# Cisco APIC REST API ユーザ ガイド

**初版**：2013 年 10 月 31 日

**最終更新**：2014 年 09 月 08 日

**シスコシステムズ合同会社**
〒107-6227 東京都港区赤坂9-7-1 ミッドタウン・タワー
http://www.cisco.com/jp
お問い合わせ先：シスコ コンタクトセンター
0120-092-255 （フリーコール、携帯・PHS含む）
電話受付時間：平日 10:00～12:00、13:00～17:00
http://www.cisco.com/jp/go/contactcenter/

【注意】シスコ製品をご使用になる前に、安全上の注意（**www.cisco.com/jp/go/safety_warning/**）をご確認ください。本書は、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。また、契約等の記述については、弊社販売パートナー、または、弊社担当者にご確認ください。

# 目 次

# はじめに

この前書きは、次の項で構成されています。



## このマニュアルについて

このマニュアルでは、APIC REST API を使用してコマンドおよびクエリーを作成して Application Policy Infrastructure Controller（APIC）に送信する方法について説明します。 このマニュアルでは、一連の共通タスクに関する例が提供されますが、Cisco Application Centric Infrastructure（ACI）ファブリックの完全な機能セットを設定する方法については説明せず、API のクラス、メソッドまたはデータ タイプの一覧表示はありません。

## 対象読者

このマニュアルは、プログラミングと API の使用について背景知識を持つソフトウェア エンジニアを対象としています。 エンジニアは、XML および JSON、RESTful Web サービス API、データ システム、ネットワーキング プロトコルおよびストレージ プロトコルの知識が必要です。

## 表記法

コマンドの説明には、次のような表記法が使用されます。

| 表記法 | 説明 |
|---|---|
| **bold** | 太字の文字は、表示どおりにユーザが入力するコマンドおよびキーワードです。 |
| *italic* | イタリック体の文字は、ユーザが値を入力する引数です。 |
| [x] | 省略可能な要素（キーワードまたは引数）は、角カッコで囲んで示しています。 |
| [x \| y] | いずれか 1 つを選択できる省略可能なキーワードや引数は、角カッコで囲み、縦棒で区切って示しています。 |
| {x \| y} | 必ずいずれか1つを選択しなければならない必須キーワードや引数は、波カッコで囲み、縦棒で区切って示しています。 |
| [x {y \| z}] | 角カッコまたは波カッコが入れ子になっている箇所は、任意または必須の要素内の任意または必須の選択肢であることを表します。 角カッコ内の波カッコと縦棒は、省略可能な要素内で選択すべき必須の要素を示しています。 |
| `variable` | ユーザが値を入力する変数であることを表します。イタリック体が使用できない場合に使用されます。 |
| string | 引用符を付けない一組の文字。 stringの前後には引用符を使用しません。引用符を使用すると、その引用符も含めてstringとみなされます。 |

例では、次の表記法を使用しています。

| 表記法 | 説明 |
|---|---|
| `screen フォント` | スイッチが表示する端末セッションおよび情報は、screenフォントで示しています。 |
| **`太字の screen フォント`** | ユーザが入力しなければならない情報は、太字の screen フォントで示しています。 |
| イタリック体の *screen* フォント | ユーザが値を指定する引数は、イタリック体の screen フォントで示しています。 |
| < > | パスワードのように出力されない文字は、山カッコ（< >）で囲んで示しています。 |
| [ ] | システム プロンプトに対するデフォルトの応答は、角カッコで囲んで示しています。 |

| 表記法 | 説明 |
|---|---|
| !、# | コードの先頭に感嘆符（!）またはポンド記号（#）がある場合には、コメント行であることを示します。 |

このマニュアルでは、次の表記法を使用しています。

（注） 「注釈」です。役立つ情報やこのマニュアルに記載されていない参照資料を紹介しています。

注意 「要注意」の意味です。機器の損傷またはデータ損失を予防するための注意事項が記述されています。

警告 安全上の重要事項

「危険」の意味です。人身事故を予防するための注意事項が記述されています。機器の取り扱い作業を行うときは、電気回路の危険性に注意し、一般的な事故防止対策に留意してください。各警告の最後に記載されているステートメント番号を基に、装置に付属の安全についての警告を参照してください。

これらの注意事項を保管しておいてください。

# 関連資料

API のクラス、プロパティ、データ タイプの詳細な参考資料については、Web ベース アプリケーションである『*Cisco APIC Management Information Model Reference*』を参照してください。

API を使用した多くの一般的なタスクの詳細な手順については、『*Cisco APIC Getting Started Guide*』で確認できます。

### Web ベースのマニュアル

- 『*Cisco APIC Management Information Model Reference*』
- 『*Cisco APIC Online Help Reference*』
- 『*Cisco ACI MIB Support List*』

### ダウンロード可能なマニュアル

- 『*Cisco Application Centric Infrastructure Release Notes*』
- 『*Cisco Application Centric Infrastructure Fundamentals Guide*』

- 『*Cisco APIC Getting Started Guide*』
- 『*Cisco APIC REST API* ユーザ ガイド』
- 『*Cisco APIC Command Line Interface User Guide*』
- 『*Cisco APIC Faults, Events, and System Message Guide*』
- 『*Cisco APIC Layer 4 to Layer 7 Device Package Development Guide*』
- 『*Cisco APIC Layer 4 to Layer 7 Services Deployment Guide*』
- 『*Cisco ACI Firmware Management Guide*』
- 『*Cisco ACI Troubleshooting Guide*』
- 『*Cisco ACI NX-OS Syslog Reference Guide*』
- 『*Cisco ACI Switch Command Reference, NX-OS Release 11.0*』
- 『*Cisco ACI MIB Quick Reference*』
- 『*Cisco Nexus CLI to Cisco APIC Mapping Guide*』
- 『*Installing the Cisco Application Virtual Switch with the Cisco APIC*』
- 『*Configuring the Cisco Application Virtual Switch using the Cisco APIC*』
- 『*Application Centric Infrastructure Fabric Hardware Installation Guide*』

# マニュアルに関するフィードバック

このマニュアルに関する技術的なフィードバック、または誤りや記載もれなどお気づきの点がございましたら、apic-docfeedback@cisco.com までご連絡ください。 ご協力をよろしくお願いいたします。

# マニュアルの入手方法およびテクニカル サポート

マニュアルの入手、Cisco Bug Search Tool（BST）の使用、サービス要求の送信、追加情報の収集の詳細については、『*What's New in Cisco Product Documentation*』を参照してください。このドキュメントは、http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html から入手できます。

『*What's New in Cisco Product Documentation*』に登録します。ここには、すべての新規および改訂済みの Cisco テクニカル マニュアルが RSS フィードとして掲載されており、コンテンツはリーダー アプリケーションを使用してデスクトップに直接配信されます。RSS フィードは無料のサービスです。

第 1 章

# APIC REST API の概要

この章の内容は、次のとおりです。



## Application Policy Infrastructure Controller について

Cisco Application Centric Infrastructure（ACI）は、外部エンドポイントの接続性がアプリケーションセントリックポリシーを通じて制御されグループ化された分散型のスケーラブルなマルチテナントインフラストラクチャです。 Application Policy Infrastructure Controller（APIC）は、ACI の自動化、管理、モニタリングおよびプログラマビリティの統合ポイントです。 APICは、インフラストラクチャの物理/仮想コンポーネント用の統合操作モデルを使用して、アプリケーションの展開、管理、およびモニタリングをあらゆる場所でサポートします。 APICは、アプリケーションの要件とポリシーに基づき、ネットワークのプロビジョニングおよび制御をプログラムで自動化します。また、これは幅広いクラウドネットワークに対する中央制御エンジンなので、管理が容易になり、アプリケーション ネットワークの定義および自動化の方法に柔軟性が得られます。 また、ノースバウンド Representational State Transfer（REST）API が提供されます。 APICは、多くのコントローラ インスタンスのクラスタとして実装される分散システムです。

## 管理情報モデル

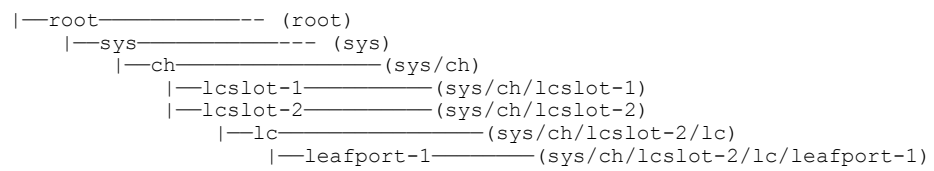
アプリケーション セントリック インフラストラクチャ ファブリック を構成するすべての物理および論理コンポーネントは、管理情報ツリー（MIT）とも呼ばれる階層型管理情報モデル（MIM）で表されます。 このツリー内の各ノードは、管理ステータスと動作ステータスを含む、管理対象オブジェクト（MO）またはオブジェクトのグループを表します。

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。 このツリー内の各ノードはMO で、ACI fabric内の各オブジェクトには、ツリー内のオブジェクトとその場所を記述する一意の識別名（DN）があります。 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。

設定ポリシーは、システム内のポリシーの大半を占め、さまざまな ACI fabric コンポーネントの設定を説明します。 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。

情報モデルは APIC によって論理モデルとして一元的に保存され、一方で各スイッチ ノードには具象モデルとしての完全なコピーが含まれます。 ユーザが APIC で設定を表すポリシーを作成すると、APIC は論理モデルを更新します。 次に APIC は、ユーザ ポリシーから十分に精緻化されたポリシーを作成し、そのポリシーを具象モデルが更新されるすべてのスイッチ ノードにプッシュする中間ステップを実行します。 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。

次のリーフ スイッチ ポートの分岐図は、ACI ファブリック MIT の topRoot から始まり、2 つのライン モジュール スロット（スロット 2 にライン モジュール 1 つ）を備えたシャーシを構成する階層を示します。




# 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。

### 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 |

# APIC REST API について

APIC REST API は、 Representational State Transfer（REST）アーキテクチャを使用する Application Policy Infrastructure Controller（APIC）へのプログラマチック インターフェイスです。 API は、JavaScript Object Notation（JSON）または Extensible Markup Language（XML）のドキュメントを含む HTTP または HTTPS メッセージを受け入れて返します。 プログラミング言語を使用して、API メソッドまたは管理対象オブジェクト（MO）の説明を含むメッセージおよび JSON または XML ドキュメントを生成できます。

API モデルにより、アプリケーション開発に対し主要な機能が提供されます。 Cisco Application Centric Infrastructure（ACI）ファブリックの設定および状態の情報は、管理情報ツリー（MIT）として知られる階層型ツリー構造に保存されます。それには API からアクセスできます。 単一のオブジェクトまたはオブジェクト サブツリーで変更を行うことができます。 API コールを使用して、スイッチ、アダプタ、ポリシー、およびその他のハードウェアおよびソフトウェア コンポーネントの設定に変更を加えることができます。

API は寛容モードで動作しますが、それは欠けている属性が内部のデータ管理エンジン（DME）で維持されるデフォルト値で代入されることを意味します（該当する場合）。 DME は不正な属性を検証し拒否します。 API はアトミックでもあります。 複数の MO を設定する場合に（たとえば、仮想 NIC）すべての MO を設定できないと、API は動作を停止します。 これにより、設定は前の状態に戻り、API 要求をリッスンする API 操作は停止され、エラー コードが返されます。

MO およびプロパティへのアップデートは既存のオブジェクト モデルに準拠し、下位互換性を確保します。 既存のプロパティが製品のアップグレード中に変更された場合、アップグレード後、データベースのロード中に管理されます。 新しいプロパティにはデフォルト値が割り当てられます。

完全なイベントサブスクリプションがイネーブルになります。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。クエリー API を使用すると、そのクエリー結果の今後の変更へのサブスクリプションを作成できます。

API の動作はトランザクション型で、単一のデータ モデルで終了します。 APIC がステート アップデートなどのすべてのエンドポイントの通信を担うため、ユーザはエンドポイントと直接通信できません。 このように、開発者は、独立した個々のコンポーネント設定の管理タスクを行う必要はありません。

API モデルには、これらのプログラマチック エンティティが含まれます。

- クラス：管理情報ツリーのオブジェクトのプロパティおよび状態を定義するテンプレート。
- メソッド：1 つまたは複数のオブジェクトに対して API が実行するアクション。
- タイプ：オブジェクト ステート（たとえば、equipmentPresence）に値をマッピングするオブジェクトのプロパティ。

一般的な要求は DME に送信され、トランザクタ キューにファーストイン ファーストアウト（FIFO）の順序で配置されます。 トランザクタはこのキューから要求を取得し、要求を解釈して認可チェックを実行します。 要求の確認後、トランザクタが MIT を更新します。 このすべての動作は、1 つのトランザクションで行われます。

（注） API のクラス、プロパティ、データ タイプの詳細な参考資料については、Web ベース アプリケーションである『*Cisco APIC Management Information Model Reference*』を参照してください。

第 2 章

# APIC REST API の使用

（注） **セキュリティ上の理由により、API 通信に対して HTTPS のみがデフォルト モードとして有効になっています。** 必要に応じて、HTTP および HTTP-to-HTTPS のリダイレクションをイネーブルにできますが、安全性は低下します。 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 コンポーネントとインタラクションの説明で HTTP に言及します。

この章の内容は、次のとおりです。



## 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 本文には、要求されたデータ、要求されたアクションの確認、またはエラー情報を含む 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） **セキュリティ上の理由により、API 通信に対して HTTPS のみがデフォルト モードとして有効になっています。** 必要に応じて、HTTP および HTTP-to-HTTPS のリダイレクションをイネーブルにできますが、安全性は低下します。 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 コンポーネントとインタラクションの説明で HTTP に言及します。

### 要求メソッド

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 ヘッダーに Access-Control-Allow-Methods:POST,GET,OPTIONS のみが表示される場合があります。

### 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。 HTTP の [Content-Type] および [Accept] ヘッダーは API によって無視されます。

# API コマンドまたはクエリー URI の作成

次の形式の Uniform Resource Identifier（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**}[**?***options*]

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**}[**?***options*]

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://192.0.20.123/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, （59 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://192.0.20.123/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://192.0.20.123/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 と残りの相対名コンポーネントに POST 送信することもできます。

```
POST https://192.0.20.123/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://192.0.20.123/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://192.0.20.123/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。
- 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。

# 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。

# 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され、このメソッドはユーザをログインし、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。 複数の AAA ログイン ドメインが設定されている場合、ユーザ名の前に **apic:***domain*\\ を追加する必要があります。
- **aaaRefresh**：メッセージ本文なしの GET メッセージ、またはメッセージ本文が **aaaLogin** の POST メッセージとして送信され、このメソッドはセッション タイマーをリセットします。 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され、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され、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://192.0.20.123/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]
    ...
}
```

前の例では、**refreshTimeoutSeconds** 属性はセッション タイムアウト時間が 300 秒であることを示します。

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://192.0.20.123/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://192.0.20.123/api/aaaLogin.json

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

### 関連トピック

例：JSON API を使用したユーザの認証, （25 ページ）
例：XML API を使用したユーザの認証, （24 ページ）

# 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「url-token」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「url-token」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?url-token=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
.
.
.
```

# 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。

# 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*]...**

このステートメントは、[query-target] オプションが **self** 以外の範囲で使用される場合に考慮されるオブジェクト クラスを指定します。 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。

サブツリーの情報を要求するには、次のように **query-target=subtree** を **target-subtree-class** と組み合わせ、特定のサブツリーを示します。

```
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。 情報は、APIC ファームウェア ステータス コンテナ firmware:IfcFwStatusCont の firmware:IfcRunning サブツリー（子）オブジェクトに含まれています。

```
GET https://192.0.20.123/api/class/firmwareIfcFwStatusCont.json?
    query-target=subtree&target-subtree-class=firmwareIfcRunning
```

**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。 このステートメントは、単独で使用するか、または **query-target** ステートメントの後ろに適用できます。

**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]：（非デフォルト）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。

**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。

たとえば、障害関連サブツリーを含めるには、リストに [fault] を指定します。 障害関連サブツリーのみを返し他の最上位の MO 情報は返さないようにするには、次の例に示すようにリストに [fault,no-scoped] を指定します。

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

### 関連トピック

クエリー フィルタ式の作成, （15 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, （53 ページ）

# 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),...)
```

### 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 | 等しい |
| 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 |

| 演算子 | 説明 |
|---|---|
| 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 | ワイルドカード |
| 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://192.0.20.123/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://192.0.20.123/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://192.0.20.123/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://192.0.20.123/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://192.0.20.123/api/class/aaaDomain.json?
    query-target-filter=
        and(ne(aaaDomain.name,"infra"),
            ne(aaaDomain.name,"common"))
```

# 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://192.0.20.123/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**=*number-of-objects-per-page*

`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**=*page-number*

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://192.0.20.123/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。 複数の APIC インスタンスとの複数のクエリー サブスクリプションをサポートするのに必要なのは単一の WebSocket 接続のみです。 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。

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://192.0.20.123/socket%TOKEN%);
```

URI では、**%TOKEN%** が現在の API セッション トークン（Cookie）です。 次に、トークン付きの URI の例を示します。

```
https://192.0.20.123/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://192.0.20.123/api/class/fvTenant.json?subscription=yes
```

クエリー応答には、サブスクリプションの識別子、**subscriptionId** が含まれ、サブスクリプションを更新し、このサブスクリプションから今後の通知を識別するために使用できます。

```
{
  "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。 サブスクリプションを更新するには、次の例のように、**subscriptionId** に相当するパラメータ **id** を使用して、HTTP GET メッセージを API 方式 **subscriptionRefresh** に送信します。

```
GET https://192.0.20.123/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。

# オブジェクトとプロパティの作成、変更、および削除

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

API コマンドを送信すると、APIC は MIM スキーマに適合するかどうかコマンドを確認します。API コマンドが MIM スキーマに違反している場合、APIC はコマンドを拒否し、エラー メッセー

ジを返します。 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） 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 のイネーブル化またはディセーブル化
- HTTPS へのリダイレクションの状態の変更
- キー リングなどの PKI の変更

ヒント このマニュアルで示される API の例に加えて、API を使用する多くの一般的なタスクの詳細な例を『*Cisco APIC Getting Started Guide*』で確認できます。

# タグの使用

API 操作を簡略化するために、オブジェクトにタグを割り当てることができます。 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。 API 操作では、識別名（DN）の代わりに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。

### 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://192.0.20.123/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://192.0.20.123/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://192.0.20.123/api/tag/orgs.xml
```

### タグ使用の追加例

（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://192.0.20.123/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://192.0.20.123/api/tag/tenants.xml

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

第 3 章

# 一般的なタスクの実行

この章の内容は、次のとおりです。



## 一般的なタスクの実行

（注） API を使用した多くの一般的なタスクの詳細な手順については、『*Cisco APIC Getting Started Guide*』で確認できます。

ここでは、テナントを作成し、アプリケーションを展開するのに必要な一般的なタスクの簡単な例を示します。 これらは一般的な手順です。

- ログイン：セッションを開き設定タスクを実行します。
- テナントの作成：アプリケーションをホストするテナント会社のファブリック内でプライベート ドメインを作成します。
- ユーザの作成：テナントのアプリケーションを導入し維持するテナントの管理者のアカウントを作成します。
- アプリケーションの導入：エンドポイント グループとコントラクトをテナント ネットワーク内で設定し、アプリケーションをホストするのに必要なサーバを接続します。

（注） 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。

# ログイン

API セッションの認証と維持, （10 ページ） に説明するように、データ構造内の有効なユーザ名とパスワードを **aaaLogin** API メソッドに送信することによって APIC REST API にログインできます。 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。

## 例：XML API を使用したユーザの認証

次に、ユーザ名とパスワードを含む XML データ構造を使用してユーザをログインする例を示します。

```
POST https://192.0.20.123/api/aaaLogin.xml

<aaaUser name="georgewa" pwd="paSSword1"/>
```

次に、ログイン成功時の応答例を示します。

```
<imdata>
  <aaaLogin
    token=
"mk/i+LHZMhFQ47YyMlfZOjmUQQV4PXwFAY/6Dyg5Bm7PC0SVdm
MVWeOlLw69AoNoRVziR8GQeNqPbX/Vu92PxWyPjGx7kOQpoTaUY
P4Y2NkBwRfl5fEDwUe5WnGl3VoBXF4nrc8QSG8N1UWCuIFIyt9O
OzgsLG58fTf4OPCUqQL="
    refreshTimeoutSeconds="300" lastName="Washington" firstName="George">
      <aaaUserDomain name="all" rolesR="0" rolesW="2">
        <aaaReadRoles/>
        <aaaWriteRoles>
          <role name="admin"/>
        </aaaWriteRoles>
      </aaaUserDomain>
      <aaaUserDomain name="common" rolesR="0" rolesW="1">
        <aaaReadRoles/>
        <aaaWriteRoles>
          <role name="read-only"/>
        </aaaWriteRoles>
      </aaaUserDomain>
    <DnDomainMapEntry dn="uni/tn-common" readPrivileges="0" writePrivileges="3"/>
  </aaaLogin>
</imdata>
```

応答には、ユーザに関するドメインとロール情報とともにセッションのタイムアウト時間とセッション トークンが含まれます。

次に、**aaaRefresh** API メソッドを使用してセッションをリフレッシュする例を示します。

```
GET https://192.0.20.123/api/aaaRefresh.xml
```

次に、セッションのリフレッシュが成功したときの応答例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<imdata>
  <aaaLogin
    token=
"mk/i+LHZMhFQ47YyMlfZOua51veZ5mdbJAbHG7vjKVJ7QAuUDW
YGGkAdwSxjjcPYK/4Hav3QdZN/E7zJ/Rhc87tmpihVWpkP42VqU
D4aeELVQULz97ZpKnGfF0ki9vqBYkCYRg0vMstXO8IhxNYLtlC8
rxzgZFti+6biMQ+boRx="
    refreshTimeoutSeconds="0"
    lastName=""
    firstName=""/>
</imdata>
```

次の例では、**aaaLogout API** メソッドを使用してユーザをログアウトします。

```
POST https://192.0.20.123/api/aaaLogout.json

<aaaUser name="admin" />
```

次に、ログアウト成功時の応答例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<imdata>
</imdata>
```

# 例：JSON API を使用したユーザの認証

次に、ユーザ名とパスワードを含む JSON データ構造を使用してユーザをログインする例を示します。

```
POST https://192.0.20.123/api/aaaLogin.json

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

次に、ログイン成功時の応答例を示します。

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

```
        },
        "children":[{
            "aaaUserDomain":{
              "attributes":{
                "name":"all",
                "rolesR":"0",
                "rolesW":"2"
              },
              "children":[{
                  "aaaReadRoles":{
                    "attributes":{}
                  }
                }, {
                  "aaaWriteRoles":{
                    "attributes":{},
                    "children":[{
                        "role":{
                          "attributes":{
                            "name":"admin"
                          }
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }, {
            "aaaUserDomain":{
              "attributes":{
                "name":"common",
                "rolesR":"0",
                "rolesW":"1"
              },
              "children":[{
                  "aaaReadRoles":{
                    "attributes":{}
                  }
                }, {
                  "aaaWriteRoles":{
                    "attributes":{},
                    "children":[{
                        "role":{
                          "attributes":{
                            "name":"read-only"
                          }
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }, {
            "DnDomainMapEntry":{
              "attributes":{
                "dn":"uni/tn-common",
                "readPrivileges":"0",
                "writePrivileges":"3"
              }
            }
          }
        ]
      }
    }
  ]
}
```


応答には、ユーザに関するドメインとロール情報とともにセッションのタイムアウト時間とセッション トークンが含まれます。

次に、**aaaRefresh** API メソッドを使用してセッションをリフレッシュする例を示します。

```
GET https://192.0.20.123/api/aaaRefresh.json
```

次に、セッションのリフレッシュが成功したときの応答例を示します。

```
{
  "imdata":[{
      "aaaLogin":{
        "attributes":{
          "token":
"Zs8jf2rcseiiYydx9WbyRVOScWs6c0Orti7EBg1H02jt0JV9oe
icX1ULLquI+LCN/93cAf46AErXpT4kN0mrCQRcCMRAl6m0ObT/r
G4D9UTm6/NNnqzhntCWTKt9IovTFDpD8Ocx1oXn2jpmS6PYDBJt
Wp4rIAu1lsdM8BA24g6=",
          "refreshTimeoutSeconds":"0",
          "lastName":"",
          "firstName":""
        }
      }
    }
  ]
}
```

次の例では、**aaaLogout** API メソッドを使用してユーザをログアウトします。

```
POST https://192.0.20.123/api/aaaLogout.json

{
  "aaaUser":{
    "attributes":{
      "name":"georgewa"
    }
  }
}
```

次に、ログアウト成功時の応答例を示します。

```
{
  "imdata":[]
}
```

# テナントの追加

テナントは仮想ファブリック内のポリシー所有者です。テナントは、プライベートエンティティまたは共有エンティティにすることができます。たとえば、安全に分割されたプライベートテナントまたは他のテナントにより共有されるコンテキストおよびブリッジ ドメインを持つテナントを作成できます。テナントの共有タイプは通常、common、default または infra という名前が付いています。

管理情報モデルでは、テナントはクラス fv:Tenant の管理対象オブジェクト（MO）で表されます。『*Cisco APIC Management Information Model Reference*』に従って、fv:Tenant クラスのオブジェクトはポリシー解決のユニバース（uni）クラスの子で、`uni/tn-[name]` の識別名（DN）形式を持っています。

次の例に、XML および JSON を使用して ExampleCorp という名前の新しいテナントを追加する方法を示します。

# 例：XML API を使用したテナントの追加

新しいテナントを作成するには、メッセージの本文または URI でクラスと十分な命名情報を指定する必要があります。

XML API を使用して ExampleCorp という名前の新しいテナントを作成するには、次の HTTP POST メッセージを送信します。

```
POST https://192.0.20.123/api/mo/uni.xml

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

または、次の例のように URI でテナントに名前を付けることができます。

```
POST https://192.0.20.123/api/mo/uni/tn-ExampleCorp.xml

<fvTenant />
```

（`?rsp-subtree=modified` を POST URI に追加することで）応答が要求された場合は、成功した操作により次の応答本文が返されます。

```
<imdata>
    <fvTenant
        instanceId="0:0"
        childAction="deleteNonPresent"
        dn="uni/tn-ExampleCorp"
        lcOwn="local"
        name="ExampleCorp"
        replTs="never"
        rn=""
        status="created"
    />
</imdata>
```

テナントを削除するには、次の HTTP DELETE メッセージを送信します。

```
DELETE https://192.0.20.123/api/mo/uni/tn-ExampleCorp.xml
```

または、次の例のように、fv:Tenant 属性内に十分な命名情報と `status="deleted"` が含まれる HTTP POST メッセージを送信できます。

```
POST https://192.0.20.123/api/mo/uni.xml

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

# 例：JSON API を使用したテナントの追加

新しいテナントを作成するには、メッセージの本文または URI でクラスと十分な命名情報を指定する必要があります。

JSON API を使用して新しいテナントを作成するには、次の HTTP POST メッセージを送信します。

```
POST https://192.0.20.123/api/mo/uni.json

{
```

```
  "fvTenant" : {
    "attributes" : {
      "name" : "ExampleCorp"
    }
  }
}
```

または、次の例のように URI でテナントに名前を付けることができます。

```
POST https://192.0.20.123/api/mo/uni/tn-ExampleCorp.json

{
  "fvTenant" : {
    "attributes" : {
    }
  }
}
```

（`?rsp-subtree=modified` を POST URI に追加することで）応答が要求された場合は、成功した操作により次の応答本文が返されます。

```
{
  "imdata" :
  [{
      "fvTenant" : {
        "attributes" : {
          "instanceId" : "0:0",
          "childAction" : "deleteNonPresent",
          "dn" : "uni/tn-ExampleCorp",
          "lcOwn" : "local",
          "name" : "ExampleCorp",
          "replTs" : "never",
          "rn" : "",
          "status" : "created"
        }
      }
    }
  ]
}
```

テナントを削除するには、次の HTTP DELETE メッセージを送信します。

```
DELETE https://192.0.20.123/api/mo/uni/tn-ExampleCorp.json
```

または、次の例のように、fv:Tenant 属性内に十分な命名情報と `"status" : "deleted"` が含まれる HTTP POST メッセージを送信できます。

```
POST https://192.0.20.123/api/mo/uni.json

{
  "fvTenant" : {
    "attributes" : {
      "name" : "ExampleCorp",
      "status" : "deleted"
    }
  }
}
```

# ユーザの追加

APICユーザアカウントには、名前とパスワードに加えて、ユーザの権限レベル（ロール）とファブリック内のユーザ制御の範囲（ドメイン）に関する情報が含まれます。 たとえば、クラウドの管理者はテナントを作成して追加の管理者を割り当てる機能を含むグローバルな制御が可能な一方で、テナントの管理者は通常、テナントのドメインまたはネットワークに制限されます。 ユーザ アカウントを設定するときは、パスワードの有効期限ポリシーなどの追加の設定項目を指定したり、それらの項目を省略してデフォルトの設定を使用することができます。

管理情報モデルでは、APIC ユーザはクラス aaa:User の管理対象オブジェクト（MO）で表されます。『*Cisco APIC Management Information Model Reference*』に従って、aaa:User クラスのオブジェクトには `uni/userext/user-[name]` の識別名（DN）形式があります。 aaa:User クラスの直接プロパティ（属性）には、ユーザ ドメインまたは権限レベルのフィールドが含まれませんが、このクラスのリファレンスは aaa:UserDomain の子クラス（サブツリー）を示し、それには aaa:UserRole の子クラスがあります。 新しいユーザを追加する API コマンド構造には、次の階層でこれらの子クラスにユーザ ドメインおよび権限レベルの設定を含めることができます。

- aaa:User：ユーザ オブジェクトには、1 つ以上のユーザ ドメインの子オブジェクトを含めることができます。
  - aaa:UserDomain：ユーザ ドメイン オブジェクトには、1 つ以上のユーザ ロール オブジェクトを含めることができます。
    - aaa:UserRole：ユーザ ロール オブジェクトは、ユーザの権限レベルを指定します。

次の例に、XML および JSON を使用して新しいユーザを管理者権限を持つユーザ ドメイン ExampleCorp に追加する方法を示します。 追加の例に、コマンドラインからタスクを実行するための cURL コマンドの使用方法を示します。

## 例：XML API を使用したユーザの追加

新しいオブジェクトを作成するには、URI またはメッセージの本文でクラスと十分な命名情報を指定する必要があります。 この例では、URI には DN が含まれ、メッセージの本文にはオブジェクト クラスが含まれます。

XML API を使用して新しいユーザを作成および設定するには、次の HTTP POST メッセージを送信します。

```
POST https://192.0.20.123/api/mo/uni/userext/user-georgewa.xml

<aaaUser
  pwd="password1"
  firstName="George"
  lastName="Washington"
  phone="4085551212"
  email="georgewa@example.com" >
    <aaaUserDomain name="ExampleCorp" >
        <aaaUserRole name="admin" />
    </aaaUserDomain>
```

```
</aaaUser>
```

または、/api/mo.xml に POST 送信して DN 属性 `dn="uni/userext/user-georgewa"` を提供するか、または /api/mo/uni/userext.xml に POST 送信して名前属性 `name="georgewa"` を提供することができます。

このコマンド構造の子セクションでは、新しいユーザがそのユーザ ドメインの管理者ロールを持つExampleCorpユーザ ドメイン内に配置されるように設定されます。 新しいユーザを作成するときは、この例に示す情報より少ない情報または多い情報を指定できます。 追加属性は、後続の操作で設定できます。

（`?rsp-subtree=modified` を POST URI に追加することで）応答が要求された場合は、成功した操作により次の応答本文が返されます。

```
<imdata>
  <aaaUser
    instanceId="0:0"
    accountStatus="active"
    childAction="deleteNonPresent"
    clearPwdHistory="no"
    descr=""
    dn="uni/userext/user-georgewa"
    email="georgewa@example.com"
    encPwd=""
    expiration="never"
    expires="no"
    firstName="George"
    intId="none"
    lastName="Washington"
    lcOwn="local"
    name="georgewa"
    phone="4085551212"
    pwd="password1"
    pwdLifeTime="no-password-expire"
    pwdSet="no"
    replTs="never"
    rn=""
    status="created" >
      <aaaUserDomain
        instanceId="0:0"
        childAction="deleteNonPresent"
        descr="" intId="none"
        lcOwn="local"
        name="common"
        replTs="never"
        rn="userdomain-common"
        status="created" >
          <aaaUserRole
            instanceId="0:0"
            childAction="deleteNonPresent"
            descr=""
            intId="none"
            lcOwn="local"
            name="read-only"
            privType="readPriv"
            replTs="never"
            rn="role-read-only"
            status="created" />
          </aaaUserDomain>
      <aaaUserDomain
        instanceId="0:0"
        childAction="deleteNonPresent"
        descr=""
        intId="none"
        lcOwn="local"
        name="ExampleCorp"
        replTs="never"
```

```
        rn="userdomain-ExampleCorp"
        status="created" >
          <aaaUserRole
            instanceId="0:0"
            childAction="deleteNonPresent"
            descr=""
            intId="none"
            lcOwn="local"
            name="admin"
            privType="readPriv"
            replTs="never"
            rn="role-admin"
            status="created" />
      </aaaUserDomain>
  </aaaUser>
</imdata>
```

この応答は、ユーザが 2 つのユーザ ドメインに作成され追加されたことを示します。 要求通りに、ユーザには ExampleCorp ユーザ ドメインでの管理者権限があります。 デフォルトでは、ユーザには共通のユーザ ドメインでの読み取り専用権限もあります。

ユーザを削除するには、次の HTTP DELETE メッセージを送信します。

```
DELETE https://192.0.20.123/api/mo/uni/userext/user-georgewa.xml
```

または、次の例のように、aaa:User 属性内に十分な命名情報と `status="deleted"` が含まれる HTTP POST メッセージを送信できます。

```
POST https://192.0.20.123/api/mo.xml

<aaaUser dn="uni/userext/user-georgewa" status="deleted" />
```

# 例：JSON API を使用したユーザの追加

新しいオブジェクトを作成するには、URI またはメッセージの本文でクラスと十分な命名情報を指定する必要があります。 この例では、URI には DN が含まれ、メッセージの本文にはオブジェクト クラスが含まれます。

JSON API を使用して新しいユーザを作成および設定するには、次の HTTP POST メッセージを送信します。

```
POST https://192.0.20.123/api/mo/uni/userext/user-georgewa.json

{
  "aaaUser" : {
    "attributes" : {
      "pwd" : "password1",
      "firstName" : "George"
      "lastName" : "Washington",
      "phone" : "4085551212",
      "email" : "georgewa@example.com"
    },
    "children" : [{
        "aaaUserDomain" : {
          "attributes" : {
            "name" : "ExampleCorp"
          },
          "children" : [{
              "aaaUserRole" : {
                "attributes" : {
                  "name" : "admin"
```

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

または、/api/mo.json に POST 送信して DN 属性 `"dn":"uni/userext/user-georgewa"` を提供するか、または /api/mo/uni/userext.json に POST 送信して名前属性 `"name":"georgewa"` を提供することができます。

このコマンド構造の子セクションでは、新しいユーザがそのユーザ ドメインの管理者ロールを持つ ExampleCorp ユーザ ドメイン内に配置されるように設定されます。 新しいユーザを作成するときは、この例に示す情報より少ない情報または多い情報を指定できます。 追加属性は、後続の操作で設定できます。

（`?rsp-subtree=modified` を POST URI に追加することで）応答が要求された場合は、成功した操作により次の応答本文が返されます。

```
{
  "imdata" : [{
      "aaaUser" : {
        "attributes" : {
          "instanceId" : "0:0",
          "accountStatus" : "active",
          "childAction" : "deleteNonPresent",
          "clearPwdHistory" : "no",
          "descr" : "",
          "dn" : "uni/userext/user-georgewa",
          "email" : "georgewa@example.com",
          "encPwd" : "",
          "expiration" : "never",
          "expires" : "no",
          "firstName" : "George",
          "intId" : "none",
          "lastName" : "Washington",
          "lcOwn" : "local",
          "name" : "georgewa",
          "phone" : "5551212",
          "pwd" : "password1",
          "pwdLifeTime" : "no-password-expire",
          "pwdSet" : "no",
          "replTs" : "never",
          "rn" : "",
          "status" : "created"
        },
        "children" : [{
            "aaaUserDomain" : {
              "attributes" : {
                "instanceId" : "0:0",
                "childAction" : "deleteNonPresent",
                "descr" : "",
                "dn" : "",
                "intId" : "none",
                "lcOwn" : "local",
                "name" : "common",
                "replTs" : "never",
                "rn" : "userdomain-common",
                "status" : "created"
              },
              "children" : [{
                  "aaaUserRole" : {
                    "attributes" : {
                      "instanceId" : "0:0",
```

```
                        "childAction" : "deleteNonPresent",
                        "descr" : "",
                        "dn" : "",
                        "intId" : "none",
                        "lcOwn" : "local",
                        "name" : "read-only",
                        "privType" : "readPriv",
                        "replTs" : "never",
                        "rn" : "role-read-only",
                        "status" : "created"
                      }
                    }
                  }
                ]
              }
            }, {
              "aaaUserDomain" : {
                "attributes" : {
                  "instanceId" : "0:0",
                  "childAction" : "deleteNonPresent",
                  "descr" : "",
                  "dn" : "",
                  "intId" : "none",
                  "lcOwn" : "local",
                  "name" : "ExampleCorp",
                  "replTs" : "never",
                  "rn" : "userdomain-ExampleCorp",
                  "status" : "created"
                },
                "children" : [{
                    "aaaUserRole" : {
                      "attributes" : {
                        "instanceId" : "0:0",
                        "childAction" : "deleteNonPresent",
                        "descr" : "",
                        "dn" : "",
                        "intId" : "none",
                        "lcOwn" : "local",
                        "name" : "admin",
                        "privType" : "writePriv",
                        "replTs" : "never",
                        "rn" : "role-admin",
                        "status" : "created"
                      }
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    ]
}
```

この応答は、ユーザが 2 つのユーザ ドメインに作成され追加されたことを示します。 要求通りに、ユーザには ExampleCorp ユーザ ドメインでの管理者権限があります。 デフォルトでは、ユーザには共通のユーザ ドメインでの読み取り専用権限もあります。

ユーザを削除するには、次の HTTP DELETE メッセージを送信します。

```
DELETE https://192.0.20.123/api/mo/uni/userext/user-georgewa.json
```

または、次の例のように、aaa:User 属性内に十分な命名情報と `"status":"deleted"` が含まれる HTTP POST メッセージを送信できます。

```
POST https://192.0.20.123/api/mo.json
```

```
{
  "aaaUser" : {
    "attributes" : {
      "dn" : "uni/userext/user-georgewa",
      "status" : "deleted"
    }
  }
}
```

# 例：cURL による JSON API を使用したユーザの追加

新しいオブジェクトを作成するには、URI またはメッセージの本文でクラスと十分な命名情報を指定する必要があります。 この例では、URI には DN が含まれ、メッセージの本文にはオブジェクト クラスが含まれます。

「例：JSON API を使用したユーザの追加, （32 ページ）」に示す形式で JSON 構造を含むローカル ファイルを作成します。 この例では、ファイル名は newuser.json で、そのファイルは別のディレクトリにあります。

cURL コマンドラインで、HTTP 方式（POST など）、ローカル ファイルのパスと名前、および API 操作の URI を指定する必要があります。

curl コマンドを使用して、次の例のように POST メッセージ内の JSON ファイルを API に送信します。

```
curl -X POST --data "@../users/newuser.json"
https://192.0.20.123/api/mo/uni/userext/user-georgewa.json
```

（注） ファイル名の前に@記号を必ず入力してください。cURL ツールは無料でオープン ソフトウェアです。 cURL の使用に関する詳細情報はオンラインで確認できます。

（`?rsp-subtree=modified` を POST URI に追加することで）応答が要求された場合は、成功した操作により「例：JSON API を使用したユーザの追加, （32 ページ）」に示す応答本文が返されます。

ユーザを削除するには、JSON ファイルを編集して `"status":"deleted"` の aaaUser 属性を追加して、コマンドを再度送信します。

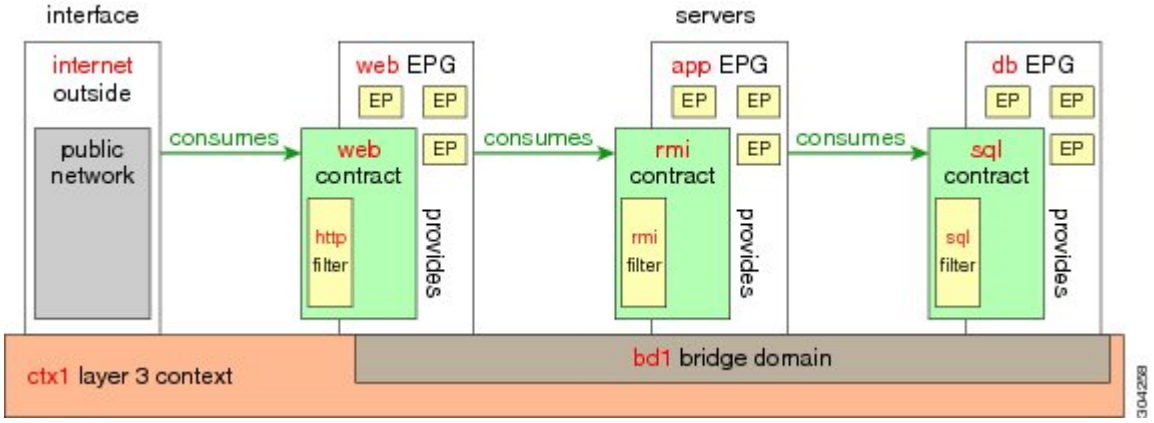
または、この HTTP DELETE メッセージを送信できます。

```
curl -X DELETE https://192.0.20.123/api/mo/uni/userext/user-georgewa.json
```

# 3 層アプリケーションのエンドポイント グループおよびコントラクトの導入

Cisco Application Centric Infrastructure（ACI）ファブリックの一般的な用途は、テナント ネットワーク内で 3 層アプリケーションをホストすることです。 APIC REST API を使用して、このようなア

プリケーションのエンドポイントグループとコントラクトを設定できます。この例では、アプリケーションは、次の図に示すように 3 台のサーバ、Web サーバ、アプリケーション サーバ、およびデータベース サーバを使用して実装されます。




以下は、構成の主要コンポーネントです。

- エンドポイントグループ（EPG）：各サーバは、各エンドポイントグループに関連付けられたコントラクトに従って他のエンドポイントグループからトラフィックを供給されるか消費するエンドポイント グループを表します。
- コントラクト：エンドポイントで送受信されるトラフィックはコントラクトによって抑制され、コントラクトはエンドポイントから提供されたトラフィックとエンドポイントで消費されるトラフィックのフィルタ セットを適用します。
- フィルタ：フィルタは、フィルタを含むコントラクトにより許可または拒否されるデータ プロトコルを指定します。

次のネットワーク コンポーネントは、すでに存在すると想定されています。

- ブリッジ ドメイン（BD）：レイヤ 3 コンテキストに関連付けられたレイヤ 2 ネットワーク
- コンテキスト：サーバ間の通信用のレイヤ 3 プライベート ネットワーク。
- 外部ポート：アプリケーションへの外部アクセス用のレイヤ 3 ネットワーク ポート。
- Virtual Machine Manager（VMM）およびハイパーバイザ コンポーネント
- VLAN

（注） 3 層アプリケーションの導入に関する拡張された包括的な例については、『*Cisco APIC Getting Started Guide*』を参照してください。

構成情報は、テナントの識別名（DN）に送信される単一の XML 構造に含まれています。この例では、テナントの名前は ExampleCorp です。XML API を使用してアプリケーションを展開するには、次の HTTP POST メッセージを送信します。

```
POST https://192.0.20.123/api/mo/uni/tn-ExampleCorp.xml
```

次の XML 構造を POST メッセージの本文に含めます。

```
<fvTenant name="ExampleCorp">

    <fvAp name="OnlineStore">
        <fvAEPg name="web">
            <fvRsBd tnFvBDName="bd1"/>
            <fvRsCons tnVzBrCPName="rmi"/>
            <fvRsProv tnVzBrCPName="web"/>
            <fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter"/>
        </fvAEPg>

        <fvAEPg name="db">
            <fvRsBd tnFvBDName="bd1"/>
            <fvRsProv tnVzBrCPName="sql"/>
            <fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter"/>
        </fvAEPg>

        <fvAEPg name="app">
            <fvRsBd tnFvBDName="bd1"/>
            <fvRsProv tnVzBrCPName="rmi"/>
            <fvRsCons tnVzBrCPName="sql"/>
            <fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter"/>
        </fvAEPg>
    </fvAp>

    <vzFilter name="http" >
        <vzEntry dFromPort="80" name="DPort-80" prot="tcp" etherT="ip" />
        <vzEntry dFromPort="443" name="DPort-443" prot="tcp" etherT="ip" />
    </vzFilter>

    <vzFilter name="rmi" >
        <vzEntry dFromPort="1099" name="DPort-1099" prot="tcp" etherT="ip" />
    </vzFilter>

    <vzFilter name="sql">
        <vzEntry dFromPort="1521" name="DPort-1521" prot="tcp" etherT="ip" />
    </vzFilter>

    <vzBrCP name="web">
        <vzSubj name="web">
            <vzRsSubjFiltAtt tnVzFilterName="http"/>
        </vzSubj>
    </vzBrCP>

    <vzBrCP name="rmi">
        <vzSubj name="rmi">
            <vzRsSubjFiltAtt tnVzFilterName="rmi"/>
        </vzSubj>
    </vzBrCP>

    <vzBrCP name="sql">
        <vzSubj name="sql">
           <vzRsSubjFiltAtt tnVzFilterName="sql"/>
        </vzSubj>
    </vzBrCP>
</fvTenant>
```

XML 構造では、最初の行が ExampleCorp という名前のテナントを変更または必要に応じて作成します。

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

次の行は、OnlineStore という名前のアプリケーション ネットワーク プロファイルを作成します。

```
<fvAp name="OnlineStore">
```

アプリケーション ネットワーク プロファイル内の要素は、3 つのエンドポイント グループを作成します（3 台のサーバそれぞれに 1 つずつ）。 次の行は、web という名前のエンドポイント グループを作成し、bd1 という名前の既存のブリッジ ドメインに関連付けます。 このエンドポイント グループは、rmi という名前のバイナリ コントラクトで許可されたトラフィックのコンシューマまたは宛先であり、web という名前のバイナリ コントラクトで許可されたトラフィックのプロバイダーまたは送信元です。 エンドポイント グループは、datacenter という名前の VMM ドメインに関連付けられます。

```
<fvAEPg name="web">
    <fvRsBd tnFvBDName="bd1"/>
    <fvRsCons tnVzBrCPName="rmi"/>
    <fvRsProv tnVzBrCPName="web"/>
    <fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter"/>
</fvAEPg>
```

残りの 2 つのエンドポイント グループは、アプリケーション サーバとデータベース サーバに対し、同様の方法で作成されます。

次の行は、TCP トラフィックのタイプ HTTP（ポート 80）および HTTPS（ポート 443）を指定する http という名前のトラフィック フィルタを定義します。

```
<vzFilter name="http">
    <vzEntry dFromPort="80" name="DPort-80" prot="tcp"  />
    <vzEntry dFromPort="443" name="DPort-443" prot="tcp"  />
</vzFilter>
```

残りの2つのフィルタは、アプリケーションのデータおよびデータベース（sql）のデータに対し、同様の方法で作成されます。

次の行は、http という名前のフィルタを組み込む web という名前のバイナリ コントラクトを作成します。

```
<vzBrCP name="web">
    <vzSubj name="web">
        <vzRsSubjFiltAtt tnVzFilterName="http"/>
    </vzSubj>
</vzBrCP>
```

残りの 2 つのコントラクトは、rmi および sql のデータ プロトコルに対し、同様の方法で作成されます。

最後の行は、構造を閉じます。

```
</fvTenant>
```

第 4 章

# ツールを使用した API の開発/試験

この章の内容は、次のとおりです。



# API インスペクタの使用

## 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 |
|---|---|
| Search | 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。 |

| 名前 | 説明 |
|---|---|
| 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。 |
| 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":[]}
```

# Managed Object Browser 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 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://192.0.20.123/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 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。 この領域では、すべての単純な APIC REST API クエリー操作がサポートされます。

| 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。 |

| 名前 | 説明 |
|---|---|
| [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。<br>• ==（等しい）<br>• !=（等しくない）<br>• <（より小さい）<br>• >（より大きい）<br>• ≤<=（以下）<br>• >=（以上）<br>• 間<br>• ワイルドカード<br>• anybit<br>• 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

[Display XML of last query] リンクでは、Visore で実行された最も最近のクエリーの完全な APIC REST API 変換が表示されます。

### [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） 管理対象オブジェクトの多くは内部でのみ使用され、APIC REST APIのプログラム開発には通常適用できません。

| 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] リンク | 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 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。

# API のテスト

## 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://192.0.20.123/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://192.0.20.123/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, （35 ページ）

# Python による API のテスト

Python 要求モジュールを使用して、Python プログラムから API メッセージを送信できます。

次に、API にログインし、認証トークンを保存し、センサーを読み込む例を示します。

```
import json
import requests

base_url = 'https://192.0.20.123/api/'

# create credentials structure
name_pwd = {'aaaUser': {'attributes': {'name': 'georgewa', 'pwd': 'pa55word'}}}
json_credentials = json.dumps(name_pwd)

# log in to API
login_url = base_url + 'aaaLogin.json'
post_response = requests.post(login_url, data=json_credentials)

# get token from login response structure
auth = json.loads(post_response.text)
login_attributes = auth['imdata'][0]['aaaLogin']['attributes']
auth_token = login_attributes['token']

# create cookie array from token
cookies = {}
cookies['APIC-Cookie'] = auth_token

# read a sensor, incorporating token in request
sensor_url = base_url + 'mo/topology/pod-1/node-1/sys/ch/bslot/board/sensor-3.json'
get_response = requests.get(sensor_url, cookies=cookies)

# display sensor data structure
```

```
print get_response.json()
```

この例では、セッションの認証トークン（Cookie）を管理する必要があります。 Python 要求モジュールには、トークンを自動的に管理する Session() メソッドが含まれます。

Python 要求モジュールの詳細については、http://www.python-requests.org を参照してください。

付 録 A

# API を使用したデバイス パッケージ ファイルのアップロード



# API を使用したデバイス パッケージ ファイルのアップロード

レイヤ 4 ～ レイヤ 7（L4 ～ L7）のサービス デバイスをインストールするには、APIC にデバイス パッケージ ファイルをアップロードする必要があります。 この操作用の API コマンドでは、次の URI の特別な形式を使用します。

{**http**| **https**}**:**//*host*[ *:port* ]**/ppi/node/mo/.**{**json**| **xml**}

URI パスには、「api」の代わりに「ppi」（パッケージ プログラミング インターフェイス）が含まれ、コマンドはメッセージの本文としてデバイス パッケージ ファイルでの POST 操作として送信されます。 デバイス パッケージ ファイルは zip ファイルです。

次の例では、デバイス パッケージ ファイルをアップロードする API 操作を示します。

```
POST https://192.0.20.123/ppi/node/mo/.json
```

L4 ～ L7 のサービス デバイス パッケージのインストールの詳細については、『*Cisco APIC Layer 4 to Layer 7 Services Deployment Guide*』 を参照してください。

API を使用したデバイス パッケージ ファイルのアップロード

付 録 B

# その他の例

この章の内容は、次のとおりです。



## 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 を使用したトップ レベルのシステム要素の取得

次に、存在するシステム デバイスを判断するために APIC を照会する例を示します。

システム要素（APIC、スパイン、およびリーフ）に関する一般情報は、クラス top:System のオブジェクトに含まれています。

次に、API クエリー メッセージの例を示します。

```
GET http://192.0.20.123/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"
        }
      }
    }
  ]
}
```


この応答は、システムが 1 つの APIC（ノード 1）と 1 つのリーフ（ノード 17）で構成されることを示します。

# 例：JSON API を使用したノードに関する情報の取得

次に、システムのノードにアクセスするために APIC を照会する例を示します。

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://192.0.20.123/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

次に、実行中のファームウェア イメージを判断するために APIC を照会する例を示します。

実行中のファームウェアの詳細情報は、APIC ファームウェアのステータス コンテナ クラス firmware:IfcFwStatusCont の子クラス（サブツリー）であるクラスのオブジェクト firmware:IfcRunning に含まれています。 実行中のファームウェア インスタンスが複数存在する場合があるので（APIC インスタンス 1 つにつき 1 個）、コンテナ クラスを照会し、実行中のファームウェア オブジェクトのサブツリーの応答をフィルタリングできます。

次に、API クエリー メッセージの例を示します。

```
GET http://192.0.20.123/api/class/firmwareIfcFwStatusCont.json?
        query-target=subtree
        &
        target-subtree-class=firmwareIfcRunning
```

正常な動作時には次のような応答本文が返されます。

```
{
  "imdata" : [{
      "firmwareIfcRunning" : {
        "attributes" : {
          "instanceId" : "0:0",
          "applId" : "3",
          "childAction" : "",
          "dn" : "ifcfwstatuscont/ifcrunning-3",
          "lcOwn" : "local",
```

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


この応答は、APIC ファームウェア バージョン 1.1 の 3 つの実行中のインスタンスについて記述しています。

# 例：JSONAPI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。

次の例では、「MyLPSelectorProf」という名前でリーフ ポート セレクタ プロファイルを作成します。この例のプロファイルには、リーフ スイッチ 1 上でリーフ ポート 1 を、リーフ スイッチ 1 上でリーフ ポート 3 ～ 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://192.0.20.123/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://192.0.20.123/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://192.0.20.123/api/mo/uni/fabric/leportp-MyLPSelectorProf.json
```

付 録 C

# 通信ポリシーの設定



## 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** [Submit] をクリックします。`

# 索 引

## ろ