REST API の使用

REST API について

REST API について

The Application Policy Infrastructure Controller (APIC) REST API is a programmatic interface that uses REST architecture. API は JavaScript オブジェクトの表記（JSON）または拡張マークアップ言語（XML）のドキュメントを含む HTTP（デフォルトでは無効）または HTTPS のメッセージを受け入れ、返します。You can use any programming language to generate the messages and the JSON or XML documents that contain the API methods or Managed Object (MO) descriptions.

The REST API is the interface into the management information tree (MIT) and allows manipulation of the object model state. The same REST interface is used by the APIC CLI, GUI, and SDK, so that whenever information is displayed, it is read through the REST API, and when configuration changes are made, they are written through the REST API. REST API は、使用する他の情報が取得できる、障害の統計情報を含む、および監査イベントのインターフェイスも提供します。Web ソケットでイベントを送信できます、MIT に変更が発生するようにもプッシュ ベースのイベント通知を購読するための手段を提供します。

API では、HTTP を通じた POST 操作、GET 操作、DELETE 操作など、標準的な REST メソッドがサポートされています。POST メソッドと DELETE メソッドは、同じ入力パラメータで複数回呼び出されても、それ以上の効果を持たないべき等です。GET メソッドはべきゼロで、何らの変更も行うことなく（つまり、読み取り専用操作）ゼロ回または複数回呼び出すことができます。

REST インターフェイスに出入りするペイロードは、XML エンコーディングまたは JSON エンコーディングによりカプセル化できます。XML の場合、エンコーディング操作は簡単です。

要素タグはパッケージとクラスの名前で、そのオブジェクトのプロパティはその要素の属性として指定します。含有は、子要素を作成して定義します。

JSON の場合、エンコーディングにはツリーベースの階層を反映する特定のエントリの定義が必要です。ただし、その定義はツリーのすべてのレベルで繰り返されるため、最初に理解していれば実装はかなり簡単です。

- すべてのオブジェクトは、キーが、パッケージとクラスの名前は、JSON 辞書としてについて説明します。値は 2 つのキーをもう 1 つのネストされた辞書: 属性と子。
- 属性キーには、オブジェクト上の属性を定義するキー値ペアを記述する、さらにネストされたディクショナリが含まれています。
- 子キーには、すべての子オブジェクトを定義するリストが含まれています。このリストの子オブジェクトは、ここで説明したように定義された、ネストされたオブジェクトを含むディクショナリです。

認証

REST API username- and password-based authentication uses a special subset of request Universal Resource Identifiers (URIs), including **aaaLogin**, **aaaLogout**, and **aaaRefresh** as the DN targets of a POST operation. Their payloads contain a simple XML or JSON payload containing the MO representation of an **aaaUser** object with the attribute name and **pwd** defining the username and password: for example, **<aaaUser name='admin' pwd='password'/>**. POST 操作の応答には、Set-Cookie ヘッダーと、応答の名前付きトークンの **aaaLogin** オブジェクトの属性の両方として認証トークンが含まれます。この場合の XPath は エンコーディングが XML の場合は **/imdata/aaaLogin/@token** です。Subsequent operations on the REST API can use this token value as a cookie named **APIC-cookie** to authenticate future requests.

サブスクリプション

The REST API supports the subscription to one or more MOs during your active API session. ユーザまたはシステムにより開始されたアクションによって、MO が作成、変更、または削除されると、イベントが生成されます。サブスクライブされたアクティブなクエリ上のデータがイベントにより変更されると、APIC はそのサブスクリプションを作成した API クライアントに通知を送信します。

管理情報モデル

アプリケーション セントリック インフラストラクチャ ファブリック を構成するすべての物理および論理コンポーネントは、MIT とも呼ばれる階層型管理情報モデル（MIM）で表されます。このツリー内の各ノードは、管理ステータスと動作ステータスを含む、MO とオブジェクトのグループを表します。

MIM を表示するには、「*Cisco APIC* 管理情報モデル リファレンス ガイド」を参照してください。

階層構造は最上部（Root）から始まり、親ノードと子ノードを含みます。Each node in this tree is an MO and each object in the ACI fabric has a unique distinguished name (DN) that describes the object

and its place in the tree. MO は、fabric リソースの抽象化です。MO は、スイッチやアダプタなどの物理オブジェクト、またはポリシーや障害などの論理オブジェクトを表すことができます。

Configuration policies make up the majority of the policies in the system and describe the configurations of different ACI fabric components. ポリシーは、ある環境下でシステムがどのように動作するかを決定します。特定の MO はユーザが作成せず、自動的に fabric によって作成されます（電源オブジェクトやファン オブジェクトなど）。API を起動することによって、MIM にオブジェクトの読み取りと書き込みを行います。

The information model is centrally stored as a logical model by the APIC, while each switch node contains a complete copy as a concrete model. When a user creates a policy in the APIC that represents a configuration, the APIC updates the logical model. The APIC then performs the intermediate step of creating a fully elaborated policy from the user policy and then pushes the policy into all the switch nodes where the concrete model is updated. モデルは、ファブリックで動作する複数のデータ管理エンジン（DME）のプロセスによって管理されます。ユーザまたはプロセスがファブリック コンポーネントへの管理上の変更を開始すると（たとえば、プロファイルをスイッチに適用する場合）、DME はその変更をまず情報モデルに適用し、次に実際の管理対象のエンドポイントに適用します。この方法を、モデル方式フレームワークといいます。

The following branch diagram of a リーフ スイッチ port starts at the top Root of the ACI fabric MIT and shows a hierarchy that comprises a chassis with two line module slots, with a line module in slot 2.

```
|──root──────────── (root)
    |──sys──────────── (sys)
        |──ch──────────(sys/ch)
            |──lcslot-1──────────(sys/ch/lcslot-1)
            |──lcslot-2──────────(sys/ch/lcslot-2)
                |──lc──────────(sys/ch/lcslot-2/lc)
                    |──leafport-1────────(sys/ch/lcslot-2/lc/leafport-1)
```

オブジェクトの命名

特定のオブジェクトは、識別名（DN）または相対名（RN）で識別できます。

（注） 既存のオブジェクトの名前は変更できません。オブジェクトまたはオブジェクトのグループへの参照を簡素化するために、エイリアスまたはタグを割り当てることができます。

識別名

DN を使用すると、特定のターゲット オブジェクトを明確に識別できます。DN は、一連の RN で構成されます。

```
dn = {rn}/{rn}/{rn}/{rn}...
```

この例では、DN によりオブジェクト ツリーの最上位からオブジェクトまで、`fabport-1` の完全修飾パスが提供されます。DN は、API コールが動作する管理対象オブジェクトを指定します。

```
< dn =”sys/ch/lcslot-1/lc/fabport-1” />
```

相対名

RNは、親オブジェクトのコンテキスト内の兄弟からのオブジェクトを識別します。DNには、一連の RN が含まれます。

例：

```
<dn = "sys/ch/lcslot-1/lc/fabport-1"/>
```

上記の DN には次の RN が含まれます。

相対名	クラス	説明
sys	top:System	このシステムの最上位レベル
ch	eqpt:Ch	ハードウェアのシャーシ コンテナ
lcslot-1	eqpt:LCSlot	ライン モジュール スロット 1
lc	eqpt:LC	ライン（I/O）モジュール
fabport-1	eqpt:FabP	ファブリック向きの外部 I/O ポート 1

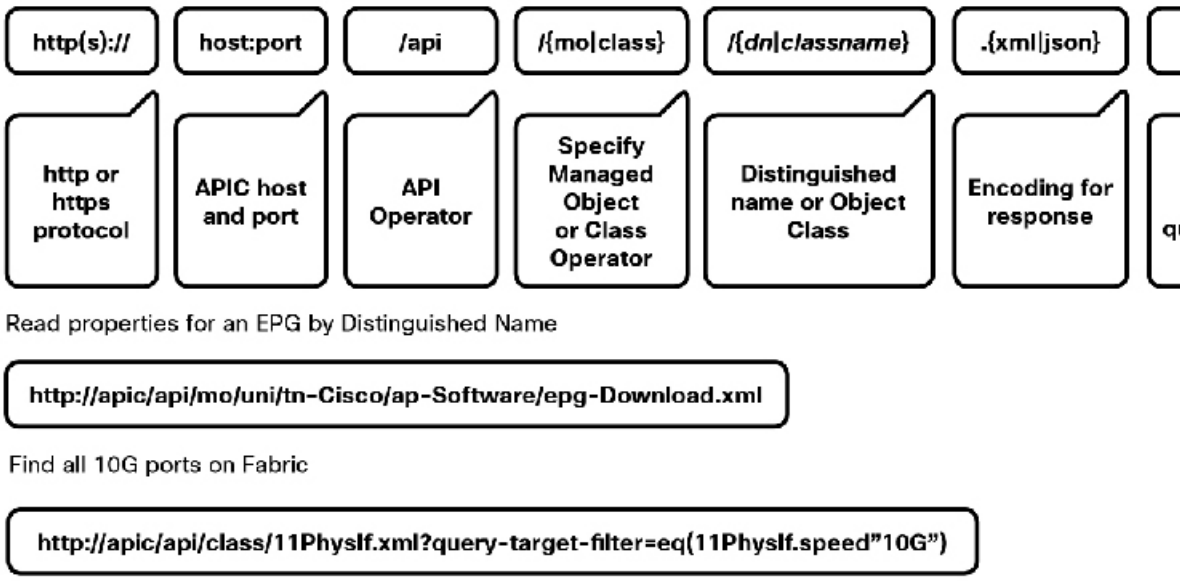
REST API 要求の構成

読み取り/書き込み操作とフィルタ

読み取り操作

オブジェクト ペイロードは、XML または JSON として適切に符号化された後は、REST API での作成、読み取り、更新、削除の操作に使用できます。次の図に、REST API からの読み取り操作の構文を示します。

図 *1 : REST* 構文



REST API は HTTP ベースなので、特定のリソース タイプにアクセスする URI を定義することが重要です。The first two sections of the request URI simply define the protocol and access details of the APIC. 要求 URI の次のセクションは、API の呼び出しを指示するリテラル文字列 **/api** です。先に説明したように、読み取り操作は一般に、オブジェクトまたはクラスに対するものであるため、URI の次の部分で操作が MO に対してか、クラスに対してかを指定します。次のコンポーネントで、オブジェクトベースのクエリに対して照会する完全修飾ドメイン名（DN）か、クラスベースのクエリに対するパッケージとクラス名のいずれかを定義します。要求 URI の最後の必須部分はエンコーディング形式で、.xml か .json のいずれかです。これは、ペイロード形式が定義されている唯一の方法です。(APIC ではコンテンツ タイプとその他のヘッダは無視されます。）

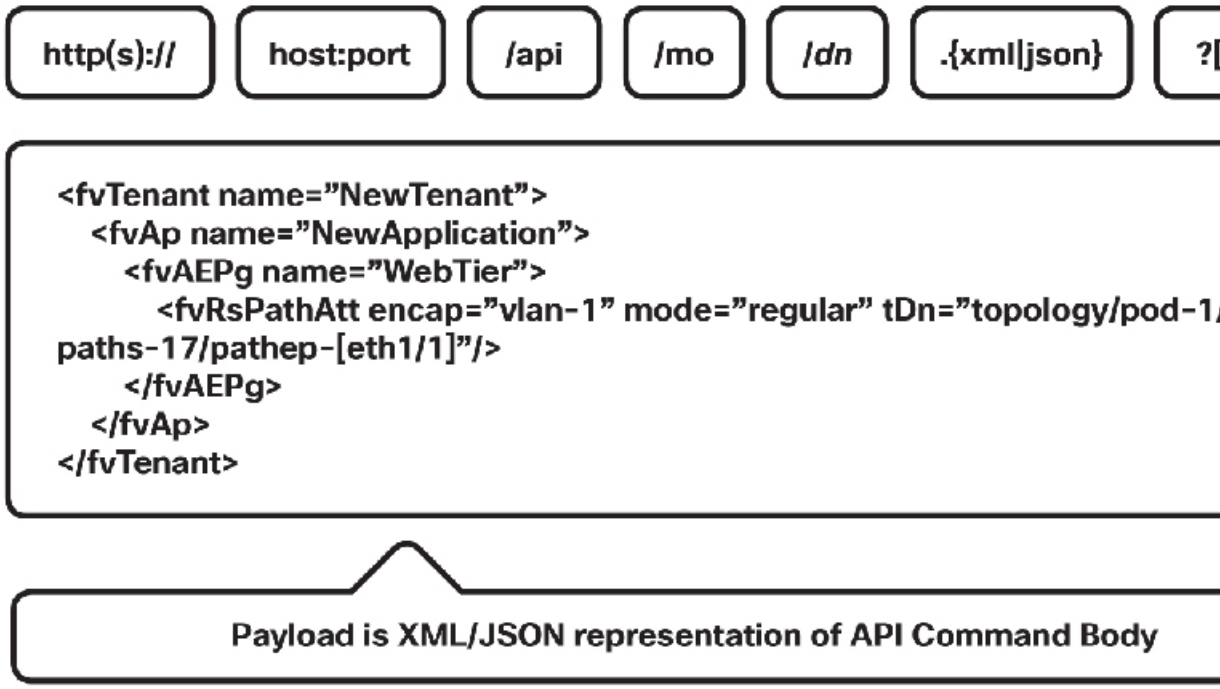
書き込み操作

REST API の作成操作と更新操作は両方とも POST メソッドを使用して実装されます。そのため、まだない場合はオブジェクトが作成され、既に存在する場合は、既存の状態と必要な状態との間の変化を反映するように更新されます。

作成操作と更新操作の両方に複合オブジェクト階層が含まれていることがあります。そのため、すべてのオブジェクトが同じコンテキスト ルートにあり、REST API のデータ ペイロードの 1 MB の制限を超えていない限り、1 つのコマンドで完全なツリーを定義することができます。この制限は、パフォーマンスを保証し、高負荷時にシステムを保護するために設定されています。

The context root helps define a method by which the APIC distributes information to multiple controllers and helps ensure consistency. ほとんどの場合、構成はユーザに透過的である必要があります。ただし、かなり大規模な構成で、それが分散トランザクションになる場合は、小さく分割する必要がある場合があります。

図 *2 : REST* ペイロード



作成操作と更新操作は、オブジェクト レベルで常に対象となること以外は、読み取り操作と同じ構文を使用します。これは、特定のクラスのあらゆるオブジェクトに変更を加えることができないからです（その必要もありません）。The create or update operation should target a specific managed object, so the literal string **/mo** indicates that the DN of the managed object will be provided, followed next by the actual DN. フィルタ文字列を POST 操作に適用できます。たとえば、POST 操作の結果を応答で取得する場合は、**rsp-subtree=modified** クエリ文字列を渡し、POST 操作によって変更されたオブジェクトを応答に含めるように指示します。

POST 操作のペイロードには、Cisco API コマンド本文を定義する MO を表す XML または JSON の符号化データが含まれます。

Filters

（注） イベント レコードの Cisco APIC REST API クエリについて、APIC システムでは最大 500,000 イベント レコードへの応答に制限しています。応答が 500,000 イベント以上の場合は、エラーが返されます。クエリを絞り込むためにフィルタを使用します。詳細については、クエリー フィルタ式の作成 （16 ページ）を参照してください。

REST API は、柔軟性のあるさまざまなフィルタをサポートしており、検索範囲を絞り込み、よりすばやく情報を見つけるのに役立ちます。フィルタ自体は、クエリの URI オプションとして付加され、疑問符（?）で始まり、アンパサンド（&）で連結されます。複数の条件をまとめて連結して複雑なフィルタを形成できます。

次のクエリ フィルタを使用できます。

表 *1*:クエリ フィルタ

Filter Type	Syntax	COBRAS クエリ プロパティ	説明
query-target	{self | children | subtree}	AbstractQuery.queryTarget	クエリの範囲を定義します
target-subtree-class	*class name*	AbstractQuery.classFilter	指定されたクラスを含む応答のみの要素
query-target-filter	フィルタ式	AbstractQuery.propFilter	条件に一致する応答のみの要素
rsp-subtree	{no | children | full}	AbstractQuery.subtree	応答に含まれる子オブジェクト レベルを指定します
rsp-subtree-class	*class name*	AbstractQuery.subtreeClassFilter	指定したクラスのみに応答します
rsp-subtree-filter	フィルタ式	AbstractQuery.subtreePropFilter	条件に一致するクラスのみに応答します
rsp-subtree-include	{faults | health :stats :…}	AbstractQuery.subtreeInclude	追加のオブジェクトを要求します
order-by	*classname.property* | {asc | desc}	未実装	プロパティ値に基づいて応答をソートします

REST API コマンドでクラスの使用

Application Policy Infrastructure Controller（APIC）のクラスは、システム イベントおよびフォールトがオブジェクト モデル内のオブジェクトとどのように関連しているかを理解するために操作面で重要です。システムの各イベントやフォールトは、設定、正常性、フォールト、および/または統計情報にアクセスできる固有のオブジェクトです。

Cisco アプリケーション セントリック インフラストラクチャ（ACI）ファブリックを作成するすべての物理および論理コンポーネントは、階層管理情報ツリー（MIT）で表されます。このツリー内の各ノードは、管理ステータスと動作ステータスを含む、管理対象オブジェクト（MO）またはオブジェクトのグループを表します。

クラスの完全なリストにアクセスするには、APIC を指定し、URL の最後で `doc/html` ディレクトリを参照します。

`https://apic-ip-address/doc/html/`

REST API コマンド内の管理オブジェクトの使用

管理対象オブジェクト（MO）またはそのプロパティで API 操作を実行する前に、Web ベースのマニュアルである『*Cisco APIC Management Information Model Reference*』でオブジェクトのクラス定義を確認する必要があります。管理情報モデル（MIM）は、次のようなルールを定義するスキーマとして機能します。

- MO を接続できる親オブジェクトのクラス
- MO に接続できる子オブジェクトのクラス
- MO に接続できるクラス タイプの子オブジェクトの数
- ユーザーが MO を作成、変更、または削除できるかどうか、またそのために必要な権限レベル
- オブジェクト クラスのプロパティ（属性）
- データ タイプとプロパティの範囲

When you send an API command, the APIC checks the command for conformance with the MIM schema. If an API command violates the MIM schema, the APIC rejects the command and returns an error message. たとえば、MO を作成できるのは、コマンド URI で指定したパスで作成が許可される場合、またはユーザがそのオブジェクト クラスに必要な権限レベルを持っている場合のみです。有効なデータのみで MO のプロパティを設定でき、プロパティは作成できません。

MO を作成する API コマンドを作成するときに必要なのは新しい MO を一意的に定義するためにコマンドの URI とデータ構造に十分な情報を盛り込むことだけです。MO の作成時にプロパティ設定を省略すると、MIM が指定している場合はプロパティがデフォルト値で入力され、それ以外は空白のままになります。

MO のプロパティを変更する場合に必要なのは、変更するプロパティとその新しい値を指定することだけです。他のプロパティは未変更のままになります。

注意事項および制約事項

- APIC またはスイッチ管理コミュニケーション ポリシーに影響する MO を変更すると、ファブリック内の APIC やスイッチ Web インターフェイスで進行中の操作に短時間の中断が発生する場合があります。中断が生じる可能性がある設定変更には次のものがあります。
  - ポート番号などの管理ポート設定の変更
  - HTTPS の有効化または無効化
  - HTTPS へのリダイレクションの状態の変更
  - キー リングなど公開キー インフラストラクチャ（PKI）の変更
- 既存の MO を読み取ると、MO の Password プロパティがセキュリティ上の理由から空白として読み取られます。APIC へ再度 MO を書き込むと、password プロパティは空白として作成されます。

ヒント　パスワード情報を含むMOを保存する必要がある場合は、構成のエクスポート ポリシーを使用します。特定の MO を保存するには、ポリシーでターゲットの 識別名として MO を指定します。

API コマンドの作成

次の形式の （URI）を使用して、HTTP または HTTPS メッセージを APIC に送信して API コマンドまたはクエリーを呼び出し、管理対象オブジェクト（MO）上で操作を実行できます。

{**http**|**https**}**://***host*[*:port*]**/api/mo/***dn***.**{**json**|**xml**}[**?**オプション]

オブジェクト クラス上での操作には次の形式を使用します。

{**http**|**https**}**://***host*[*:port*]**/api/class/***className***.**{**json**|**xml**}[**?**オプション]

次の例では、class fv:Tenant の MO に関与する API 操作用の URI の例を示します。

```
https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

URI のコンポーネント

URI のコンポーネントは次のとおりです。

- `http://` または `https://`：HTTP または HTTPS を指定します。デフォルトでは、HTTPS だけがイネーブルです。必要に応じて、「GUI を使用した HTTP および HTTPS の設定 （35 ページ）」で説明するように、HTTP または HTTP-to-HTTPS のリダイレクションを明示的にイネーブルにし設定する必要があります。HTTP と HTTPS は共存できます。
- *`host`* ：APIC のホスト名または IP アドレスを指定します。
- *`:port`* ：APIC との通信に使用するポート番号を指定します。システムが HTTP（80）または HTTPS（443）に標準のポート番号を使用する場合は、このコンポーネントを省略できます。
- `/api/`：メッセージが API に向けられることを指定します。
- `mo | class` ：操作のターゲットが MO またはオブジェクト クラスであるかどうかを指定します。
- *`dn`* ：対象の MO の識別名（DN）を指定します。
- *`className`* ：対象のクラスの名前を指定します。この名前は、照会されたオブジェクトのパッケージ名と、対応するパッケージのコンテキストで照会されたクラスの名前を連結したものです。

  たとえば、クラス aaa:User は、URI 内の aaaUser の *className* をもたらします。

- `json | xml` ：コマンドまたは応答 HTML 本文のエンコード形式が JSON または XML かを指定します。
- `?options` ：（任意）クエリーに1つ以上のフィルタ、セレクタ、または修飾子を指定します。複数のオプション ステートメントは、アンパサンド（&）で結合されます。

MO での API 操作用の URI

特定の MO を作成、読み取り、更新または削除する API 操作では、『*Cisco APIC Management Information Model Reference*』に記載するように、リソース パスは `/mo/` とその後の MO の DM で構成されます。たとえば、テナント オブジェクトの DN は、クラス fv:Tenant の参照定義で述べたように、`uni/tn-[name]` となります。この URI は、ExampleCorp という名前の fv:Tenant オブジェクトでの操作を指定します。

```
https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

または、POST 操作で、次の例のように、/api/mo に POST 送信してメッセージの本文に DN を提供することができます。

```
POST https://apic-ip-address/api/mo.xml

<fvTenant dn="uni/tn-ExampleCorp"/>
```

また、次の例のように、メッセージ本文に名前のみを提供して、/api/mo と残りの RN コンポーネントに POST 送信することもできます。

```
POST https://apic-ip-address/api/mo/uni.xml

<fvTenant name="ExampleCorp"/>
```

ノード MO での API 操作用の URI

ファブリックの特定のノード デバイス上の MO にアクセスするための API 操作では、リソースのパスは `/mo/topology/pod-number/node-number/sys/` とその後に続くノード コンポーネントで構成されます。たとえば、ポッド 1 のノード 1 のシャーシ スロット b 内のボード センサーにアクセスするには、次の URI を使用します。

```
GET https://apic-ip-address/api/mo/topology/pod-1/node-1/sys/ch/bslot/board/sensor-3.json
```

クラスでの API 操作用の URI

クラスに関する情報を入手するための API 操作では、『*Cisco APIC Management Information Model Reference*』に記載されているように、リソース パスは `/class/` とその後に続くクラスの名前で構成されます。URI では、クラス名のコロンは削除されます。たとえば、次の URI はクラス aaa:User でのクエリーを指定します。

```
GET https://apic-ip-address/api/class/aaaUser.json
```

API コマンド本文の作成

POST 操作の HTML 本文には、コマンドの実行に必要な必須情報を提供する JSON または XML のデータ構造を含める必要があります。GET または DELETE 操作ではデータ構造は送信されません。

API コマンド本文を作成するためのガイドライン

- データ構造は、ターゲット MO またはメソッドの属性と要素のセット全体を表す必要はありませんが、少なくとも URI に組み込まれたプロパティやパラメータを除く、MO を識別しコマンドを実行するのに必要な最低限のプロパティまたはパラメータのセットを提供する必要があります。
- データ構造は、すべての子ノードが一意の DN を持ち一意である、単一ツリーです。重複するノードは許可されません。同じノードを二度含めることで、1 つのノードを 2 度変更することはできません。この場合、1 つのノードに変更をマージする必要があります。
- データ構造では、パッケージ名の後のコロンはクラス名とメソッド名から除外されます。たとえば、クラス zzz:Object の MO のデータ構造では、クラス要素は zzzObject となります。
- Although the JSON specification allows unordered elements, the APIC REST API requires that the JSON 'attributes' element precede the 'children' array or other elements.
- XML データ構造に子またはサブツリーが含まれない場合、オブジェクト要素は自己終了できます。
- API の大文字小文字は区別されます。
- URL に「api」を含んで API コマンドを送信する場合、API POST コマンドの HTML 本文の最大サイズは 1 MB です。
- URL に「ppi」を含んでデバイス パッケージ ファイルをアップロードする場合、POST コマンドの HTML 本文の最大サイズは 10 MB です。

メソッドを呼び出す API コマンド本文の作成

メソッドを呼び出すコマンドを作成するには、『*Cisco APIC Management Information Model Reference*』のメソッド説明を使用して、メソッドのパラメータを含む JSON または XML データ構造を作成します。

一般的なメソッドの API リファレンスが、存在する入力パラメータと戻り値を一覧にします。メソッドは重要な入力パラメータが含まれている構造とともに呼び出され、正常な応答で戻り値を含む完全な構造を返します。

仮想的なメソッド設定の説明：メソッドは次のようにAPIリファレンスに表示される可能性があります。

```
Method config:Method(
   inParameter1,
   inParameter2,
   inParameter3,
   outParameter1,
   outParameter2
)
```

「in」で始まるパラメータは、入力パラメータをを表します。「out」で始まるパラメータは、メソッドにより返される値を表します。「in」または「out」プレフィックスを含まないパラメータは、入力パラメータです。

メソッドを呼び出す JSON 構造は、次のような構造で構成されます。

```
{
  "configMethod":
  {
    "attributes":
    {
      "inParameter1":"value1",
      "inParameter2":"value2",
      "inParameter3":"value3"
    }
  }
}
```

メソッドを呼び出す XML 構造は、次のような構造で構成されます。

```
<configMethod
    inParameter1="value1"
    inParameter2="value2"
    inParameter3="value3"
/>
```

（注） 一部のメソッドのパラメータには、フィルタ設定やMOの設定などサブ構造が含まれます。特定の情報については、「『*Cisco APIC Management Information Model Reference*』」を参照してください。

MO での API 操作用の API コマンドの本文の作成

MO を作成、変更または削除するためのコマンドを作成するには、『*Cisco APIC Management Information Model Reference*』にあるクラスの記述を使用して、オブジェクトのクラスの重要なプロパティと子を説明する JSON または XML のデータ構造を作成します。コマンドの実行に必要でない属性または子は省略できます。

仮定のクラス zzz:Object の MO 用の JSON 構造は、次の構造に似ています。

```
{
  "zzzObject" : {
    "attributes" : {
      "property1" : "value1",
      "property2" : "value2",
      "property3" : "value3"
    },
    "children" :
    [{
        "zzzChild1" : {
          "attributes" : {
            "childProperty1" : "childValue1",
            "childProperty2" : "childValue1"
          },
          "children" : []
        }
      }
    ]
  }
}
```

仮定のクラス zzz:Object の MO 用の XML 構造は、次の構造に似ています。

```
<zzzObject
  property1 = "value1",
  property2 = "value2",
  property3 = "value3">
    <zzzChild1
      childProperty1 = "childValue1",
      childProperty2 = "childValue1">
    </zzzChild1>
</zzzObject>
```

正常な操作では、MO の完全なデータ構造が返されます。

タグおよびエイリアスの使用

API 操作を簡略化するために、オブジェクトにエイリアスまたはタグを割り当てることができます。API 操作では、識別名（DN）の代わりにエイリアスまたはタグ名でオブジェクトまたはオブジェクトのグループを参照できます。次のようにタグとエイリアスの使用には違いがあります。

- タグ：タグを使用すると、わかりやすい名前で複数のオブジェクトをグループ化できます。複数のオブジェクトに同じタグ名を割り当て、1 つのオブジェクトに 1 つ以上のタグ名を割り当てることができます。
- エイリアス：単一のテナントを示す場合、エイリアスは DN よりも簡単でわかりやすいものにすることができます。1 つのオブジェクトだけに固有のエイリアス名を割り当てることができます。システムは 2 番目のオブジェクトに同じエイリアス名を割り当てできないようにします。

（注） すべてのオブジェクトがタグをサポートしているわけではありません。To determine whether an object is taggable, inspect the class of the object in the『*Cisco APIC Management Information Model Reference*』. オブジェクト クラスのその階層にタグのインスタンスがあれば（`tag:AInst` または `tag:AInst` から派生したクラス）、そのクラスのオブジェクトにタグ付けすることができます。

タグの追加

API POST 操作の URI で、次の構文を使用して 1 つ以上のタグを追加できます。

/api/tag/mo/*dn* **.** {**json**|**xml**}**?add=**[*, name, ...*][*, name, ...*]

この構文で、*name* はタグの名前、*dn* はタグが割り当てられるオブジェクトの識別名です。

次に、ExampleCorp という名前のテナントにタグのテナントと組織を割り当てる例を示します。

```
POST https://apic-ip-address/api/tag/mo/uni/tn-ExampleCorp.xml?add=tenants,orgs
```

タグの削除

API POST 操作の URI で、次の構文を使用して 1 つ以上のタグを削除できます。

/api/tag/mo/*dn* **.** {**json**|**xml**}**?remove=***name*[*, name, ...*]

次に、ExampleCorp という名前のテナントからタグの組織を削除する例を示します。

```
POST https://apic-ip-address/api/tag/mo/uni/tn-ExampleCorp.xml?remove=orgs
```

API DELETE 操作の URI で、次の構文を使用してタグのすべてのインスタンスを削除できます。

/api//tag/*name* **.** {**json**|**xml**}

次に、すべてのオブジェクトからタグの組織を削除する例を示します。

```
DELETE https://apic-ip-address/api/tag/orgs.xml
```

エイリアスの追加

API POST 操作の URI で、次の構文を使用してエイリアスを追加できます。

/api/alias/mo/*dn* **.** {**json**|**xml**}**?set=***name*

この構文で、*name* はエイリアスの名前、*dn* はエイリアスが割り当てられるオブジェクトの識別名です。

次に、ExampleCorp という名前のテナントにエイリアス tenant8 を割り当てる例を示します。

```
POST https://apic-ip-address/api/alias/mo/uni/tn-ExampleCorp.xml?set=tenant8
```

エイリアスの削除

API POST 操作の URI で、次の構文を使用してエイリアスを削除できます。

/api/alias/mo/*dn* **. {json|xml}?clear=yes**

次に、ExampleCorp という名前のテナントからエイリアスを削除する例を示します。

```
POST https://apic-ip-address/api/alias/mo/uni/tn-ExampleCorp.xml?clear=yes
```

その他の例

（注） ここに示す例では、タグに無関係な属性を削除するために応答が編集されています。

次に、ExampleCorp という名前のテナントに割り当てられたすべてのタグを検索する例を示します。

```
GET https://apic-ip-address/api/tag/mo/uni/tn-ExampleCorp.xml

RESPONSE:
<imdata>
    <tagInst
         dn="uni/tn-ExampleCorp/tag-tenants"
         name="tenants"
    />
    <tagInst
         dn="uni/tn-ExampleCorp/tag-orgs"
         name="orgs"
    />
</imdata>
```

次に、タグ「tenants」を持つすべてのオブジェクトを検索する例を示します。

```
GET https://apic-ip-address/api/tag/tenants.xml

RESPONSE:
<imdata>
    <fvTenant
        dn="uni/tn-ExampleCorp"
        name="ExampleCorp"
    />
</imdata>
```

REST API クエリの構成

クエリー フィルタ式の作成

論理演算子および値の式を適用して、API クエリーへの応答をフィルタリングできます。

基本的な等式または不等式のテストは次のように表されます。

```
query-target-filter=[eq|ne](attribute,value)
```

カッコやカンマを使用して演算子と条件を組み合わせることで、より複雑なテストを作成できます。

```
query-target-filter=[and|or]([eq|ne](attribute,value),[eq|ne](attribute,value),...)
```

（注） 範囲フィルタには最大で 20 の「`(attribute,value)`」フィルタ式を含めることができます。制限を超えると、API はエラーを返します。

使用可能な論理演算子

このテーブルは、クエリーのフィルタ式で使用可能な論理演算子を示します。

演算子	説明
`eq`	[Equal to]
`ne`	等しくない
`lt`	より小さい
`gt`	より大きい
`le`	以下
`ge`	以上
`bw`	間
`not`	論理反転
および	論理積
または	論理和
`xor`	論理排他的 OR

演算子	説明
`true`	ブール値 TRUE
`false`	ブール値 FALSE
`anybit`	少なくとも 1 つのビットが設定されている場合は TRUE
`allbits`	すべてのビットが設定されている場合は TRUE
`wcard`	Wildcard
`pholder`	プロパティ ホルダー
`passive`	パッシブ ホルダー

例

次の例では、姓が「Washington」に等しいクラス aaaUser のすべての管理対象オブジェクトが返されます。

```
GET https://apic-ip-address/api/class/aaaUser.json?
      query-target-filter=eq(aaaUser.lastName,"Washington")
```

次の例では、fabEncap プロパティが「vxlan-12780288」であるエンドポイント グループが返されます。

```
GET https://apic-ip-address/api/class/fvAEPg.xml?
    query-target-filter=eq(fvAEPg.fabEncap,"vxlan-12780288")
```

次の例は、現在のヘルス スコアが 50 未満のテナント オブジェクトすべてを示します。

```
GET https://apic-ip-address/api/class/fvTenant.json?
    rsp-subtree-include=health,required
        &
        rsp-subtree-filter=lt(healthInst.cur,"50")
```

次の例では、テナント ExampleCorp 下のすべてのエンドポイント グループとそれらの障害が返されます。

```
GET https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml?
    query-target=subtree
        &
        target-subtree-class=fvAEPg
            &
            rsp-subtree-include=faults
```

次の例では、名前が「infra」または「common」でない aaa:Domain オブジェクトが返されます。

```
GET https://apic-ip-address/api/class/aaaDomain.json?
    query-target-filter=
```

```
and(ne(aaaDomain.name,"infra"),
    ne(aaaDomain.name,"common"))
```

クエリー スコープ フィルタの適用

スコープ フィルタを適用して、API クエリーへの応答の範囲を制限できます。論理フィルタ式によって、クラス、プロパティ、カテゴリ、または認定に基づいてオブジェクトの第 1 レベルまたは 1 つ以上そのサブツリーまたは子への範囲を制限できます。

query-target={self | children | subtree}

このステートメントは、クエリーの範囲を制限します。次のリストは使用可能な範囲について説明します。

- **self**]：（デフォルト）MO 自体のみを考慮し、子またはサブツリーは考慮しません。
- **children**]：MO の子のみを考慮し、MO 自体は考慮しません。
- **subtree**]：MO 自体およびそのサブツリーを考慮します。

target-subtree-class=*mo-class1*[,*mo-class2*]...

This statement specifies which object classes are to be considered when the **query-target** option is used with a scope other than **self**. スペースなしのカンマ区切りリストとして複数の希望するオブジェクト タイプを指定できます。

To request subtree information, combine **query-target=subtree** with the **target-subtree-class** statement to indicate the specific subtree as follows:

```
query-target=subtree&target-subtree-class=className
```

この例では、実行中のファームウェアに関する情報を要求しています。The information is contained in the firmware:CtrlrRunning subtree (child) object of the APIC firmware status container firmware:CtrlrFwStatusCont:

```
GET https://apic-ip-address/api/class/firmwareCtrlrFwStatusCont.json?
    query-target=subtree&target-subtree-class=firmwareCtrlrRunning
```

query-target-filter=*filter-expression*

このステートメントは、応答に適用される論理フィルタを指定します。This statement can be used by itself or applied after the **query-target** statement.

rsp-subtree={no | children | full}

返されたオブジェクトについて、このオプションは子オブジェクトが応答に含まれるかどうかを指定します。次のリストは使用可能なオプションについて説明します。

- **no**]：（デフォルト）応答には子が含まれていません。
- **children**]：応答には子だけが含まれます。

• **full**]：応答には、子を含め構造全体が含まれます。

rsp-subtree-class=*mo-class*

子オブジェクトが返される場合、このステートメントは、指定されたオブジェクト クラスの子オブジェクトだけが応答に含まれることを指定します。

rsp-subtree-filter=*filter-expression*

子オブジェクトが返される場合、このステートメントは、子オブジェクトに適用される論理フィルタを指定します。

（注） When an **rsp-subtree-filter** query statement includes a *class.property* operand, the specified class name is used only to identify the property and its type. 返された結果はクラスでフィルタリングされず、同じ名前のプロパティを含む子オブジェクトを含む可能性がありますが、そのオブジェクトのプロパティがクエリー条件に一致する場合、別のクラスに属します。クラスでフィルタリングするには、追加のクエリー フィルタを使用する必要があります。

rsp-subtree-include=*category1*[,*category2*...][*option*]

子オブジェクトが返される場合、このステートメントは、応答に含まれる追加のオブジェクトまたはオプションを指定します。スペースなしのカンマ区切りリストで次のカテゴリの1つ以上を指定できます。

• **audit-logs**]：応答には、管理対象オブジェクトのユーザによる変更の履歴を含むサブツリーが含まれます。

• **event-logs**]：応答には、イベントの履歴情報を含むサブツリーが含まれます。

• **faults**]：応答には、現在アクティブな障害を含むサブツリーが含まれます。

• **fault-records**]：応答には、障害履歴情報を含むサブツリーが含まれます。

• **health**]：応答には、現在の動作状態情報を含むサブツリーが含まれます。

• **health-records**]：応答には、動作状態履歴情報を含むサブツリーが含まれます。

• **relations**]：応答には、関係関連のサブツリーの情報が含まれます。

• **stats**]：応答には、統計関連のサブツリーの情報が含まれます。

• **tasks**]：応答には、タスク関連のサブツリーの情報が含まれます。

上記いずれかのカテゴリとともに、次のオプションのいずれかを指定してさらにクエリー結果を絞り込むこともできます。

• **count**]：応答には、一致するサブツリーの数が含まれますが、サブツリー自体は含まれません。

- **no-scoped**]：応答には、要求したサブツリーの情報のみが含まれます。ターゲット MO の他の最上位の情報は応答に含まれていません。
- **required**]：応答には、指定したカテゴリと一致するサブツリーを持つ管理対象オブジェクトのみが含まれます。

For example, to include fault-related subtrees, specify **faults** in the list. To return only fault-related subtrees and no other top-level MO information, specify **faults,no-scoped** in the list as shown in this example:

```
query-target=subtree&rsp-subtree-include=faults,no-scoped
```

（注） 親オブジェクトがファブリック ノード（リーフ）にプッシュされるまで、一部のタイプの子オブジェクトは作成されません。Until such a parent object has been pushed to a fabric node, a query on the parent object using the **rsp-subtree-include** filter might return no results. たとえば、クエリー オプション `rsp-subtree-include=stats` を含む `fvAEPg` のクラス クエリーは、テナントに適用され、ファブリック ノードにプッシュされたエンドポイントのグループだけの統計を返します。

rsp-prop-include={all | naming-only | config-only}

This statement specifies what type of properties should be included in the response when the **rsp-subtree** option is used with an argument other than **no**.

- **all**]：応答には、管理対象オブジェクトのすべてのプロパティが含まれます。
- **naming-only**]：応答には、管理対象オブジェクトのプロパティの名前だけが含まれます。
- **config-only**]：応答には、管理対象オブジェクトの設定可能なプロパティのみが含まれます。

（注） 管理対象オブジェクトが設定できず、エクスポート（バックアップ）できないと、管理対象オブジェクトは返されません。

関連トピック

クエリー フィルタ式の作成 （16 ページ）
例：JSON API を使用した実行中のファームウェアの取得 （29 ページ）

API クエリー結果のフィルタリング

1 つ以上の条件文を次の形式でパラメータとしてクエリー URI に追加することで、API クエリーの結果をフィルタリングできます。

```
https://URI?condition[&condition[&...]]
```

複数の条件文は、アンパサンド（&）で結合されます。

（注） 条件文にはスペースを含めることはできません。

オブジェクト属性およびオブジェクトサブツリーによってフィルタするオプションが使用可能です。

フィルタの条件演算子

クエリーのフィルタリング機能は、次の条件演算子をサポートします。

演算子	説明
eq	[Equal to]
ne	等しくない
lt	より小さい
gt	より大きい
le	以下
ge	以上
bw	間
not	論理反転
および	論理積
または	論理和
xor	論理排他的 OR
true	ブール値 TRUE
false	ブール値 FALSE
anybit	少なくとも 1 つのビットが設定されている場合は TRUE
allbits	すべてのビットが設定されている場合は TRUE
wcard	Wildcard
pholder	プロパティ ホルダー
passive	パッシブ ホルダー

クエリー結果の並べ替えとページ付け

大量のデータを返すAPIクエリーを送信するときは、返されたデータを並べ替えてページ付けし、必要な情報が簡単に検索できるようにすることができます。

結果の並べ替え

`order-by` 演算子をクエリー URI に追加することで、クエリー応答をクラス内の 1 つ以上のプロパティによって並べ替えることができ、次の構文を使用して順序の方向を指定できます。

order-by=*classname*.*property*[| {**asc**|**desc**}][,*classname*.*property*[| {**asc**|**desc**}]][,...]

昇順（`asc`）または降順（`desc`）を指定するにはオプションのパイプ区切り文字（'|'）を使用します。順序が指定されていない場合、デフォルトは昇順となります。

複数のプロパティ（姓や名など）によってマルチレベルのソートを実行できますが、すべてのプロパティを同じMOにする必要があるか、または同じ抽象クラスから継承する必要があります。

次の例で、ユーザを姓で並べ替え、次に名で並べ替える方法を示します。

```
GET
https://apic-ip-address/api/class/aaaUser.json?order-by=aaaUser.lastName|asc,aaaUser.firstName|asc
```

結果のページ付け

`page-size` 演算子をクエリー URI に追加することで、次の構文を使用してクエリーの結果をオブジェクトのグループ（ページ）に分けることができます。オペランドは各グループ内のオブジェクトの数を指定します。

page-size=ページあたりのオブジェクト番号

`page` 演算子をクエリー URI に追加することで、次の構文を使用して単一のグループが返されるように指定できます。ページは、番号 0 から始まります。

page=ページ番号

次に、1ページあたり15個の障害インスタンスを降順で最初のページのみを返すように指定する方法を示します。

```
GET
https://apic-ip-address/api/class/faultInfo.json?order-by=faultInst.severity|desc&page=0&page-size=15
```

（注） ページ付けされているかどうかにかかわらず、すべてのクエリーが新しい結果のセットを生成します。単一ページだけを返すクエリーを実行すると、クエリー応答には集計結果の総数が含まれますが、送信されないページは保存されず、後続のクエリーによって取得できません。後続のクエリーは新しい結果のセットを生成し、そのクエリーで要求されたページを返します。

クエリー結果へのサブスクライブ

API クエリーを実行するときは、実行中の API セッション中に発生するそのクエリーの結果の今後の変更へのサブスクリプションを作成するオプションがあります。ユーザまたはシステムにより開始されたアクションによって、MO が作成、変更、または削除されると、イベントが生成されます。そのイベントがアクティブなサブスクライブ済みのクエリーの結果を変更すると、APIC はサブスクリプションを作成した API クライアントへのプッシュ通知を生成します。

WebSocket のオープン

API サブスクリプション機能は、WebSocket プロトコル（RFC 6455）を使用して API クライアントとの双方向接続を実行し、その接続によって API はクライアントに要求されていない通知メッセージを送信できます。この通知チャネルを確立するには、まず API との WebSocket 接続をオープンする必要があります。Only a single WebSocket connection is needed to support multiple query subscriptions with multiple APIC instances. WebSocket 接続は API セッション接続に依存し、API セッションが終了すると閉じます。

（注） 月のイベントがイベント manager を通過と (`eventmgr`)、クライアントの任意の売り上げ WebSocket サブスクリプションの通知を受信します。Although most of the APIC MOs do go through `eventmgr`, stats objects do not go through it, because updates are very frequent and not scalable. したがって、統計情報オブジェクトを購読すると no 通知されます。代わりに定期的にクエリを実行したり、MOs 統計情報をエクスポートできます。

WebSocket 接続は通常、次の例に示すように、HTML5 対応ブラウザで JavaScript 方式によって開きます。

```
var Socket = new WebSocket(https://apic-ip-address/socket%TOKEN%);
```

In the URI, the **%TOKEN%** is the current API session token (cookie). 次に、トークン付きの URI の例を示します。

```
https://apic-ip-address/socketGkZl5NLRZJl5+jqChouaZ9CYjgE58W/pMccR+LeXmdO0obG9NB
Iwo1VBo7+YC1oiJL9mS6I9qh62BkX+Xddhe0JYrTmSG4JcKZ4t3bcP2Mxy3VBmgoJjwZ76ZOuf9V9AD6X
l83lyoR4bLBzqbSSU1R2NIgUotCGWjZt5JX6CJF0=
```

WebSocket 接続の確立後は、API セッションの更新時に API セッション トークンを再送信する必要はありません。

サブスクリプションの作成

クエリーにサブスクリプションを作成するには、オプション `?subscription=yes` でクエリーを実行します。次の例では、JSON 形式で fv:Tenant クラスのクエリーにサブスクリプションを作成します。

```
GET https://apic-ip-address/api/class/fvTenant.json?subscription=yes
```

The query response contains a subscription identifier, **subscriptionId**, that you can use to refresh the subscription and to identify future notifications from this subscription.


```
{
  "subscriptionId" : "72057611234574337",
  "imdata" : [{
      "fvTenant" : {
        "attributes" : {
          "instanceId" : "0:0",
          "childAction" : "",
          "dn" : "uni/tn-common",
          "lcOwn" : "local",
          "monPolDn" : "",
          "name" : "common",
          "replTs" : "never",
          "status" : ""
        }
      }
    }
  ]
}
```


受信通知

サブスクリプションからのイベント通知では、サブスクリプション ID と MO の説明を含むデータ構造が提供されます。この JSON の例では、新しいユーザが名前「sysadmin5」で作成されています。


```
{
  "subscriptionId" : ["72057598349672454", "72057598349672456"],
  "imdata" : [{
      "aaaUser" : {
        "attributes" : {
          "accountStatus" : "active",
          "childAction" : "",
          "clearPwdHistory" : "no",
          "descr" : "",
          "dn" : "uni/userext/user-sysadmin5",
          "email" : "",
          "encPwd" : "TUxISkhH$VHyidGgBX0r7N/srt/YcMYTEn5248ommFhNFzZghvAU=",
          "expiration" : "never",
          "expires" : "no",
          "firstName" : "",
          "intId" : "none",
          "lastName" : "",
          "lcOwn" : "local",
          "name" : "sysadmin5",
          "phone" : "",
          "pwd" : "",
```

```
          "pwdLifeTime" : "no-password-expire",
          "pwdSet" : "yes",
          "replTs" : "2013-05-30T11:28:33.835",
          "rn" : "",
          "status" : "created"
        }
      }
    }
  ]
}
```

複数のアクティブなサブスクリプションが1つのクエリーに対し存在できるため、通知には例に示すように複数のサブスクリプション ID を含めることができます。

（注） 通知は、JSON または XML 形式でサポートされています。

サブスクリプションの更新

イベント通知を受信し続けるには、APIセッション中に各サブスクリプションを定期的に更新する必要があります。To refresh a subscription, send an HTTP GET message to the API method **subscriptionRefresh** with the parameter **id** equal to the **subscriptionId** as in this example:

```
GET https://apic-ip-address/api/subscriptionRefresh.json?id=72057611234574337
```

サブスクリプションが期限切れになっていなければ、APIはリフレッシュ メッセージに空の応答を返します。

（注） サブスクリプションのタイムアウト時間は1分です。通知が失われないようにするには、サブスクリプションのリフレッシュ メッセージを少なくとも60秒ごとに送信する必要があります。

REST API の例

API の例に関する情報

この例では、JSON および XML 構造が読みやすいように改行、スペース、インデントで展開されています。

例：JSON API を使用したリーフ ポート セレクタ プロファイルの追加

次に、リーフ ポート セレクタ プロファイルを追加する例を示します。

『*Cisco APIC Management Information Model Reference*』に示すように、このクラスの階層はリーフ ポート セレクタ プロファイルを形成します。

- fabric:LePortP：リーフ ポート プロファイルは、このクラスの管理対象オブジェクト（MO）で表され、uni/fabric/leportp-[*name*] の識別名（DN）形式（leportp-[*name*] は相対名（RN））を持ちます。リーフ ポート プロファイル オブジェクトは、子オブジェクトとしてリーフ ポート セレクタを含むことができるテンプレートです。
  - fabric:LFPortS：リーフ ポート セレクタは、このクラスの MO で表され、lefabports-[*name*]-typ-[*type*] の RN 形式を持ちます。リーフ ポート セレクタ オブジェクトには、子オブジェクトとして 1 つ以上のポートまたはポートの範囲を含めることができます。
    - fabric:PortBlk：リーフ ポートまたはリーフ ポートの範囲は、このクラスの MO で表され、portblk-[*name*] の RN 形式を持ちます。

新しいリーフ ポート セレクタ プロファイル MO を作成する API コマンドは、子 MO を作成および設定することもできます。

この例は、「MyLPSelectorProf」という名前のリーフ ポート セレクタ プロファイルを作成します。例のプロファイルには、リーフ スイッチ 1 のリーフ ポート 1 を選択し、リーフ スイッチ 1 のリーフ ポート 3 から 5 を選択する「MySelectorName」という名前のセレクタが含まれます。新しいプロファイルを作成および設定するには、この HTTP POST メッセージを送信します。

```
POST http://apic-ip-address/api/mo/uni/fabric/leportp-MyLPSelectorProf.json

{
  "fabricLePortP" : {
    "attributes" : {
      "descr" : "Selects leaf ports 1/1 and 1/3-5"
    },
    "children" : [{
        "fabricLFPortS" : {
          "attributes" : {
            "name" : "MySelectorName",
            "type" : "range"
          },
          "children" : [{
              "fabricPortBlk" : {
                "attributes" : {
                  "fromCard" : "1",
                  "toCard" : "1",
                  "fromPort" : "1",
                  "toPort" : "1",
                  "name" : "block2"
                }
              }
            }, {
              "fabricPortBlk" : {
                "attributes" : {
                  "fromCard" : "1",
                  "toCard" : "1",
                  "fromPort" : "3",
                  "toPort" : "5",
                  "name" : "block3"
                }
              }
            }
```

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

正常な動作時には次の応答本文が返されます。

```
{
  "imdata" : [{
      "fabricLePortP" : {
        "attributes" : {
          "instanceId" : "0:0",
          "childAction" : "deleteNonPresent",
          "descr" : "Select leaf ports 1/1 and 1/3-5",
          "dn" : "uni/fabric/leportp-MyLPSelectorProf",
          "lcOwn" : "local",
          "name" : "MyLPSelectorProf",
          "replTs" : "never",
          "rn" : "",
          "status" : "created"
        },
        "children" : [{
            "fabricLFPortS" : {
              "attributes" : {
                "instanceId" : "0:0",
                "childAction" : "deleteNonPresent",
                "dn" : "",
                "lcOwn" : "local",
                "name" : "MySelectorName",
                "replTs" : "never",
                "rn" : "lefabports-MySelectorName-typ-range",
                "status" : "created",
                "type" : "range"
              },
              "children" : [{
                  "fabricPortBlk" : {
                    "attributes" : {
                      "instanceId" : "0:0",
                      "childAction" : "deleteNonPresent",
                      "dn" : "",
                      "fromCard" : "1",
                      "fromPort" : "3",
                      "lcOwn" : "local",
                      "name" : "block3",
                      "replTs" : "never",
                      "rn" : "portblk-block3",
                      "status" : "created",
                      "toCard" : "1",
                      "toPort" : "5"
                    }
                  }
                }, {
                  "fabricPortBlk" : {
                    "attributes" : {
                      "instanceId" : "0:0",
                      "childAction" : "deleteNonPresent",
                      "dn" : "",
                      "fromCard" : "1",
                      "fromPort" : "1",
                      "lcOwn" : "local",
                      "name" : "block2",
```

```
                    "replTs" : "never",
                    "rn" : "portblk-block2",
                    "status" : "created",
                    "toCard" : "1",
                    "toPort" : "1"
                  }
                }
              }
            ]
          }
        }
      ]
    }
  }
]
}
```

新しいプロファイルを削除するには、次の例のように、`"status":"deleted"` の fabricLePortP 属性を持つ HTTP POST メッセージを送信します。

```
POST http://apic-ip-address/api/mo/uni/fabric/leportp-MyLPSelectorProf.json

{
  "fabricLePortP" : {
    "attributes" : {
      "status" : "deleted"
    }
  }
}
```

または、この HTTP DELETE メッセージを送信できます。

```
DELETE http://apic-ip-address/api/mo/uni/fabric/leportp-MyLPSelectorProf.json
```

例：JSON API を使用したノードに関する情報の取得

This example shows how to query the APIC to access a node in the system.

ファブリックの特定のノード デバイスに API 操作を指示するために、リソースのパスは `/mo/topology/pod-number/node-number/sys/` とその後に続くノード コンポーネントで構成されます。たとえば、この URI はノード 1 のシャーシ スロット B のボード センサー 3 にアクセスします。

```
GET http://apic-ip-address/api/mo/topology/pod-1/node-1/sys/ch/bslot/board/sensor-3.json
```

正常な動作時には次のような応答本文が返されます。

```
{
  "imdata" :
  [{
      "eqptSensor" : {
        "attributes" : {
          "instanceId" : "0:0",
```

```
          "childAction" : "",
          "dn" : "topology/pod-1/node-1/sys/ch/bslot/board/sensor-3",
          "id" : "3",
          "majorThresh" : "0",
          "mfgTm" : "not-applicable",
          "minorThresh" : "0",
          "model" : "",
          "monPolDn" : "",
          "rev" : "0",
          "ser" : "",
          "status" : "",
          "type" : "dimm",
          "vendor" : "Cisco Systems, Inc."
        }
      }
    }
  ]
}
```

例：JSON API を使用した実行中のファームウェアの取得

This example shows how to query the APIC to determine which firmware images are running.

The detailed information on running firmware is contained in an object of class firmware:CtrlrRunning, which is a child class (subtree) of the APIC firmware status container class firmware:CtrlrFwStatusCont. Because there can be multiple running firmware instances (one per APIC instance), you can query the container class and filter the response for the subtree of running firmware objects.

次に、API クエリー メッセージの例を示します。

```
GET http://apic-ip-address/api/class/firmware:CtrlrFwStatusCont.json?
        query-target=subtree
        &
        target-subtree-class=firmwareCtrlrRunning
```

正常な動作時には次のような応答本文が返されます。

```
{
  "imdata" : [{
      "firmwareCtrlrRunning" : {
        "attributes" : {
          "instanceId" : "0:0",
          "applId" : "3",
          "childAction" : "",
          "dn" : "Ctrlrfwstatuscont/ctrlrrunning-3",
          "lcOwn" : "local",
          "replTs" : "never",
          "rn" : "",
          "status" : "",
          "ts" : "2012-12-31T16:00:00.000",
          "type" : "ifc",
          "version" : "1.1"
        }
      }
    }, {
      "firmwareCtrlrRunning" : {
        "attributes" : {
          "instanceId" : "0:0",
```

```
            "applId" : "1",
            "childAction" : "",
            "dn" : "ctrlrfwstatuscont/ctrlrrunning-1",
            "lcOwn" : "local",
            "replTs" : "never",
            "rn" : "",
            "status" : "",
            "ts" : "2012-12-31T16:00:00.000",
            "type" : "ifc",
            "version" : "1.1"
          }
        }
      }, {
        "firmwareCtrlrRunning" : {
          "attributes" : {
            "instanceId" : "0:0",
            "applId" : "2",
            "childAction" : "",
            "dn" : "ctrlrfwstatuscont/ctrlrrunning-2",
            "lcOwn" : "local",
            "replTs" : "never",
            "rn" : "",
            "status" : "",
            "ts" : "2012-12-31T16:00:00.000",
            "type" : "ifc",
            "version" : "1.1"
          }
        }
      }
    ]
}
```

This response describes three running instances of APIC firmware version 1.1.

例：JSON API を使用したトップ レベルのシステム要素の取得

This example shows how to query the APIC to determine what system devices are present.

General information about the system elements (APICs, spines, and leafs) is contained in an object of class top:System.

次に、API クエリー メッセージの例を示します。

```
GET http://apic-ip-address/api/class/topSystem.json
```

正常な動作時には次のような応答本文が返されます。

```
{
  "imdata" :
  [{
      "topSystem" : {
        "attributes" : {
          "instanceId" : "0:0",
          "address" : "10.0.0.32",
          "childAction" : "",
          "currentTime" : "2013-06-14T04:13:05.584",
          "currentTimeZone" : "",
          "dn" : "topology/pod-1/node-17/sys",
          "fabricId" : "0",
```

```
            "id" : "17",
            "inbMgmtAddr" : "0.0.0.0",
            "lcOwn" : "local",
            "mode" : "unspecified",
            "name" : "leaf0",
            "nodeId" : "0",
            "oobMgmtAddr" : "0.0.0.0",
            "podId" : "1",
            "replTs" : "never",
            "role" : "leaf",
            "serial" : "FOX-270308",
            "status" : "",
            "systemUpTime" : "00:00:02:03"
          }
        }
      }, {
        "topSystem" : {
          "attributes" : {
            "instanceId" : "0:0",
            "address" : "10.0.0.1",
            "childAction" : "",
            "currentTime" : "2013-06-14T04:13:29.301",
            "currentTimeZone" : "PDT",
            "dn" : "topology/pod-1/node-1/sys",
            "fabricId" : "0",
            "id" : "1",
            "inbMgmtAddr" : "0.0.0.0",
            "lcOwn" : "local",
            "mode" : "unspecified",
            "name" : "apic0",
            "nodeId" : "0",
            "oobMgmtAddr" : "0.0.0.0",
            "podId" : "0",
            "replTs" : "never",
            "role" : "apic",
            "serial" : "",
            "status" : "",
            "systemUpTime" : "00:00:02:26"
          }
        }
      }
    ]
}
```


This response indicates that the system consists of one APIC (node-1) and one leaf (node-17).

例：XML API および OwnerTag を使用した監査ログ情報のアクションへの追加

This example shows how to use the **ownerTag** or **ownerKey** property to add custom audit log information when an object is created or modified.

All configurable objects contain the properties **ownerTag** and **ownerKey**, which are user-configurable string properties. When any configurable object is created or modified by a user action, an audit log record object (**aaa:ModLR**) is automatically created to contain information about the change to be reported in the audit log. The audit log record object includes a list (the **changeSet** property) of the configured object's properties that were changed by the action. In the command to create or modify the configurable object, you can add your own specific tracking information, such as a job ticket number or the name of the person

making the change, to the **ownerTag** or **ownerKey** property of the configurable object. この追跡情報は変更の詳細とともに監査ログ レコード内に取り込まれます。

（注） The **ownerTag** information will appear in the log only when the **ownerTag** contents have been changed. To include the same information in a subsequent configuration change, you can clear the **ownerTag** contents before making the next configuration change. This condition applies also to the **ownerKey** property.

次の例では、ドメイン参照がテナント設定に追加されます。As part of the command, the operator's name is entered as the **ownerKey** and a job number is entered as the **ownerTag**.

```
POST https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml

<fvTenant name="ExampleCorp" ownerKey="georgewa" ownerTag="chg:00033">
    <aaaDomainRef name="ExampleDomain" ownerKey="georgewa" ownerTag="chg:00033"/>
</fvTenant>
```

In this case, two **aaa:ModLR** records are generated — one for the **fv:Tenant** object and one for the **aaa:DomainRef** object. Unless the **ownerKey** or **ownerTag** properties are unchanged from a previous configuration, their new values will appear in the **changeSet** list of the **aaa:ModLR** records, and this information will appear in the audit log record that reports this configuration change.

例：IP および MAC アドレスによる XML 取得エンドポイント（デバイス）

fvCEp クラスは、ファブリックにアタッチされたエンドポイント（デバイス）および関連付けられている IP および MAC アドレス、および各オブジェクトのカプセル化のリストを取得するために使用できます。

手順

それぞれの IP アドレスと MAC アドレスを持つエンドポイントのリストを取得するには、次の例のように XML クエリを使用します。

例：

```
GET https://apic-ip-address/api/node/class/fvCEp.xml
```

例: Monitoring REST API を使用して

In the examples in this topic, the JSON and XML structures have been expanded with line feeds, spaces, and indentations for readability.

XML の例: ファブリックでの障害の現在のリストを取得します。

You can use the **faultInst** class to derive all faults associated with the fabric, tenant, or individual managed objects within the APIC. この例などと XML のクエリを送信します。

```
 GET https://apic-ip-address/api/node/class/faultInst.xml?
query-target-filter=and(eq(faultInst.cause,"config-failure"))
```

XML の例: が失敗した設定によって引き起こされたファブリックでの障害の現在のリストを取得します。

使用することも、 **インストールの障害** と次の例などの XML の障害が発生した設定によって引き起こされた障害への応答を制限するフィルタとクラス。

```
 GET https://apic-ip-address/api/node/class/faultInst.xml?
query-target-filter=and(e(stultification,"config-failure"))
```

XML の例: 特定の管理対象オブジェクト DN のプロパティを取得します。

次の例などの XML で、テナント名のプロパティを取得するのに月クエリを使用できます。

```
 GET https://apic-ip-address/api/node/mo/uni/tn-common.xml?query-target=self
```

REST API へのアクセス

REST API へのアクセス

手順

By using a script or a browser-based REST client, you can send an API POST or GET message of the form: **https://***apic-ip-address*/**api/***api-message-url*

初期設定時に設定したアウトオブバンド管理 IP アドレスを使用します。

（注）
- https だけがデフォルトでイネーブルになっています。デフォルトでは、http および http から https へのリダイレクションがディセーブルになっています。
- API セッションを開始するために認証メッセージを送信する必要があります。初期設定時に設定した管理者ログイン名とパスワードを使用します。

API の呼び出し

HTTP/1.1 か HTTPS POST、GET、または DELETE メッセージを Application Policy Infrastructure Controller（APIC）に送信することで、API 機能を呼び出すことができます。POST メッセージ

の HTML 本文には、MO または API メソッドを記述する Javascript Object Notation（JSON）または XML データ構造が含まれます。応答メッセージの HTML 本文は要求されたアクションが要求されたデータ、確認、エラー情報が含まれる JSON または XML 構造が含まれます。

（注） 応答構造のルート要素は imdata です。この要素は応答用のコンテナにすぎず、管理情報モデル（MIM）のクラスではありません。

HTTP 要求メソッドとコンテンツ タイプの設定

API コマンドとクエリーは、次の項で説明されるように、サポートされている HTTP または HTTPS の要求メソッドとヘッダー フィールドを使用する必要があります。

（注） **セキュリティを確保するために、API 通信のデフォルト モードとして HTTPS のみがイネーブルになります。**HTTP、および HTTP から HTTPS へのリダイレクションは必要に応じてイネーブルにすることができますが、安全性は低くなります。わかりやすくするために、このマニュアルではプロトコル コンポーネントおよびインタラクションの説明において HTTP に言及しています。

Request Methods

API は次のように HTTP POST、GET および DELETE の要求メソッドをサポートしています。

- MO を作成または更新する API コマンド、またはメソッドを実行する API コマンドは、HTTP POST メッセージとして送信されます。
- MO のプロパティおよびステータスを読み取る API クエリー、またはオブジェクトを検出する API クエリーは、HTTP GET メッセージとして送信されます。
- MO を削除する API コマンドは、HTTP POST または DELETE メッセージとして送信されます。ほとんどの場合、POST 操作でそのステータスを deleted に設定することで MO を削除できます。

PUT などの他の HTTP メソッドはサポートされません。

（注） DELETE メソッドはサポートされていますが、HTTP ヘッダには次のコマンドのみ表示される可能性があります。 Access-Control-Allow-Methods: POST, GET, OPTIONS

コンテンツ タイプ

API は、API の要求または応答の HTML 本文で JSON または XML のデータ構造をサポートしています。使用する形式を示す .json または .xml を URI パス名の末尾に付け加えることでコン

テンツ タイプを指定する必要があります。The HTTP **Content-Type** and **Accept** headers are ignored by the API.

GUI を使用した HTTP および HTTPS の設定

この手順では、GUI および REST API へアクセスするためのサポート対象の通信プロトコルを設定します。

デフォルトでは、HTTPS だけがイネーブルです。必要に応じて、HTTP または HTTP-to-HTTPS を明示的にイネーブルにし設定する必要があります。HTTP と HTTPS は共存できます。

手順

ステップ 1 メニュー バーで、[FABRIC] > [Fabric Policies] をクリックします。

ステップ 2 [Navigation] ペインで、[Pod Policies] > [Policies] > [Communication] を展開します。

ステップ 3 [Communication] で、デフォルト ポリシーをクリックします。

ステップ 4 [Work] ペインの [HTTP] または [HTTPS] 領域で、[Admin State] ドロップダウン リストから目的の状態を選択してプロトコルをイネーブルまたはディセーブルにします。

ステップ 5 [HTTP] 領域で、[Redirect] ドロップダウン リストから目的の状態を選択して HTTP-to-HTTPS リダイレクションをイネーブルまたはディセーブルにします。

ステップ 6 [送信（Submit）] をクリックします。

GUI を使用した Cisco ACI HTTPS アクセス用カスタム証明書の設定

注意：ダウンタイムの可能性があるため、メンテナンス時間中にのみこのタスクを実行してください。ダウンタイムは外部ユーザまたはシステムからの APIC クラスタおよびスイッチへのアクセスには影響しますが、APIC とスイッチの接続には影響しません。スイッチ上の NGINX プロセスも影響を受けますが、外部接続のみでファブリックのデータ プレーンには影響ありません。APIC、設定、管理、トラブルシューティングなどへのアクセスは影響を受けることになります。この操作中にファブリック内のすべての Web サーバの再起動が予期されます。

始める前に

適切な認証局を作成できるように、信頼できる証明書を取得する機関を決定します。

手順

ステップ 1 メニュー バーで、**[Admin]** > **[AAA]** の順に選択します。

ステップ 2 [Navigation] ペインで、**[Public Key Management]** > **[Certificate Authorities]** の順に選択します。

ステップ 3 [Work] ペインで、**[Actions]** > **[Create Certificate Authority]** の順に選択します。

ステップ 4 [Create Certificate Authority] ダイアログボックスの [Name] フィールドに、認証局の名前を入力します。

ステップ 5 [Certificate Chain] フィールドに、Application Policy Infrastructure Controller（APIC）の証明書署名要求（CSR）に署名する認証局の中間証明書およびルート証明書をコピーします。

証明書は、Base64 エンコード X.509（CER）形式である必要があります。中間証明書はルート CA 証明書の前に配置されます。次の例のようになります。

```
-----BEGIN CERTIFICATE-----
<Intermediate Certificate>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Root CA Certificate>
-----END CERTIFICATE-----
```

ステップ 6 [Submit] をクリックします。

ステップ 7 [Navigation] ペインで、**[Public Key Management]** > **[Key Rings]** の順に選択します。

ステップ 8 [Work] ペインで、**[Actions]** > **[Create Key Ring]** の順に選択します。

ステップ 9 [Create Key Ring] ダイアログボックスで、[Name] フィールドに、名前を入力します。

ステップ 10 [Certificate] フィールドには、コンテンツを追加しないでください。

ステップ 11 [Modulus] フィールドで、目的のキー強度のラジオボタンをクリックします。

ステップ 12 [Certificate Authority] フィールドのドロップダウン リストから、前に作成した認証局を選択します。[Submit] をクリックします。

（注） キー リングは削除しないでください。キー リングを削除すると、CSR で使用されている関連秘密キーが自動的に削除されます。

[Work] ペインの [Key Rings] 領域では、作成したキー リングに対する [Admin State] に [Started] と表示されます。

ステップ 13 [Navigation] ペインで、**[Public Key Management]** > **[Key Rings]** > ***[key_ring_name]*** の順に選択します。

ステップ 14 [Work] ペインで、**[Actions]** > **[Create Certificate Request]** の順に選択します。

ステップ 15 [Subject] フィールドに、APIC の完全修飾ドメイン名（FQDN）を入力します。

ステップ 16 必要に応じて、残りのフィールドに入力します。

（注） 使用可能なパラメータの説明については、[Create Certificate Request] ダイアログボックスでオンライン ヘルプ情報を確認してください。

ステップ 17 [Submit] をクリックします。
[Navigation] ペインでは、前に作成したキー リングの下にオブジェクトが作成され、表示されます。[Navigation] ペインでそのオブジェクトをクリックすると、[Work] ペインの [Properties] 領域の [Request] フィールドにその CSR が表示されます。認証局に送信するコンテンツをフィールドからコピーします。

ステップ 18 [Navigation] ペインで、**[Public Key Management]** > **[Key Rings]** > ***[key_ring_name]*** の順に選択します。

ステップ 19 [Work] ペインの [Certificate] フィールドに、認証局から受信した署名付き証明書を貼り付けます。

ステップ 20 [Submit] をクリックします。

（注） CSR がキー リングで示されている認証局によって署名されていない場合、または証明書に MS-DOS 形式の行末が含まれている場合は、エラー メッセージが表示され、証明書は承認されません。MS-DOS 形式の行末を削除します。

キーが確認されて [Work] ペインの [Admin State] が [Completed] に変わり、HTTP ポリシーを使用できるようになります。

ステップ 21 メニュー バーで、**[Fabric]** > **[Fabric Policies]** の順に選択します。

ステップ 22 [Navigation] ペインで、**[Pod Policies]** > **[Policies]** > **[Management Access]** > **[default]** の順に選択します。

ステップ 23 [Work] ペインの [Admin Key Ring] ドロップダウン リストで目的のキー リングを選択します。

ステップ 24 [Submit] をクリックします。
すべての Web サーバが再起動されます。証明書がアクティブになり、デフォルト以外のキーリングが HTTPS アクセスに関連付けられています。

次のタスク

証明書の失効日には注意しておき、期限切れになる前に対応する必要があります。更新された証明書に対して同じキーペアを維持するには、CSR を維持する必要があります。これは、CSR にはキーリング内の秘密キーとペアになる公開キーが含まれているためです。証明書が期限切れになる前に、同じ CSR を再送信する必要があります。キーリングを削除すると、APIC に内部的に保存されている秘密キーも削除されるため、新しいキーリングの削除または作成は行わないでください。

API セッションの認証と維持

APIにアクセスする前に、設定したユーザの名前とパスワードで最初にログインする必要があります。

ログイン メッセージが受け入れられると、API は秒単位のセッション タイムアウト時間を含むデータ構造と、セッションを表すトークンを返します。トークンは、HTTP 応答ヘッダーに Cookie としても返されます。セッションを維持するために、セッション タイムアウト時間より長い期間他のメッセージが送信されなかった場合は API にログイン リフレッシュ メッセージを送信する必要があります。トークンはセッションが更新されるたびに変わります。

（注） デフォルトのセッション タイムアウト時間は 300 秒、つまり 5 分です。

これらの API メソッドにより、セッション認証を管理することができます。

- **aaaLogin**；POST メッセージとして送信し、このメソッドによりユーザーにログインし、セッションを開きます。メッセージ本文には、名前とパスワードの属性を含む aaa:User オブジェクトが含まれ、応答にはセッション トークンと Cookie が含まれます。If multiple AAA login domains are configured, you must prepend the user's name with **apic:***domain*\\.
- **aaaRefresh**—Sent as a GET message with no message body or as a POST message with the **aaaLogin** message body, this method resets the session timer. レスポンスには、新しいセッション トークンと Cookie が含まれます。
- **aaaLogout**：POST メッセージとして送信し、このメソッドはユーザをログアウトし、セッションを閉じます。メッセージ本文には、name 属性を持つ aaa:User オブジェクトが含まれます。応答には、空のデータ構造が含まれます。
- **aaaListDomains**：GET メッセージとして送信し、このメソッドは有効な AAA ログイン ドメインのリストを返します。ログインせずにこのメッセージを送信できます。

JSON または XML データ構造を指定して、この構文を使用して認証方式を呼び出すことができます。

{**http**|**https**}**://***host*[*:port*]**/api/***methodName***.**{**json**|**xml**}

次に、JSON データ構造を使用するユーザ ログイン メッセージの例を示します。

```
POST https://apic-ip-address/api/aaaLogin.json

{
  "aaaUser" : {
    "attributes" : {
      "name" : "georgewa",
      "pwd" : "paSSword1"
    }
  }
}
```

次の例に、トークンと更新のタイムアウト時間を含む、ログイン成功時の応答の一部を示します。

```
RESPONSE:
{
  "imdata" : [{
      "aaaLogin" : {
        "attributes" : {
          "token" :
             "GkZl5NLRZJl5+jqChouaZ9CYjgE58W/pMccR+LeXmdO0obG9NB
             Iwo1VBo7+YC1oiJL9mS6I9qh62BkX+Xddhe0JYrTmSG4JcKZ4t3
             bcP2Mxy3VBmgoJjwZ76ZOuf9V9AD6Xl83lyoR4bLBzqbSSU1R2N
             IgUotCGWjZt5JX6CJF0=",
          "refreshTimeoutSeconds" : "300",
          "lastName" : "Washington",
          "firstName" : "George"
        },
        "children" : [{
    ...
[TRUNCATED]
    ...
```

```
}
```

In the preceding example, the **refreshTimeoutSeconds** attribute indicates that the session timeout period is 300 seconds.

次に、有効なログイン ドメインのリストを要求する例を示します。

```
GET https://apic-ip-address/api/aaaListDomains.json

RESPONSE:
{
  "imdata": [{
    "name": "ExampleRadius"
  },
  {
    "name": "local",
    "guiBanner": "San Jose Fabric"
  }]
}
```

前の例では、応答データは2つの予想されるログイン ドメイン「ExampleRadius」および「local」を示します。次に、ExampleRadius ログイン ドメインに対するユーザ ログイン メッセージの例を示します。

```
POST https://apic-ip-address/api/aaaLogin.json

{
  "aaaUser" : {
    "attributes" : {
      "name" : "apic:ExampleRadius\\georgewa",
      "pwd" : "paSSword1"
    }
  }
}
```

API セッションに必要なチャレンジ トークン

APIセッションのセキュリティをより強力にするために、セッションがチャレンジ トークンを使用するように要求することができます。ログイン時にこの機能を要求すると、API によりトークン文字列が返されるのでそれをAPIへのすべての後続メッセージに含める必要があります。正常なセッション トークンとは異なり、チャレンジ トークンは、ブラウザによって自動的に提供されるCookieとして保存されません。API コマンドとクエリーは、次の方法のいずれかを使用してチャレンジ トークンを提供する必要があります。

- チャレンジ トークンは、API メッセージのURI内の「challenge」パラメータとして送信されます。
- チャレンジ トークンは、「APIC-challenge」を使用したHTTPまたはHTTPSヘッダーの一部です。

チャレンジ トークンを要求するセッションを開始するには、次の例に示すように、ログイン メッセージに URI パラメータ ステートメント `?gui-token-request=yes` を加えます。

```
POST https://192.0.20.123/api/aaaLogin.json?gui-token-request=yes
```

応答メッセージの本文には、形式の属性 "urlToken":"*token*" が含まれ、*token* はチャレンジ トークンを表す文字の長い文字列です。このセッション中の API へのすべての後続メッセージにはチャレンジ トークンを含める必要があります。次に示す例では「challenge」URI パラメータとして送信されています。

```
GET https://192.0.20.123/api/class/aaaUser.json?challenge=fa47e44df54562c24fef6601dc...
```

次の例では、チャレンジ トークンが HTTP ヘッダーの「APIC-challenge」フィールドとしてどのように送信されるかを示します。

```
GET //api/class/aaaUser.json
 HTTP/1.1
 Host: 192.0.20.123
 Connection: keep-alive
 Accept: text/html,application/xhtml+xml,application/xml,application/json
 APIC-challenge: fa47e44df54562c24fef6601dcff72259299a077336aecfc5b012b036797ab0f
.
.
.
```

ログイン

You can log in to the APIC REST API by sending a valid username and password in a data structure to the **aaaLogin** API method, as described in API セッションの認証と維持 （37 ページ）. ログイン成功後に、定期的にセッションを更新する必要があります。

次の例に、XML および JSON を使用して管理者としてログインし、設定時にセッションを更新し、ログアウトする方法を示します。

（注） At this time, the **aaaLogout** method returns a response but does not end a session. Your session ends after a refresh timeout when you stop sending **aaaRefresh** messages.

自分のユーザ クレデンシャルの変更

APIC にログインすると、パスワード、SSH キー、X.509 証明書などの、自分のユーザ クレデンシャルを変更できます。次の API メソッドが、ログインしたユーザのユーザ クレデンシャルの変更用に提供されています。

- `changeSelfPassword`
- `changeSelfSshKey`
- `changeSelfX509Cert`

（注） これらのメソッドを使用して、自分がログインするアカウントのクレデンシャルだけを変更できます。

各メソッドのメッセージ本文には変更されるオブジェクトのプロパティが含まれています。The properties are shown in the 『*Cisco APIC Management Information Model Reference*』.

パスワードの変更

パスワードを変更するには、`aaa:changePassword` オブジェクトを変更する `changeSelfPassword` API メソッドを送信します。次のオブジェクト プロパティがメッセージ本文に必要です。

- `userName`：ログイン ID。
- `oldPassword`：現在のパスワード。
- `newPassword`：新しいパスワード。

この例では、User1 によって送信されると、User1 のパスワードが変更されます。

```
POST http://192.0.20.123/api/changeSelfPassword.json

{
  "aaaChangePassword" : {
    "attributes" : {
      "userName" : "User1",
      "oldPassword" : "p@$sw0rd",
      "newPassword" : "dr0ws$@p"
    }
  }
}
```

この例のように、正常に動作すると、空の `imdata` 要素が返されます。

```
{
    "totalCount" : "0",
    "imdata" : []
}
```

SSH キーの変更

SSH キーを変更するには、`aaa:changeSshKey` オブジェクトを変更する `changeSelfSshKey` API メソッドを送信します。次のオブジェクト プロパティがメッセージ本文に必要です。

- `userName`：ログイン ID。
- `name`：キーのシンボリック名。APIC はシングル ユーザで最大 32 の SSH キーをサポートしています。
- `data`：新しい SSH キー。

この例では、User1 によって送信されると、User1 の SSH キーが変更されます。

```
POST http://192.0.20.123/api/changeSelfSshKey.json

{
  "aaaChangeSshKey" : {
    "attributes" : {
      "userName" : "User1",
      "name" : "A",
      "data" : "ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAuKxY5E4we6uCR2z== key@example.com"
    }
  }
}
```

正常に動作すると、空の `imdata` 要素が返されます。

X.509 証明書の変更

X.509 証明書を変更するには、`aaa:changeX509Cert` オブジェクトを変更する `changeSelfX509Cert` API メソッドを送信します。次のオブジェクト プロパティがメッセージ本文に必要です。

- `userName`：ログイン ID。
- `name`：証明書のシンボリック名。APIC はシングル ユーザで最大 32 の X.509 証明書をサポートします。
- `data`：新しい X.509 証明書のデータ本文全体。

この例では、User1 によって送信されると、User1 の X.509 証明書が変更されます。

```
POST http://192.0.20.123/api/changeSelfX509Cert.json

{
  "aaaChangeX509Cert" : {
    "attributes" : {
      "userName" : "User1",
      "name" : "A",
      "data" : "-----BEGIN CERTIFICATE-----\nMIIE2TCCA8GgAwIBAgIKamlnsw

[EXAMPLE TRUNCATED]

       1BCIolblPFft6QKoSJFjB6thJksaE5/k3Npf\n-----END CERTIFICATE-----"
    }
  }
}
```

正常に動作すると、空の `imdata` 要素が返されます。

SSH キーまたは X.509 証明書の削除

キーまたは証明書を削除するには、削除するキーまたは証明書の名前を指定し、データ属性を空白にして、キーまたは証明書の変更メソッドを送信します。

この例では、User1 によって送信されると、User1 の SSH キーが削除されます。

```
POST http://192.0.20.123/api/changeSelfSshKey.json

{
  "aaaChangeSshKey" : {
    "attributes" : {
      "userName" : "User1",
      "name" : "A",
      "data" : ""
    }
  }
}
```

正常に動作すると、空の `imdata` 要素が返されます。

REST API ツール

管理情報モデルのリファレンス

管理情報モデル（MIM）には、システム内のすべての管理対象オブジェクトとそのプロパティが含まれます。For details, see the *Cisco APIC Management Information Model Reference Guide*.

MIT 内のオブジェクトを検索するために管理者がどのように MIM を使用できるかに関する例については、次の図を参照してください。

図 *3 : MIM* リファレンス

GUI 内の API 交換の表示

APIC グラフィカル ユーザ インターフェイス（GUI）でタスクを実行すると、GUI は内部 API メッセージを作成してタスクを実行するためのオペレーティング システムに送信します。APIC の組み込み型ツールである API インスペクタ を使用して、これらの API メッセージを表示およびコピーできます。ネットワーク管理者は、主要操作を自動化するためにこれらのメッセージを複製したり、API を使用する外部アプリケーションを開発するためにこれらのメッセージを例として使用できます。.

手順

ステップ 1 APIC GUI にログインします。

ステップ 2 APIC ウィンドウの右上隅で、「welcome, <name>」メッセージをクリックしてドロップダウン リストを表示します。

ステップ 3 ドロップダウン リストで、[Show API Inspector] を選択します。

[API Inspector] が新しいブラウザ ウィンドウで開きます。

ステップ 4 [API Inspector] ウィンドウの [Filters] ツールバーで、表示する API ログ メッセージのタイプを選択します。

表示されたメッセージは選択されたメッセージのタイプに応じて色分けされます。次のテーブルに、使用可能なメッセージ タイプを表示します。

名前	説明
trace	トレース メッセージを表示します。
debug	デバッグ メッセージを表示します。このタイプには、ほとんどの API コマンドと応答が含まれます。
info	情報メッセージを表示します。
warn	警告メッセージを表示します。
error	エラー メッセージを表示します。
fatal	重大メッセージを表示します。
all	このチェックボックスをオンにすると、他のチェックボックスすべてがオンになります。他のチェックボックスのいずれかをオフにすると、このチェックボックスもオフになります。

ステップ 5 [Search] ツールバーで、正確な文字列に対し表示されるメッセージまたは正規表現で表示されるメッセージを検索できます。

次の表に、検索のコントロールを示します。

名前	説明
検索	このテキスト ボックスに、直接検索の文字列を入力するか、または regex 検索の正規表現を入力します。入力に応じて、ログ リストの最初に一致したフィールドが強調表示されます。
Reset	[Search] テキスト ボックスの内容を削除するには、このボタンをクリックします。
Regex	[Search] テキスト ボックスの内容を検索の正規表現として使用するには、このチェックボックスをオンにします。
Match case	検索で大文字と小文字が区別されるようにするには、このチェックボックスをオンにします。
Disable	検索を無効にし、ログ リストの検索一致結果の強調表示をクリアするには、このチェックボックスをオンにします。
Next	ログ リストを次の一致したエントリまでスクロールするには、このボタンをクリックします。このボタンは、検索がアクティブである場合にのみ表示されます。
Previous	ログ リストを前の一致したエントリまでスクロールするには、このボタンをクリックします。このボタンは、検索がアクティブである場合にのみ表示されます。
Filter	一致しない行を非表示にするには、このチェックボックスをオンにします。このチェックボックスは、検索がアクティブである場合にのみ表示されます。
Highlight all	すべての一致したフィールドを強調表示するには、このチェックボックスをオンにします。このチェックボックスは、検索がアクティブである場合にのみ表示されます。

ステップ 6 [Options] ツールバーで、表示されるメッセージを並べ替えることができます。

次の表に、使用可能なオプションを示します。

名前	説明
Log	ロギングをイネーブルにするには、このチェックボックスをオンにします。
Wrap	ログ リストの水平スクロールを無効にするために行の折り返しをイネーブルにするには、このチェックボックスをオンにします。
Newest at the top	ログ エントリを逆の時系列で表示するには、このチェックボックスをオンにします。
Scroll to latest	最新のログ エントリに迅速にスクロールするには、このチェックボックスをオンにします。
Clear	ログ リストを削除するには、このボタンをクリックします。

名前	説明
Close	API インスペクタを閉じるには、このボタンをクリックします。

例

次の例では、API インスペクタ ウィンドウの 2 つのデバッグ メッセージを示します。

```
13:13:36 DEBUG - method: GET url: http://192.0.20.123/api/class/infraInfra.json
response: {"imdata":[{"infraInfra":{"attributes":{"instanceId":"0:0","childAction":"",
"dn":"uni/infra","lcOwn":"local","name":"","replTs":"never","status":""}}}]}

13:13:40 DEBUG - method: GET url: http://192.0.20.123/api/class/l3extDomP.json?
query-target=subtree&subscription=yes
response: {"subscriptionId":"72057598349672459","imdata":[]}
```

ブラウザのアドオンを使用した API のテスト

ブラウザの使用

API 要求をテストするために、ブラウザのアドオン ユーティリティを使用して HTTP メッセージを構築し、それを送信し、応答を検査できます。最も一般的なブラウザのアドオンとして利用可能な RESTful API クライアントでは、API との対話に使いやすいインターフェイスが提供されます。クライアントには次のものが含まれます。

- Firefox/Mozilla 用：Poster、RESTClient
- Chrome 用：高度な REST クライアント、Postman

ブラウザのアドオンでは、ペイロードデータ構造にトークンを含める必要がないように、セッション トークンが Cookie として渡されます。

cURL による API のテスト

URL 構文を使用してファイルを転送するツールである cURL を使用して、コンソールまたはコマンドライン スクリプトから API メッセージを送信できます。

POST メッセージを送信するには、JSON または XML コマンドの本文を含むファイルを作成し、次の形式で cURL コマンドを入力します。

```
curl -X POST --data "@<filename>" <URI>
```

ディスクリプタ ファイルの名前と API 操作の URI を指定する必要があります。

（注） ディスクリプタ ファイル名の前に「@」記号を必ず入力してください。

次に、ファイル「newtenant.json」で JSON データ構造を使用して、ExampleCorp という名前の新しいテナントを作成する例を示します。

```
curl -X POST --data "@newtenant.json"
https://apic-ip-address/api/mo/uni/tn-ExampleCorp.json
```

GET メッセージを送信するには、次の形式で cURL コマンドを入力します。

```
curl -X GET <URI>
```

次に、JSON 形式でテナントに関する情報を読み取る例を示します。

```
curl -X GET https://apic-ip-address/api/mo/uni/tn-ExampleCorp.json
```

（注） cURL でテストするときは、API にログインし、認証トークンを保存し、トークンを後続の API 操作に含める必要があります。

関連トピック

例：cURL による JSON API を使用したユーザの追加

『Cisco APIC Python SDK

Python API は Python プログラミング インターフェイスを基盤の REST API に提供することで、APIC およびネットワークを制御する独自のアプリケーションを展開することが可能になり、インフラストラクチャの自動化、管理、プログラミング性をより柔軟にできます。

Python API は、Python バージョン 2.7 をサポートしています。

詳細については、「*Cisco APIC Python Sdk*」、「*Cisco APIC Python SDK* のインストール」、「http://www.python-requests.org」を参照してください。

管理オブジェクト ブラウザの使用

Managed Object Browser、つまり Visore は、APIC に組み込まれたユーティリティで、ブラウザを使用した管理対象オブジェクト（MO）のグラフィカル表示が提供されます。Visore ユーティリティは、APIC REST API クエリー メソッドを使用して アプリケーション セントリック インフラストラクチャ ファブリック 内でアクティブな MO を参照するので、ユーザは情報を取得するために使用されたクエリーを確認できます。Visore ユーティリティは、設定を行うためには使用できません。

（注） Firefox、Chrome および Safari ブラウザでのみ、Visore アクセスがサポートされます。

Visore のブラウザ ページ

[Filter] 領域

フィルタ形式は大文字と小文字が区別されます。This area supports all simple APIC REST API query operations.

名前	説明
[Class or DN] フィールド	管理対象オブジェクトのオブジェクト クラス名または完全な識別名。
[Property] フィールド	結果をフィルタリングする管理対象オブジェクトのプロパティ。[Property] フィールドを空のままにすると、検索では特定のクラスのインスタンスすべてが返されます。
[Op] ドロップダウン リスト	結果をフィルタリングするプロパティの値の演算子。有効な演算子は次のとおりです。 • ==（等しい） • **!=**（等しくない） • <（より小さい） • >（より大きい） • ≤（以下） • ≥（以上） • **間** • **wildcard** • **anybit** • **allbits**
[Val1] フィールド	フィルタリングするプロパティの初期値。
[Val2] フィールド	フィルタリングする 2 番目の値。

[Display XML of Last Query] リンク

The **Display XML of last query** link displays the full APIC REST API translation of the most recent query run in Visore.

[Results] 領域

クエリーは URL で符号化されるため、ブラウザにクエリー結果のページをブックマークして再度表示できます。

（注） Many of the managed objects are only used internally and are not generally applicable to APIC REST API program development.

名前	説明
桃色の背景	個別の管理対象オブジェクトのインスタンスを切り離し、その下のオブジェクトのクラス名を表示します。
青色および緑色の背景	管理対象オブジェクトのプロパティ名を示します。
黄色およびベージュ色の背景	プロパティ名の値を示します。
[dn] プロパティ	オブジェクト モデルの各管理対象オブジェクトの絶対アドレス。
[dn] リンク	クリックすると、その dn のすべての管理対象オブジェクトが表示されます。
Class name link	クリックすると、そのクラスのすべての管理対象オブジェクトが表示されます。
←	クリックすると、管理対象オブジェクトの親オブジェクトに移動します。
→	クリックすると、管理対象オブジェクトの子オブジェクトに移動します。
疑問符	管理対象オブジェクトの XML API ドキュメントにリンクします。

Visore へのアクセス

手順

ステップ 1 サポートされているブラウザを開き、APIC の URL とその後に `/visore.html` を入力します。

例：

```
https://apic-ip-address/visore.html
```

ステップ 2 プロンプトが表示されたら、APIC CLI または GUI ユーザ インターフェイスへのログインと同じクレデンシャルを使用してログインします。

読み取り専用アカウントを使用できます。

Visore でのクエリーの実行

手順

ステップ 1 [Class or DN] テキスト ボックスに MO のクラスまたは DN 名を入力します。

ステップ 2 （任意） [Property] テキスト ボックスに MO のプロパティを入力し、[Op] テキスト ボックスに演算子を入力し、[Val1] および [Val2] テキスト ボックスに 1 個または 2 個の値を入力することでクエリーをフィルタリングすることができます。

ステップ 3 [Run Query] をクリックします。

Visore によってクエリーが APIC に送信され、要求された MO が表形式で表示されます。

ステップ 4 （任意） クエリーを実行した API コールを表示するには、[Display URI of last query] リンクをクリックします。

ステップ 5 （任意） クエリーからの API 応答データ構造を表示するには、[Display last response] リンクをクリックします。

ステップ 6 （任意） 表示された MO の親および子クラスを取得するには、MO 説明テーブルの [dn] フィールドで [<] および [>] アイコンをクリックします。

[>] をクリックすると、MO の子用のクエリーが APIC に送信されます。[<] をクリックすると、MO の親用のクエリーが送信されます。

ステップ 7 （任意） MO の統計情報、障害、または動作状態情報を表示するには、MO 説明テーブルの [dn] フィールドで追加のアイコンをクリックします。