gNMI プロトコル

gNMI プロトコルの機能により、gRPC ネットワーク管理インターフェイス（gNMI）の機能を使用したモデル駆動型の設定と運用データの取得、およびリモートプロシージャコール（RPC）の Get、Set、Subscribe 関数が記述されます。gNMI バージョン 0.4.0 がサポートされています。

gNMI プロトコルの制約事項

gNMI プロトコル機能には、次のような制約事項が適用されます。

- BYTES および ASCI エンコーディングオプションはサポートされていません。

  PROTO エンコーディングは、Cisco IOS XE Dublin 17.11.1 以降でサポートされています。

- JSON IETF キーには、子要素の名前空間が親とは異なる YANG プレフィックスが含まれている必要があります。たとえば、openconfig-vlan.yang の拡張から派生した routed-vlan は、親ノードの名前空間とは異なるため（親ノードはプレフィックス oc-if を持ちます）、*oc-vlan:routed-vlan* と入力する必要があります。

- GetRequest：

  - 運用データのフィルタリングはサポートされていません。

  - モデルの使用はサポートされていません。これらは、Get RPC コールへの応答として返す必要があるデータ要素を定義するスキーマ定義モジュールを示す一連のモデルデータ メッセージです。

- GetResponse：

- Alias はサポートされていません。これは、通知メッセージの中で指定されたプレフィックスのエイリアスを提供する文字列です。
- Delete はサポートされていません。これは、データツリーから削除する一連のパスです。

gNMI プロトコルの概要

gNMI について

gNMI は Google によって開発された gRPC ネットワーク管理インターフェイスです。gNMI はネットワークデバイスの設定をインストール、操作、および削除し、また、運用データの表示も実行するメカニズムです。gNMI を通じて提供されるコンテンツは YANG を使用してモデル化できます。

gRPC は、クラウドサーバと通信するモバイルクライアントを使用して低遅延で拡張可能な配布を実現するために Google によって開発されたリモート プロシージャ コールです。gRPC は gNMI を伝送し、データと動作要求を公式化して送信する手段を提供します。

gNMI サービスの障害が発生した場合、gNMI ブローカ（GNMIB）によって、up から down への動作状態の変化が示され、データベースが起動して実行されるまではすべての RPC がサービス利用不可のメッセージを返します。リカバリ時には、GNMIB によって down から up への動作状態の変化が示され、RPC の通常の処理が再開されます。

gNMI は<subscribe> RPC サービスをサポートします。詳細については、「モデル駆動型テレメトリ」の章を参照してください。

YANG データ ツリーの JSON IETF エンコーディング

RFC 7951 では、YANG データ ツリーとそのサブツリーの JavaScript オブジェクト表記（JSON）エンコーディングが規定されています。gNMI は、コンテンツ層でのデータのエンコードに JSON を使用します。

JSON タイプは、値が JSON 文字列としてエンコードされていることを示します。JSON_IETF でエンコードされたデータは、RFC 7951 で規定されている JSON シリアル化のルールに準拠している必要があります。クライアントとターゲットの両方が JSON エンコーディングをサポートしている必要があります。

YANG データ ノード（リーフ、コンテナ、リーフリスト、リスト、anydata ノード、および anyxml ノード）のインスタンスは、JSON オブジェクトまたは名前と値のペアのメンバーとしてエンコードされます。エンコーディング ルールは、設定データ、状態データ、RPC 操作のパラメータ、アクション、通知など、すべてのタイプのデータ ツリーで同じです。

データ ノード インスタンスはすべて名前と値のペアとしてエンコードされ、その名前はデータ ノード識別子から形成されます。値は、データ ノードのカテゴリによって異なります。

リーフデータノード

リーフ ノードは、データ ツリー内に値がありますが子はありません。リーフ インスタンスは、名前と値のペアとしてエンコードされます。この値には、リーフのタイプに応じて、文字列、数値、リテラル true または false、または特殊な配列 [null] を使用できます。指定されたパスのデータ項目がリーフ ノードの場合（子が存在せず、関連付けられた値を持つ）、そのリーフの値が直接エンコードされます（そのままの JSON 値が含まれています。JSON オブジェクトは必要ありません）。

次に、リーフ ノード定義の例を示します。

```
leaf foo {
 type uint8;
}
```

次に、JSON でエンコードされた有効なインスタンスを示します。

```
"foo": 123
```

PROTO エンコーディング

gNMI プロトコルは、すでにサポートされている JSON および JSON_IETF 形式に加えて PROTO エンコーディングもサポートします。*gnmi.proto* ファイルには、クライアント側とサーバー側で gNMI プロトコルのフレームワークを表す一連の手順を生成するためのブループリントが定義されています。

JSON および JSON_IETF 形式の既存のエンコーディング手順では、入力データがプッシュされるため、出力値で不完全なラッピングなどの問題が発生し、データの精度が失われます。

PROTO エンコーディングを使用すると、アプリケーションのクエリでデータをスカラー（TypedValue）値で取得できます。各リーフは、gNMI 仕様に従ってリーフ自体の更新時に送信されます。これにより、精度を高めるために浮動小数点値と倍精度浮動小数点値を使用できます。

PROTO エンコーディングは、gNMI でサポートされるすべてのパスに対して使用できます。PROTO エンコーディングは、subscribe RPC をサポートしますが、GET および SET RPC はサポートしません。

PROTO エンコーディングは gNMI プロトコルの一部であり、gNMI 機能が有効になっている場合に有効になります。

PROTO を使用した subscribeRequest RPC の例を以下に示します。

```
subscribe:
           prefix:
               origin: "legacy"
               elem:
                   name: "mdt-oper-v2:mdt-oper-v2-data"
               >
           >
           subscription:
             path:
              elem:
                  name: "mdt-subscriptions"
                  >
              >
```

```
            mode: SAMPLE
            sample_interval: 10000000000
        >
        mode: STREAM
        encoding: PROTO
    >
```

PROTO を使用した subscribeRequest 応答の例を示します。

```
update: <
  timestamp: 1652408966709576000
  prefix: <
    origin: "openconfig"
    elem: <
      name: "oc-if:interfaces"
    >
    elem: <
      name: "interface"
      key: <
        key: "name"
        value: "*"
      >
    >
    elem: <
      name: "state"
    >
  update: <
    path: <
      origin: "openconfig"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface"
        key: <
          key: "name"
          value: "GigabitEthernet3"
        >
      >
      elem: <
        name: "ethernet"
      >
      elem: <
        name: "config"
      >
    >
    val: <
      string_val: "{
\"mac-address\":\"00:0c:29:29:42:e9\”
  }"
    >
  >
update: <
    path: <
      origin: "openconfig"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface"
        key: <
          key: "name"
          value: "GigabitEthernet3"
        >
```

```
      >
      elem: <
        name: "ethernet"
      >
      elem: <
        name: "config"
      >
    >
    val: <
      bool_val: "{
\",\"auto-negotiate\":true
  }"
    >
  >
update: <
    path: <
      origin: "openconfig"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface"
        key: <
          key: "name"
          value: "GigabitEthernet3"
        >
      >
      elem: <
        name: "ethernet"
      >
      elem: <
        name: "config"
      >
    >
    val: <
      bool_val: "{
\"enable-flow-control\":true
  }"
    >
  >
```

エラーは、RPC 戻りメッセージの *status.proto* メッセージを使用して報告されます。

gNMI GET Request

gNMI Get RPC は、データ ツリーから、1 つ以上の設定属性、状態属性、派生状態属性、またはサポートされているモードに関連付けられたすべての属性を取得する方法を指定します。データ ツリーから値を取得するために、GetRequest がクライアントからターゲットに送信されます。GetRequest への応答として GetResponse が送信されます。

GetRequest の JSON 構造

次に、GetRequest JSON の構造の例を示します。GetRequest と GetResponse の両方が表示されます。

GetRequest

```
The following is a path for the
openconfig-interfaces model
```

```
++++++++ Sending get request: ++++++++
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
}
```

GetResponse

```
encoding: JSON_IETF
++++++++ Received get response: ++++++++
notification {
  timestamp: 1521699434792345469
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Loopback111\""
        }
      }
    }


    val {
      json_ietf_val: "{\n\t\"openconfig-interfaces:name\":\t\
      "Loopback111\",\n\t\
       "openconfig-interfaces:config\":\t{\n\t\t\
       "openconfig-interfaces:type\":\t\"ianaift:
        softwareLoopback\",\n\t\t\
       "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\
       "openconfig-interfaces:enabled\":\t\"true\"\n\t},\n\t\
       "openconfig-interfaces:state\":\t{\n\t\t\
       "openconfig-interfaces:type\":\t\"ianaift:
        softwareLoopback\",\n\t\t\
       "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\
       "openconfig-interfaces:enabled\":\t\"true\",\n\t\t\
       "openconfig-interfaces:ifindex\":\t52,\n\t\t\


       "openconfig-interfaces:admin-status\":\t\"UP\",\n\t\t\
       "openconfig-interfaces:oper-status\":\t\"UP\",\n\t\t\
       "openconfig-interfaces:last-change\":\t2018,\n\t\t\
       "openconfig-interfaces:counters\":\t{\n\t\t\t\
       "openconfig-interfaces:in-octets\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-unicast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-broadcast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-multicast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-discards\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-errors\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-unknown-protos\":\t0,\n\t\t\t\
       "openconfig-interfaces:out-octets\":\t0,\n\t\t\t\
       "openconfig-interfaces:out-unicast-pkts\":\t0,\n\t\t\t\
```

```
        "openconfig-interfaces:out-broadcast-pkts\":\t0,\n\t\t\t\
        "openconfig-interfaces:out-multicast-pkts\":\t0,\n\t\t\t\
        "openconfig-interfaces:out-discards\":\t0,\n\t\t\t\
        "openconfig-interfaces:out-errors\":\t0,\n\t\t\t\
        "openconfig-interfaces:last-clear\":\t2018\n\t\t},\n\t\t\

        "openconfig-platform:hardware-port\":\t\
        "Loopback111\"\n\t},\n\t\
        "openconfig-interfaces:subinterfaces\":\t{\n\t\t\
        "openconfig-interfaces:index\":\t0,\n\t\t\
        "openconfig-interfaces:config\":\t{\n\t\t\t\
        "openconfig-interfaces:index\":\t0,\n\t\t\t\
        "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\t\
        "openconfig-interfaces:enabled\":\t\"true\"\n\t\t},\n\t\t\
        "openconfig-interfaces:state\":\t{\n\t\t\t\
        "openconfig-interfaces:index\":\t0,\n\t\t\t\
        "openconfig-interfaces:name\":\t\"Loopback111.0\",\n\t\t\t\
        "openconfig-interfaces:enabled\":\t\"true\",\n\t\t\t\
        "openconfig-interfaces:admin-status\":\t\"UP\",\n\t\t\t\
        "openconfig-interfaces:oper-status\":\t\"UP\",\n\t\t\t\
        "openconfig-interfaces:last-change\":\t2018,\n\t\t\t\
        "openconfig-interfaces:counters\":\t{\n\t\t\t\t\
        "openconfig-interfaces:in-octets\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-unicast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-broadcast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-multicast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-discards\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-errors\":\t0,\n\t\t\t\t\

        "openconfig-interfaces:out-octets\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-unicast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-broadcast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-multicast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-discards\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-errors\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:last-clear\":\
         t2018\n\t\t\t\t}\n\t\t},\n\t\t\
        "openconfig-if-ip:ipv6\":\t{\n\t\t\t\
        "openconfig-if-ip:config\":\t\"false\",\n\t\t\t\
        "openconfig-if-ip:state\":\t\"false\"\n\t\t}\n\t}\n}"
    }
  }
}
```

リーフ値に対する GetRequest

次に、リーフ値に対するGetRequestの例を示します。GetRequest と、対応する GetResponse の両方が表示されます。

GetRequest

```
++++++++ Sending get request: ++++++++
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
```

```
      value: "Loopback111"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "oper-status"
  }
}
```

GetResponse

```
encoding: JSON_IETF
++++++++ Received get response: ++++++++
notification {
  timestamp: 1521699326012374332
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Loopback111\""
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      json_ietf_val: "\"UP\""
    }
  }
}
```

gNMI SetRequest

Set RPC は、サポートされているモデルに関連付けられた 1 つ以上の設定可能な属性を設定する方法を指定します。データツリー内の値を更新するために、SetRequest がクライアントからターゲットに送信されます。

SetRequest は JSON キーもサポートしており、キーには YANG プレフィックスが含まれている必要があります。このプレフィックスでは要素の名前空間が親とは異なります。

たとえば、openconfig-vlan.yang の拡張から派生した routed-vlan は、親ノードの名前空間とは異なるため（親ノードのプレフィックスは oc-if）、oc-vlan:routed-vlan と入力する必要があります。

1 つの SetRequest に含まれる削除、置換、および更新は、全体で 1 つのトランザクション セットとして扱われます。トランザクションのいずれかの下位要素で障害が発生した場合は、トラ

ンザクション全体が拒否されてロールバックされます。SetRequest に対して SetResponse が返信されます。

表 *1 : SetRequest* の *JSON* 構造の例

SetRequest	SetResponse

`++++++++ Sending set request: ++++++++`
`update {`
`  path {`
`  elem {`
`  name: "interfaces"`
`  }`
`  elem {`
`  name: "interface"`
`  key {`
`  key: "name"`
`  value: "Loopback111"`
`  }`
`  }`
`  elem {`
`  name: "config"`
`  }`
`  }`
`  val {`
`  json_ietf_val:`
`"{\"openconfig-interfaces:enabled\":\"false\"}"`
`  }`
`}`
	`++++++++ Received set response: ++++++++`
	`response {`
	`  path {`
	`  elem {`
	`  name: "interfaces"`
	`  }`
	`  elem {`
	`  name: "interface"`
	`  key {`
	`  key: "name"`
	`  value: "Loopback111"`
	`  }`
	`  }`
	`  elem {`
	`  name: "config"`
	`  }`
	`  }`
	`  op: UPDATE`
	`}`
	`timestamp: 1521699342123890045`

表 *2 :* リーフ値での *SetRequest* の例

SetRequest	SetResponse

`++++++++ Sending set request: ++++++++`
`update {`
`  path {`
`  elem {`
`  name: "interfaces"`
`  }`
`  elem {`
`  name: "interface"`
`  key {`
`  key: "name"`
`  value: "Loopback111"`
`  }`
`  }`
`  elem {`
`  name: "config"`
`  }`
`  elem {`
`  name: "description"`
`  }`
`  }`
`  val {`
`  json_ietf_val: "\"UPDATE DESCRIPTION\""`
`  }`
`}`
	`++++++++ Received set response: ++++++++`
	`response {`
	`  path {`
	`  elem {`
	`  name: "interfaces"`
	`  }`
	`  elem {`
	`  name: "interface"`
	`  key {`
	`  key: "name"`
	`  value: "Loopback111"`
	`  }`
	`  }`
	`  elem {`
	`  name: "config"`
	`  }`
	`  elem {`
	`  name: "description"`
	`  }`
	`  }`
	`  op: UPDATE`
	`}`
	`timestamp: 1521699342123890045`

gNMI の名前空間

名前空間は、メッセージの origin フィールドで使用されるパスプレフィックスを指定します。

ここでは、Cisco IOS XE Gibraltar 16.10.1 以降のリリースで使用される名前空間について説明します。

- RFC 7951 で指定された名前空間：パス プレフィックスは、RFC 7951 で定義されている YANG モジュール名を使用します。

  RFC 7951 で指定された値のプレフィックスは、YANG モジュール名を使用します。

  値のプレフィックスは、選択されたパスプレフィックスの名前空間の影響を受けません。次に、RFC 7951 で指定された値のプレフィックスの例を示します。

```
val {
   json_ietf_val:"{
     "openconfig-interfaces:config": {
       “openconfig-interfaces:description":
         “DESCRIPTION”
     }
   }"
 }
```

  RFC 7951 で指定された名前空間プレフィックスは、YANG モジュール名も使用します。たとえば、ループバック インターフェイスへの openconfig パスは次のようになります。

```
/openconfig-interfaces:interfaces/interface[name=Loopback111]/
```

  次の例は、RFC 7951 の名前空間指定を使用した gNMI パスを示しています。

```
path {
  origin: “rfc7951"
  elem {
    name: "openconfig-interface:interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
}
```

- openconfig：パスプレフィックスを使用しません。これらは openconfig モデルへのパスでのみ使用できます。

  openconfig 名前空間プレフィックスの動作は、発信元または名前空間が指定されていない場合と同じです。たとえば、ループバック インターフェイスへの openconfig パスは次のようになります。

```
/interfaces/interface[name=Loopback111]/
```

  次の例は、openconfig 名前空間指定を使用した gNMI パスを示しています。

```
path {
  origin: “openconfig"
  elem {
    name: "interfaces"
```

```
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
}
```

- 空：openconfig プレフィックスと同じです。これがデフォルトです。

  次の例は、空の openconfig 名前空間指定を使用した gNMI パスを示しています。

```
path {
elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
}
```

ここでは、Cisco IOS XE Gibraltar 16.10.1 より前のリリースで使用されるパス プレフィックスについて説明します。

ここでは、パス プレフィックスは、YANG モジュール定義で定義されている YANG モジュール プレフィックスを使用します。たとえば、ループバック インターフェイスへの openconfig パスは次のようになります。

```
/oc-if:interfaces/interface[name=Loopback111]/
```

次の例は、従来の名前空間指定を使用した gNMI パスを示しています。

```
path {
  origin: "legacy"
  elem {
    name: "oc-if:interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
}
```

gNMI のワイルドカード

gNMI プロトコルは、Get パスのワイルドカードをサポートしています。これは、複数の要素を一致させるためにパス内でワイルドカードを使用する機能です。これらのワイルドカードは、スキーマ内の指定されたサブツリーにあるすべての要素を示します。

elem は要素であり、xPath 内の / 文字の間の値です。elem は gNMI パスでも使用できます。たとえば、 elem 名を基準とするワイルドカードの位置は、ワイルドカードがインターフェイスを表し、すべてのインターフェイスとして解釈されることを暗に意味します。

ワイルドカードには暗黙的と明示的の2つのタイプがあり、どちらもサポートされています。Get パスは、パス ワイルドカードのすべてのタイプと組み合わせをサポートします。

- 暗黙的なワイルドカード：これらは、要素ツリー内の要素のリストを展開します。暗黙的なワイルドカードは、リストの要素にキー値が指定されていない場合に出現します。

  次に、パスの暗黙的なワイルドカードの例を示します。このワイルドカードは、デバイスにあるすべてのインターフェイスの説明を返します。

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
  }
  elem {
    name: "config"
  }
  elem {
    name: "description"
  }
}
```

- 明示的なワイルドカード：下記の指定によって同じ機能を提供します。
  - パス要素名またはキー名のいずれかにアスタリスク（*）を指定します。

    次に、パスのアスタリスクワイルドカードをキー名として使用する例を示します。このワイルドカードは、デバイスにあるすべてのインターフェイスの説明を返します。

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
     key: "name"
      value: "*"
    }
  }
  elem {
    name: "config"
  }
  elem {
    name: "description"
  }
}
```

    次に、パスのアスタリスク ワイルドカードをパス名として使用する例を示します。このワイルドカードは、Loopback111 インターフェイスで使用可能なすべての要素の説明を返します。

```
path {
  elem {
    name: "interfaces"
```

```
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
   elem {
    name: "*"
  }
  elem {
    name: "description"
  }
}
```

- 要素名として省略記号（...）または空のエントリを指定します。これらのワイルドカードは、パス内の複数の要素に展開できます。

  次に、パスの省略記号ワイルドカードの例を示します。このワイルドカードは、/interfaces 配下で使用可能なすべての説明フィールドを返します。

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "..."
  }
  elem {
    name: "description"
  }
}
```

次に、暗黙的なワイルドカードを使用した GetRequest の例を示します。この GetRequest は、デバイスにあるすべてのインターフェイスの oper-status を返します。

```
path {
  elem {
    name: "interfaces"
    }
  elem {
    name: "interface"
  }
  elem {
    name: "state"
  }
  elem {
    name: "oper-status"
  }
},
type: 0,
encoding: 4
```

次に、暗黙的なワイルドカードを使用した GetResponse の例を示します。

```
notification {
  timestamp: 1520627877608777450
  update {
```

```
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"FortyGigabitEthernet1/1/1\""
        }
      }
      elem {
        name: "state"
}
      elem {
        name: "oper-status"
      }
    }
    val {
      json_ietf_val: "\"LOWER_LAYER_DOWN\""
    }
  },

<snip>
…
</snip>

update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Vlan1\""
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      json_ietf_val: "\"DOWN\""
    }
  }
}
```

gNMI 設定の永続化

gNMI 設定の永続化機能により、gNMI SetRequest RPC によって行われたすべての正常な設定変更が、デバイスの再起動後も設定に保持されるようになります。この機能が導入される前は、gNMI 設定はデバイスの実行コンフィギュレーションに保存されていました。また、変更は **write memory** コマンドまたは SaveConfig NETCONF RPC の発行によって保存されていました。

実行コンフィギュレーションのすべての変更は、gNMI 以外の処理によって変更されたデータであっても、SetRequest RPC が発行されるとスタートアップ コンフィギュレーションにデータが保存されます。

この機能はデフォルトで有効であり、無効にすることはできません。

gNMI ユーザ名とパスワードによる認証

ユーザログイン情報、ユーザ名、およびパスワードは、各 gNMI RPC でメタデータとして承認を提供します。次に、ユーザ名とパスワードを使用するサンプル gNMI 機能 RPC を示します。

```
metadata = [('username','admin'), ('password','lab')]
cap_request = gnmi_pb2.CapabilityRequest()
# pass metadata to the gnmi_pb2_grpc.gNMIStub object
secure_stub.Capabilities(cap_request, metadata=metadata)
```

gNMI のエラー メッセージ

エラーが発生すると、gNMI は説明的なエラー メッセージを返します。次のセクションでは gNMI エラー メッセージをいくつか示します。

次に、パスが無効な場合に表示されるエラー メッセージの例を示します。

```
gNMI Error Response:
  <_Rendezvous of RPC that terminated with (StatusCode.TERMINATED,
   An error occurred while parsing provided xpath: unknown tag:
   "someinvalidxpath" Additional information: badly formatted or nonexistent path)>
```

次に、非実装エラーが発生した場合に表示されるエラー メッセージの例を示します。

```
gNMI Error Response:
  <_Rendezvous of RPC that terminated with (StatusCode.UNIMPLEMENTED,
    Requested encoding "ASCII" not supported)>
```

次に、データ要素が空の場合に表示されるエラー メッセージの例を示します。

```
gNMI Error Response:
  <_Rendezvous of RPC that terminated with (StatusCode.NOT_FOUND,
   Empty set returned for path "/oc-if:interfaces/noinfohere")>
```

gNMI プロトコルを有効にする方法

gNMI プロトコル を有効にするには、次の手順を実行します。

1. gNMI クライアントと、認証局（CA）によって署名されたデバイス用に一連の証明書を作成します。

1. Linux で OpenSSL を使用して証明書を作成します。
2. デバイスに証明書をインストールします。
3. デバイスで gNMI を設定します。
4. gNMI が有効になっていて実行されているかどうかを確認します。

2. 前の手順で設定したクライアント証明書とルート証明書を使用して gNMI クライアントを接続します。

Linux での OpenSSL を使用した証明書の作成

証明書とトラストポイントは、セキュア gNMI サーバにのみ必要です。

次に、Linux マシン上で OpenSSL を使用して証明書を作成する例を示します。

```
# Setting up a CA
openssl genrsa -out rootCA.key 2048
openssl req -subj /C=/ST=/L=/O=/CN=rootCA -x509 -new -nodes -key rootCA.key -sha256 -out
 rootCA.pem

# Setting up device cert and key
openssl genrsa -out device.key 2048
openssl req -subj /C=/ST=/L=/O=/CN=<hostnameFQDN> -new -key device.key -out device.csr
openssl x509 -req -in device.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
device.crt -sha256
# Encrpyt device key - needed for input to IOS
openssl rsa -des3 -in device.key -out device.des3.key -passout pass:<password - remember
 this for later>

# Setting up client cert and key
openssl genrsa -out client.key 2048
openssl req -subj /C=/ST=/L=/O=/CN=gnmi_client -new -key client.key -out client.csr
openssl x509 -req -in client.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
client.crt -sha256
```

CLI によるデバイスへの証明書のインストール

次の例は、デバイスに証明書をインストールする方法を示しています。

```
# Send:
Device# configure terminal
Device(config)# crypto pki import trustpoint1 pem terminal password password1

# Receive:
% Enter PEM-formatted CA certificate.
% End with a blank line or "quit" on a line by itself.

# Send:
# Contents of rootCA.pem, followed by newline + 'quit' + newline:
-----BEGIN CERTIFICATE-----
<snip>
-----END CERTIFICATE-----
quit
```

```
# Receive:
% Enter PEM-formatted encrypted private General Purpose key.
% End with "quit" on a line by itself.

# Send:
# Contents of device.des3.key, followed by newline + 'quit' + newline:
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,D954FF9E43F1BA20
<snip>
-----END RSA PRIVATE KEY-----
quit

# Receive:
% Enter PEM-formatted General Purpose certificate.
% End with a blank line or "quit" on a line by itself.

# Send:
# Contents of device.crt, followed by newline + 'quit' + newline:
-----BEGIN CERTIFICATE-----
<snip>
-----END CERTIFICATE-----
quit

# Receive:
% PEM files import succeeded.
Device(config)#

# Send:
Device(config)# crypto pki trustpoint trustpoint1
Device(ca-trustpoint)# revocation-check none
Device(ca-trustpoint)# end
Device#
```

非セキュア モードでの gNMI の有効化

（注） このタスクは、Cisco IOS XE Amsterdam 17.3.1 以降のリリースに適用されます。

[Day Zero setup] で、最初にデバイスを非セキュア モードで有効にしてから、デバイスを無効にし、セキュア モードを有効にします。インセキュアモードで gNxI を停止するには、**no gnxi server** コマンドを使用します。

（注） gNxI 非セキュアサーバとセキュアサーバはデバイス上で同時に実行できます。

（注） **gnxi** コマンドは、gNMI および gRPC ネットワーク操作インターフェイス（gNOI）サービスの両方に適用されます。gNxI ツールは、gNMI および gNOI プロトコルを使用するネットワーク管理用ツールのコレクションです。

手順の概要

1. **enable**
2. **configure terminal**
3. **gnxi**
4. **gnxi server**
5. **gnxi port** *port-number*
6. **end**
7. **show gnxi state**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**enable** 例： `Device> enable`	特権 EXEC モードを有効にします。 • パスワードを入力します（要求された場合）。
ステップ 2	**configure terminal** 例： `Device# 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 3	**gnxi** 例： `Device(config)# gnxi`	gNxl プロセスを起動します。
ステップ 4	**gnxi server** 例： `Device(config)# gnxi server`	gNxI サーバを非セキュアモードで有効にします。
ステップ 5	**gnxi port** *port-number* 例： `(Optional) Device(config)# gnxi port 50000`	リッスンする gNxI ポートを設定します。 • デフォルトの非セキュア gNxI ポートは 50052 です。
ステップ 6	**end** 例： `Device(config)# end`	グローバル コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。
ステップ 7	**show gnxi state** 例： `Device# show gnxi state`	gNxI インターフェイスのステータスが表示されます。

セキュア モードでの gNMI の有効化

（注） このタスクは、Cisco IOS XE Amsterdam 17.3.1 以降のリリースに適用されます。

セキュアモードで gNxI を停止するには、**no gnxi secure-server** コマンドを使用します。

（注） gNxI 非セキュアサーバとセキュアサーバはデバイス上で同時に実行できます。

（注） **gnxi** コマンドは、gNMI および gRPC ネットワーク操作インターフェイス（gNOI）サービスの両方に適用されます。gNxI ツールは、gNMI および gNOI プロトコルを使用するネットワーク管理用ツールのコレクションです。

手順の概要

1. **enable**
2. **configure terminal**
3. **gnxi**
4. **gnxi secure-trustpoint** *trustpoint-name*
5. **gnxi secure-server**
6. **gnxi secure-client-auth**
7. **gnxi secure-port**
8. **end**
9. **show gnxi state**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**enable** 例： `Device> enable`	特権 EXEC モードを有効にします。 • パスワードを入力します（要求された場合）。
ステップ 2	**configure terminal** 例： `Device# 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 3	**gnxi** 例： `Device(config)# gnxi`	gNxl プロセスを起動します。

	コマンドまたはアクション	目的
ステップ 4	**gnxi secure-trustpoint** *trustpoint-name* 例： `Device(config)# gnxi secure-trustpoint trustpoint1`	gNxI が認証に使用するトラストポイントと証明書セットを指定します。
ステップ 5	**gnxi secure-server** 例： `Device(config)# gnxi secure-server`	gNxI サーバをセキュアモードで有効にします。
ステップ 6	**gnxi secure-client-auth** 例： `Device(config)# gnxi secure-client-auth`	（任意）gNxIプロセスは、ルート証明書と照合してクライアント証明書を認証します。
ステップ 7	**gnxi secure-port** 例： `Device(config)# gnxi secure-port`	（任意）リッスンする gNxI ポートを設定します。 • デフォルトのセキュア gNxI ポートは9339です。
ステップ 8	**end** 例： `Device(config)# end`	グローバル コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。
ステップ 9	**show gnxi state** 例： `Device# show gnxi state`	gNxI サーバのステータスを表示します。

例

次に、**show gnxi state** コマンドの出力例を示します。

```
Device# show gnxi state

State          Status
--------------------------------
Enabled        Up
```

gNMI クライアントの接続

以前に設定したクライアント証明書とルート証明書を使用して gNMI クライアントが接続されます。

次に、Python を使用して gNMI クライアントを接続する例を示します。

```
# gRPC Must be compiled in local dir under path below:
>>> import sys
```

```
>>> sys.path.insert(0, "reference/rpc/gnmi/")
>>> import grpc
>>> import gnmi_pb2
>>> import gnmi_pb2_grpc
>>> gnmi_dir = '/path/to/where/openssl/creds/were/generated/'

# Certs must be read in as bytes
>>> with open(gnmi_dir + 'rootCA.pem', 'rb') as f:
>>>     ca_cert = f.read()
>>> with open(gnmi_dir + 'client.crt', 'rb') as f:
>>>     client_cert = f.read()
>>> with open(gnmi_dir + 'client.key', 'rb') as f:
>>>     client_key = f.read()

# Create credentials object
>>> credentials = grpc.ssl_channel_credentials(root_certificates=ca_cert,
private_key=client_key, certificate_chain=client_cert)

# Create a secure channel:
# Default port is 9339, can be changed on ios device with 'gnxi secure-port ####'
>>> port = 9339
>>> host = <HOSTNAME FQDN>
>>> secure_channel = grpc.secure_channel("%s:%d" % (host, port), credentials)

# Create secure stub:
>>> secure_stub = gnmi_pb2_grpc.gNMIStub(secure_channel)

# Done! Let's test to make sure it works:
>>> secure_stub.Capabilities(gnmi_pb2.CapabilityRequest())
supported_models {
<snip>
}
supported_encodings: <snip>
gNMI_version: "0.4.0"
```

gNMI プロトコルの設定例

例：非セキュア モードでの gNMI の有効化

（注） この例は Cisco IOS XE Amsterdam 17.3.1 以降のリリースに適用されます。

次に、gNxI サーバを非セキュアモードで有効にする例を示します。

```
Device> enable
Device# configure terminal
Device(config)# gnxi
Device(config)# gnxi server
Device(config)# gnxi port 50000 <The default port is 50052.>
Device(config)# end
Device#
```

例：セキュア モードでの gNMI の有効化

（注） この例は Cisco IOS XE Amsterdam 17.3.1 以降のリリースに適用されます。

次に、gNxI サーバをセキュアモードで有効にする例を示します。

```
Device> enable
Device# configure terminal
Device(config)# gnxi
Device(config)# gnxi secure-trustpoint trustpoint1
Device(config)# gnxi secure-server
Device(config)# gnxi secure-client-auth
Device(config)# gnxi secure-port 50001 <The default port is 9339.>
Device(config)# end
Device#
```

gNMI プロトコルの関連資料

関連資料

関連項目	マニュアル タイトル
DevNet	https://developer.cisco.com/site/ios-xe/
gNMI	https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md
gNMI パス エンコーディング	https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-path-conventions.md

標準および RFC

標準/RFC	タイトル
RFC 7951	YANG でモデル化されたデータの JSON エンコーディング

シスコのテクニカル サポート

説明	リンク
シスコのサポート Web サイトでは、シスコの製品やテクノロジーに関するトラブルシューティングにお役立ていただけるように、マニュアルやツールをはじめとする豊富なオンライン リソースを提供しています。 お使いの製品のセキュリティ情報や技術情報を入手するために、Cisco Notification Service（Field Notice からアクセス）、Cisco Technical Services Newsletter、Really Simple Syndication（RSS）フィードなどの各種サービスに加入できます。 シスコのサポート Web サイトのツールにアクセスする際は、Cisco.com のユーザ ID およびパスワードが必要です。	http://www.cisco.com/support

gNMI プロトコルの機能情報

次の表に、このモジュールで説明した機能に関するリリース情報を示します。この表は、ソフトウェア リリース トレインで各機能のサポートが導入されたときのソフトウェア リリースだけを示しています。その機能は、特に断りがない限り、それ以降の一連のソフトウェア リリースでもサポートされます。

プラットフォームのサポートおよびシスコ ソフトウェアイメージのサポートに関する情報を検索するには、Cisco Feature Navigator を使用します。Cisco Feature Navigator にアクセスするには、www.cisco.com/go/cfn に移動します。Cisco.com のアカウントは必要ありません。

表 *3 : gNMI* プロトコルの機能情報

機能名	リリース	機能情報
gNMI プロトコル	Cisco IOS XE Fuji 16.8.1a	この機能では、gNMI の機能と GET および SET RPC を使用したモデル駆動型の設定と運用データの取得について説明します。 この機能は、次のプラットフォームに実装されていました。 • Cisco Catalyst 9300 シリーズ スイッチ • Cisco Catalyst 9400 シリーズ スイッチ • Cisco Catalyst 9500 シリーズ スイッチ
	Cisco IOS XE Gibraltar 16.10.1	Cisco IOS XE Gibraltar 16.10.1 では、この機能は Cisco Catalyst 9500 ハイ パフォーマンス シリーズ スイッチに実装されていました。
	Cisco IOS XE Gibraltar 16.11.1	Cisco IOS XE Gibraltar 16.11.1 では、この機能は Cisco Catalyst 9600 シリーズ スイッチに実装されていました。
	Cisco IOS XE Gibraltar 16.12.1	この機能は、Cisco IOS XE Gibraltar 16.12.1 で次のプラットフォームに実装されました。 • Cisco Catalyst 9200 および 9200L シリーズ スイッチ • Cisco Catalyst 9300L SKU • Cisco cBR-8 コンバージドブロードバンド ルータ
	Cisco IOS XE Amsterdam 17.1.1	

機能名	リリース	機能情報
		この機能は、Cisco IOS XE Amsterdam 17.1.1で次のプラットフォームに実装されました。 • Cisco ASR 900 シリーズ アグリゲーション サービス ルータ • Cisco ASR 920 シリーズ アグリゲーション サービス ルータ • Cisco Network Convergence System 520 シリーズ • Cisco Network Convergence System 4200 シリーズ
	Cisco IOS XE Amsterdam 17.2.1r	この機能は、Cisco IOS XE Amsterdam 17.2.1r で、Cisco ASR 1000 シリーズ アグリゲーション サービス ルータに実装されました。
	Cisco IOS XE Cupertino 17.8.1	Cisco IOS XE Cupertino 17.8.1 では、この機能は次のプラットフォームに実装されました。 • Cisco Catalyst 9800-CL ワイヤレス コントローラ • Cisco Catalyst 9800-40 ワイヤレスコントローラ • Cisco Catalyst 9800-80 ワイヤレスコントローラ

機能名	リリース	機能情報
gNMI IPv6 のサポート	Cisco IOS XE Dublin 17.10.1	gNMI IPv6 のサポートは、Cisco IOS XE Dublin 17.10.1 で有効になっています。 • Cisco Catalyst 9200 および 9200L シリーズ スイッチ • Cisco Catalyst 9300、9300L、および 9300X シリーズ スイッチ • Cisco Catalyst 9400 シリーズ スイッチ • Cisco Catalyst 9500 および 9500 ハイパフォーマンス シリーズ スイッチ • Cisco Catalyst 9600 シリーズ スイッチ
gNMI ユーザ名とパスワードによる認証	Cisco IOS XE Gibraltar 16.12.1	ユーザ名とパスワードによる認証機能が gNMI プロトコルに追加されました。この機能は、gNMI をサポートするすべての IOS XE プラットフォームでサポートされます。
gNMI 設定の永続化	Cisco IOS XE Amsterdam 17.3.1	gNMI SetRequest RPCを介して行われたすべての正常な設定変更は、デバイスの再起動後も保持されます。この機能は、次のプラットフォームに実装されていました。 • Cisco Catalyst 9200 シリーズ スイッチ • Cisco Catalyst 9300 シリーズ スイッチ • Cisco Catalyst 9400 シリーズ スイッチ • Cisco Catalyst 9500 シリーズ スイッチ • Cisco Catalyst 9600 シリーズ スイッチ

機能名	リリース	機能情報
gNOI 証明書の管理	Cisco IOS XE Amsterdam 17.3.1	gNOI 証明書の管理サービスは、RPC を提供して、インストール、ローテーション、証明書の取得、証明書の失効、および証明書署名要求の生成を行います。この機能は、次のプラットフォームに実装されていました。 • Cisco Catalyst 9200 シリーズ スイッチ • Cisco Catalyst 9300 シリーズ スイッチ • Cisco Catalyst 9400 シリーズ スイッチ • Cisco Catalyst 9500 シリーズ スイッチ • Cisco Catalyst 9600 シリーズ スイッチ
PROTO エンコーディング	Cisco IOS XE Dublin 17.11.1	gNMI プロトコルは、PROTO エンコーディングをサポートします。*gnmi.proto* ファイルには、クライアント側とサーバー側で gNMI プロトコルのフレームワークを表す一連の手順を生成するためのブループリントが定義されています。 この機能は、次のプラットフォームに実装されていました。 • Cisco Catalyst 9200、9200L、および 9200X シリーズ スイッチ • Cisco Catalyst 9300、9300L、および 9300X シリーズ スイッチ • Cisco Catalyst 9400 および 9400X シリーズ スイッチ • Cisco Catalyst 9500、9500 ハイパフォーマンス、および 9500X シリーズ スイッチ • Cisco Catalyst 9600 シリーズ スイッチ

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。