-Syslog-oper:syslog/messages** 例： `switch(config-tm-sensor)# path Cisco-NX-OS-Syslog-oper:syslog/messages`	スイッチで生成された syslog をストリーミングする syslog パスを設定します。
ステップ 6	**destination-group** *grp_id* 例： `switch(config-tm-sensor)# destination-group 33`	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。
ステップ 7	ip address *ip_addr* **port** *port* **protocol** {**HTTP**	**gRPC** } **encoding** { **JSON**	**GPB**	**GPB-compact** } 例： `switch(config-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json` 例：	サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。

	コマンドまたはアクション	目的
	`switch(config-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`	
ステップ 8	**subscription** *sub-id* 例： `switch(config-tm-dest)# subscription 33`	テレメトリ サブスクリプションサブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval* 例： `switch(config-tm-sub)# snsr-group 6 sample-interval 0`	センサーグループを現在のサブスクリプションにリンクし、データ サンプリングを 0 に設定して、syslogイベントが発生したときにスイッチがテレメトリ データを送信するようにします。*interval* については、0 のみが受け入れ可能な値です。
ステップ 10	**dst-group** *dgrp_id* 例： `switch(config-tm-sub)# dst-grp 33`	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで構成した接続先グループとマッチする必要があります。

Syslog パスのテレメトリ データ ストリーミング

送信元パスごとに、次のテーブルは、サブスクリプションが最初に作成されるときの「ベースライン」において、そしてイベントの通知が発生するときに、どんな情報がストリーミングされるかを示しています。

パス	サブスクリプション ベースライン	イベント通知
Cisco-NX-OS-Syslog-oper:syslog/messages	スイッチから既存のすべての syslog をストリーミングします。	スイッチで発生した syslog のイベント通知を送信します。 • message-id • node-name • time-stamp • time-of-day • time-zone • category • message-name • severity • text

syslog パス情報の表示

syslog パスの統計情報とカウンタ、またはエラーを表示するには、Cisco NX-OS の **show telemetry event collector** コマンドを使用します。

統計情報の表示

show telemetry event collector stats コマンドを入力すると、syslog パスごとの統計情報とカウンタを表示できます。

次に、syslog パスの統計情報の例を示します。

```
switch# show telemetry event collector stats

------------------------------------------------------------------------------
Row ID         Collection Count  Latest Collection Time     Sensor Path(GroupId)
------------------------------------------------------------------------------
1              138               Tue Dec 03 11:20:08.200 PST
Cisco-NX-OS-Syslog-oper:syslog(1)
2              138               Tue Dec 03 11:20:08.200 PST
Cisco-NX-OS-Syslog-oper:syslog/messages(1)
```

エラー カウンタの表示

show telemetry event collector errors コマンドを使用すると、すべての syslog パスのエラーの合計を表示できます。

```
switch(config-if)# show telemetry event collector errors

------------------------------------------------------------------------------
Error Description                                 Error Count
------------------------------------------------------------------------------
Dme Event Subscription Init Failures                    - 0
Event Data Enqueue Failures                             - 0
Event Subscription Failures                             - 0
Pending Subscription List Create Failures               - 0
Subscription Hash Table Create Failures                 - 0
Subscription Hash Table Destroy Failures                - 0
Subscription Hash Table Insert Failures                 - 0
Subscription Hash Table Remove Failures                 - 0
```

JSON 出力の例

次に、JSON 出力のサンプルを示します。

```
172.19.216.13 - - [03/Dec/2019 19:38:50] "POST
/network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages HTTP/1.0" 200 -
172.19.216.13 - - [03/Dec/2019 19:38:50] "POST
/network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages HTTP/1.0" 200 -
>>> URL            : /network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages
>>> TM-HTTP-VER    : 1.0.0
>>> TM-HTTP-CNT    : 1
>>> Content-Type   : application/json
>>> Content-Length : 578
    Path => Cisco-NX-OS-Syslog-oper:syslog/messages
            node_id_str   : task-n9k-1
            collection_id : 40
```

```
            data_source   : YANG
            data          :
[
  [
    {
      "message-id": 420
    },
    {
      "category": "ETHPORT",
      "group": "ETHPORT",
      "message-name": "IF_UP",
      "node-name": "task-n9k-1",
      "severity": 5,
      "text": "Interface loopback10 is up ",
      "time-of-day": "Dec 3 2019 11:38:51",
      "time-stamp": "1575401931000",
      "time-zone": ""
    }
  ]
]
```

-

KVGPB の出力例

次に KVGPB の出力例を示します。

```
KVGPB Output:
---Telemetry msg received @ 18:22:04 UTC

Read frag:1 size:339 continue to block on read..

All the fragments:1 read successfully total size read:339

node_id_str: "task-n9k-1"

subscription_id_str: "1"

collection_id: 374

data_gpbkv {

  fields {

    name: "keys"

    fields {

      name: "message-id"

      uint32_value: 374

    }

  }

  fields {
```

```
name: "content"
fields {
  fields {
    name: "node-name"
    string_value: "task-n9k-1"
  }
  fields {
    name: "time-of-day"
    string_value: "Jun 26 2019 18:20:21"
  }
  fields {
    name: "time-stamp"
    uint64_value: 1574293838000
  }
  fields {
    name: "time-zone"
    string_value: "UTC"
  }
  fields {
    name: "process-name"
    string_value: ""
  }
  fields {
    name: "category"
    string_value: "VSHD"
  }
  fields {
    name: "group"
    string_value: "VSHD"
  }
  fields {
    name: "message-name"
```

```
        string_value: "VSHD_SYSLOG_CONFIG_I"
      }
      fields {
        name: "severity"
        uint32_value: 5
      }
      fields {
        name: "text"
        string_value: "Configured from vty by admin on console0"
      }
    }
  }
}
```

その他の参考資料

関連資料

関連項目	マニュアル タイトル
VXLAN EVPN のテレメトリ展開の構成例。	*[VXLAN EVPN ソリューションのテレメトリ展開（Telemetry Deployment for VXLAN EVPN Solution）]*

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。