CHAPTER 2

API の例

この章では、次のメソッドについて例を挙げて説明します。

認証

認証とは、XML ドキュメント（API）が UCS 環境にアクセスするために実行する手段です。さらに認証では、特定の API 開発者が UCS 環境に対して実行できることと実行できないことを決定します。つまり、認証とは、誰が何を実行できるかを制御するアクセス権限を設定する手段です。

Hello World

ほとんどの開発環境と同様に、最初の例は一般的な「Hello World」アプリケーションです。これは API を使用する開発者と、この場合は UCS との間の最も小規模なやり取りを表すプログラムです。「Hello World」について次に示します。

bash> telnet < *ファブリック インターコネクトの IP アドレス* > 80

POST /nuova HTTP/1.1

User-Agent: lwp-request/2.06

Host: < *ホストの IP アドレス* >

Content-Length: 132

Content-Type: application/x-www-form-urlencoded

<!-- 簡単なクラス指定によるクエリーの例 -->

<configResolveClass

cookie=" 1206347805/4fa58144-c7f0-4990-b9f3-dc71a164aed7 " classId="equipmentChassis"/>

ログインの例

ここでは、UCS にログインするプロセスを見ていきます。

UCS にログインするには、最初に TCP 接続を確立し、次に「aaaLogin」メソッドを呼び出して、ユーザ名とパスワードを指定します。aaaLogin のようなメソッドは、セキュアなサブネット上のセキュアなポート（443）でポストすることを推奨します。

クライアントのポスト：

```
<aaaLogin inName="bob" inPassword="abc123" />
```

応答：

```
<aaaLogin response="yes" outCookie="1217377205/85f7ff49-e4ec-42fc-9437-da77a1a2c4bf"
outRefreshPeriod="600"
outPriv="aaa,ext-lan-policy,ext-lan-qos,ext-san-policy,operations,pod-policy,pod-qos,read-only
" outDomains="mgmt02-dummy" outChannel="noencssl" outEvtChannel="noencssl">
</aaaLogin>
```

クライアントのポスト メッセージは次のとおりです。

クライアントのポスト：

```
1. <aaaLogin
2.         inName="bob"
3.         inPassword="abc123"
4. />
```

API は明確に書式化された XML ドキュメントです。この例では、ルート エレメントの「aaaLogin」といくつかの属性だけが含まれており、子エレメントはありません。

1 行目では、XML エレメントが「aaaLogin」であり、これは UCS にログインするために使用する API メソッドです。2 行目では、「inName」が属性名であり、属性値は「bob」になっています。3 行目では「inPassword」が属性名であり、属性値は「abc123」です。2 行目と 3 行目では、属性はメソッドに対するパラメータであり、XML エレメントによって表され、cookie を評価します。4 行目はルート エレメント「aaaLogin」を閉じて XML ドキュメントを完了させるタグです。

このように、上記の XML ドキュメントはコンテンツを含まない空のエレメントです。ほとんどの場合、エレメントはコンテンツを含みます。また、XML のバージョンと DOCTYPES が指定されておらず、使用する必要がないことに注意してください。

各 XML ドキュメントは UCS で実行する操作を表します。UCS が XML API ドキュメントを受信すると、要求を読み込み、そのメソッドと含まれている属性に基づいて処理を実行します。この場合は、UCS が inName と inPassword の属性値を使用してログイン要求を処理します。

XML ドキュメントを通じて UCS に対して API 呼び出しが行われると、UCS は XML ドキュメントの形式で応答メッセージを返します。この応答では、要求の処理の成否が示されています。UCS の応答メッセージは次のとおりです。

応答：

```
1. <aaaLogin
2.         response="yes"
```

```
3.      outCookie="1217377205/85f7ff49-e4ec-42fc-9437-da77a1a2c4bf"
4.      outRefreshPeriod="600"
5.      outPriv="aaa,ext-lan-policy,ext-lan-qos,ext-san-policy,operations,pod-policy,pod-qos,read-only"
6.      outDomains="mgmt4711 "
7.      outChannel="noencssl"
8.      outEvtChannel="noencssl">
9.  /aaaLogin>
```

ログインに失敗した場合には、次の応答が返されます。

応答：

```
1.  <aaaLogin
2.      cookie=""
3.      response="yes"
4.      errorCode="551"
5.      invocationResult="unidentified-fail"
6.      errorDescr="Authentication failed">
7.  </aaaLogin>
```

API の応答は完全な XML ドキュメントです。この例では、ルート エレメントの「aaaLogin」と複数の属性だけが含まれています。1 行目のエレメントである「aaaLogin」は UCS にログインするために使用する API メソッドです。UCS は応答メッセージにメソッドからの結果を添えて送信します。

2 行目では、これが API 要求（1 行目の aaaLogin）の応答であることを示しています。

3 行目の属性「outCookie」とその値「1217377205/85f7ff49-e4ec-42fc-9437-da77a1a2c4bf」は、セッションの cookie の属性です。

4 行目の属性「outRefreshPeriod」とその値「600」は、推奨される cookie のリフレッシュ間隔（600 秒間 = 10 分間）です。実際の猶予期間はデフォルトで 2 時間です。

5 行目の属性「outPriv」の値には、ユーザ アカウントに割り当てされている権限が含まれています。6 行目の属性「outDomains」とその値は「mgmt4711-dummy」は、UCS の一意の名前です。7 行目の属性「outChannel」とその値「noencssl」は、このセッションで ssl での暗号化を使用していないことを示しています。

8 行目の属性「outEvtChannel」とその値は「noencssl」は、いずれの「イベント サブスクリプション」でも ssl での暗号化を使用しないことを示しています。

最後に、9 行目はルート エレメント「aaaLogin」を閉じるタグです。このタグは XML ドキュメントを完了させるものであり、API メソッドの末尾でもあります。

（注） イベント サブスクリプションについては、「イベント サブスクリプション メソッド」（P.1-16）を参照してください。

接続とログインの例

- TCP 80 または 443 でセッションを開きます。
- 送信：

```
POST http://myserver.cisco.com:80/nuova
Content-Type: application/x-www-form-urlencoded
<aaaLogin
        inName="bob"
        inPassword="abc123"
/>
```

- 応答：

```
<aaaLogin
    response="yes"
    outCookie="1217377205/85f7ff49-e4ec-42fc-9437-da77a1a2c4bf"
    outRefreshPeriod="600"
    outPriv="aaa,ext-lan-policy,ext-lan-qos,ext-san-policy,operations,pod-policy,pod-qos,read-only"
    outDomains="sys-machine"
    outChannel="noencssl"
    outEvtChannel="noencssl">
</aaaLogin>
```

その他の応答の例

クエリーの対象となったオブジェクトが存在していない場合や、要求に何らかの問題があって処理に失敗した場合の応答の例は次のとおりです。

- オブジェクトが存在しない

```
<configResolveDn
        dn="sys-machine/chassis-1/blade-4711"
        cookie="<real cookie>"
        response="yes">
        <outConfig> </outConfig>
</configResolveDn>
```

- 要求の処理に失敗

```
<configConfMo
        dn="fabric/server"
        cookie="<real cookie>"
        response="yes"
```

errorCode="103"

invocationResult="unidentified-fail"

errorDescr="can't create; object already exists.">

</**configConfMo**

クエリー

このセクションでは、クエリー メソッドを使用するいくつかの例について説明します。クエリーを使用すると、UCS から情報（階層、状態、スコープ、解決オブジェクトなど）を入手することができます。

認定者名の解決

dn を解決するときに考慮する必要のある項目は次のとおりです。

- インスタンス メソッド（常に dn で指定）
- 結果を階層にできる
- 第 1 章「UCS の XML API」のメソッドについての説明を参照
- 列挙された値、classId、ビットマスクが「tringified」

configResolveDn

```
<configResolveDn inHierarchical="true"
      cookie="1206347805/4fa58144-c7f0-4990-b9f3-dc71a164aed7" dn="sys">
</configResolveDn>
```

子の解決

次の例では、次のパラメータを使用して処理を実行します。

- ネームド クラス（この場合は「adaptorHostEthIf」）のインスタンスであるネームド オブジェクトのすべての子を取得する
- クラス メソッド：classId を指定する
- 結果セットに適用されるフィルタ（inHierarchial の「true」または「false」）を受け入れる

configResolveChildren

次のクエリーの例では、「classId」が「adaptor/Host/EthIf」のすべてのオブジェクトにクエリーを実行し、そのクラス内の子を返します。

```
<configResolveChildren classId="adaptorHostEthIf"
      cookie="1214933755/7bc19b1f-332e-40b5-9520-d75f72ed3da9"
      inDn="sys/chassis-1/blade-2/adaptor-1" inHierarchical="false">
</configResolveChildren>
```

クエリーのスコープ

このクエリー メソッドの例では、ネームド クラス（computeBlade）のインスタンスであるネームド オブジェクトのすべての子を API によって「取得」します。次の項目に注意してください。

- インスタンス メソッド : dn を指定
- フィルタ（inHierarchical）を受け入れて結果セットに適用

configScope

次の例では、UCS に対してクエリーを実行し、ログインのスコープを判断します。

```
<configScope
        cookie="1214933755/7bc19b1f-332e-40b5-9520-d75f72ed3da9"
        inClass="computeBlade" inHierarchical="false"
        dn="sys"
</configScope>
```

configResolveClass

UCS 内の機器のすべてのインスタンスを入手するには、equipmentItem クラスに対してクエリーを実行します。

```
<configResolveClass cookie="1239661687/6d2eaff9-6ab6-4a46-bc47-324f5b2a58b2"
inHierarchical="false" classId="equipmentItem"/>
```

上記の例では、抽象クラス「equipmentItem」のすべてのインスタンスに対してクエリーを実行しています。したがって、結果セットは非常に大量になります。結果セットはできるだけ正確に定義する必要があります。

たとえば、サーバのリストだけを入手したいときには、上記のクエリーで「classId」の属性値として「computeBlade」を使用できます。

MAC プールに対するクエリー

MIT 内のすべての MAC アドレスのリストを入手するには、「macpoolAddr」に対してクエリーを実行します。これらは個々の（システムが作成した）「macpoolUniverse」の子です。

要求をポストする例は次のとおりです。

送信：

```
<configScope cookie="1239665984/58f22805-f4cc-41ba-9747-0892147da9f0" inHierarchical="false"
dn="mac" inClass="macpoolAddr"/>
```

応答：

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

```
<configScope
    cookie="1239665984/58f22805-f4cc-41ba-9747-0892147da9f0"
    dn="mac"
    response="yes">
<outConfigs>
    <macpoolAddr
        assigned="no"
        ssignedToDn=""
        dn="mac/00:00:00:00:FF:0F"
        id="00:00:00:00:FF:0F"
        owner="pool">
    </macpoolAddr>
    <macpoolAddr
        assigned="no"
        assignedToDn=""
        dn="mac/00:00:00:00:FF:0E"
        id="00:00:00:00:FF:0E"
        owner="pool">
    </macpoolAddr>
    <macpoolAddr
        assigned="no"
        assignedToDn=""
        dn="mac/00:00:00:00:FF:0D"
        id="00:00:00:00:FF:0D"
        owner="pool">
    </macpoolAddr>
    <macpoolAddr
        assigned="no"
        assignedToDn=""
        dn="mac/00:00:00:00:FF:0C"
        id="00:00:00:00:FF:0C"
        owner="pool">
    </macpoolAddr>
    <macpoolAddr
        assigned="no"
        assignedToDn=""
        dn="mac/00:00:00:00:FF:0B"
        id="00:00:00:00:FF:0B"
```

```
        owner="pool">
    </macpoolAddr>
    <macpoolAddr
        assigned="no"
```

さらに数ページ続きます。

「macpoolAddr」クラスのオブジェクトが MAC プール ユニバースの子としてだけ存在するため、次のようにさらに簡潔なクエリーも実行できます。

```
<configResolveClass cookie="1239661687/6d2eaff9-6ab6-4a46-bc47-324f5b2a58b2"
inHierarchical="false" classId="macpoolAddr"/>
```

MAC アドレスが割り当てられている computeBlade（サーバ）を特定するには、「assignedToDn」フィールドを確認します。たとえば、10:00:00:00:00:03 という MAC アドレスについて調べたい場合、そのアドレスが割り当てられている機器は次の行に表示されています。

送信：

```
<configResolveDn cookie="1239661687/6d2eaff9-6ab6-4a46-bc47-324f5b2a58b2"
inHierarchical="false" dn="mac/10:00:00:00:00:03"/>
```

上記のクエリーの結果として、そのオブジェクトが返されます。その応答は次のとおりです。

応答：

```
<configResolveDn dn="mac/10:00:00:00:00:03" cookie="1239666709/fc0faf16-6211-4e12-afff-9e96b75f4e96"
response="yes">
    <outConfig>
        <macpoolAddr assigned="yes" assignedToDn="org-root/ls-BOB/ether-eth1"
        dn="mac/10:00:00:00:00:03" id="10:00:00:00:00:03" owner="pool" />
    </outConfig>
</configResolveDn>
```

この結果から、この MAC アドレスはサーバ プロファイル「ls-Bob」のイーサネット インターフェイス（子オブジェクト）に割り当てられていることがわかります。「lsServer」が判明すれば、そのブレードの情報を確認できます。

統計情報に対するクエリー

統計情報はさまざまオブジェクトについて多数存在するため、一度にすべての統計情報を得るクエリーを実行することは推奨しません。報告される統計情報とオブジェクトの種類を明確にして実行してください。

次に例を示します。

sys/chassis-1/blade-1/board/cpu-2 の compCpuStats オブジェクトを入手するには、次に示すように

sys/chassis-1/blade-1/board/cpu-2 のすべての子に対してクエリーを実行します。

このクエリーをより正確に実行するには、configScope メソッドを使用します。

送信：

```
<configScope cookie="1239664817/85d9637b-d22a-4faa-95b7-0133bafb5ce9" inHierarchical="false"
dn="sys/chassis-1/blade-1/board-1/cpu-2" inClass="faultInst"/>
```

応答：

```
<?xml version="1.0" encoding="UTF-8"?>
<configScope
    cookie="1239664817/85d9637b-d22a-4faa-95b7-0133bafb5ce9"
    dn="sys/chassis-1/blade-1/board-1/cpu-2"
    response="yes">
  <outConfigs>
  </outConfigs>
</configScope>
```

次の呼び出しでは、「dn」を使用してクエリーを実行してこの統計値を入手します。configScope よりも負荷の低い方法です。

```
<configResolveDn inHierarchical="false"
cookie="1239665195/33253282-6cf0-449b-ad0a-cc630dfa2804"
dn="sys/chassis-1/blade-1/board/cpu-2/stats"></configResolveDn>
```

統計情報オブジェクトの「dn」がわかっていれば、階層クエリーを実行して階層情報（統計情報オブジェクトの子）を入手するクエリーを実行できます。統計情報オブジェクトと階層情報を入手したい場合は、「hierarchical」パラメータを「true」に変更できます。

```
<configResolveDn inHierarchical="true"
cookie="1239665195/33253282-6cf0-449b-ad0a-cc630dfa2804"
dn="sys/chassis-1/blade-1/board/cpu-2/stats"></configResolveDn>
```

障害に対するクエリー

ファブリック インターコネクトのすべての障害についての一覧を入手するには、次のように実行します。

```
<configResolveClass cookie="1239661687/6d2eaff9-6ab6-4a46-bc47-324f5b2a58b2"
inHierarchical="false" classId="faultInst"/>
```

スイッチ上で発生した重大なすべての障害についての一覧を入手するには、次の例を使用します。ここではフィルタを使用しています（<inFilter>eq class= 「faultInst」）。

```
<configResolveClass cookie="1239661687/6d2eaff9-6ab6-4a46-bc47-324f5b2a58b2"
inHierarchical="false" classId="faultInst">
    <inFilter>
        <eq class="faultInst" property="highestSeverity" value="major" />
    </inFilter>
</configResolveClass>
```

フィルタを使用するクエリー

以降のセクションでは、フィルタを使用するクエリー要求の使用方法について説明します。フィルタは、より具体的で対象を絞ったクエリーを作成するために使用します。フィルタの種類全体の一覧については、「クエリー フィルタ」（P.1-12）を参照してください。

Equality フィルタ

次の例は、関連するすべてのサーバを抽出するクエリーを表しています。

```
<configResolveClass cookie="1239479253/25c923a6-f7db-4478-bc8d-786cfedd02f5"
inHierarchical="false" classId="lsServer">
    <inFilter>
        <eq class="lsServer" property="assocState" value="associated" />
    </inFilter>
</configResolveClass>
```

上記の例で、<inFilter><eq class="lsServer" property="assocState" value="associated" の行に注意してください。

ここでは、「Equal to」フィルタによって、関連付けられているサーバが抽出されます。

Inequality フィルタ

次の例では、Inequality フィルタを使用して、関連付けられていないサーバをすべて抽出します。

```
<configResolveClass cookie="1239479253/25c923a6-f7db-4478-bc8d-786cfedd02f5"
inHierarchical="false" classId="lsServer">
    <inFilter>
        <ne class="lsServer" property="assocState" value="associated" />
    </inFilter>
</configResolveClass>
```

Equality フィルタと同様に、この例の「<inFilter><ne class=」の要素で Inequality フィルタが囲まれています。

Wildcard フィルタ

次の例では、「Wildcard」フィルタを使用して、シリアル番号が「QCI1」というプレフィクスで始まるアダプタ ユニットすべてを抽出します。

```
<configResolveClass cookie="1241527652/528daa5b-7cbf-47f2-a3c4-3bf74473f1c4"
inHierarchical="false" classId="adaptorUnit">
    <inFilter>
      <wcard class="adaptorUnit" property="serial" value="QCI1*" />
    </inFilter>
</configResolveClass>
```

Greater than or Equal to フィルタ

次の例では、「Greater than or Equal to」フィルタを使用して、2048MB 以上の容量があるメモリ アレイを抽出します。

```
<configResolveClass cookie="1241527652/528daa5b-7cbf-47f2-a3c4-3bf74473f1c4"
    inHierarchical="false" classId="memoryArray">
    <inFilter>
      <ge class="memoryArray" property="currCapacity" value="2048" />
    </inFilter>
</configResolveClass>
```

次の例では、「Less than or Equal to」フィルタを使用して、2048 MB 以下の容量のメモリ アレイを抽出します。

```
<configResolveClass cookie="1241527652/528daa5b-7cbf-47f2-a3c4-3bf74473f1c4"
inHierarchical="false" classId="memoryArray">
    <inFilter>
      <le class="memoryArray" property="currCapacity" value="2048" />
    </inFilter>
</configResolveClass>
```

Not Equal to フィルタ

次の例は「Not Equal to」フィルタの例であり、割り当てられていない（割り当て状態のプロパティが「assigned」でない）サーバをすべて抽出します。

```
<configResolveClass cookie="1241527652/528daa5b-7cbf-47f2-a3c4-3bf74473f1c4"
inHierarchical="false" classId="lsServer">
```

```
    <inFilter>
      <ne class="lsServer" property="assignState" value="assigned" />
    </inFilter>
</configResolveClass>
```

Greater than フィルタ

次の「Greater than」フィルタの例では、1024MB より大きな容量を持つメモリ アレイを抽出します。

```
<configResolveClass cookie="1241527652/528daa5b-7cbf-47f2-a3c4-3bf74473f1c4"
inHierarchical="false" classId="memoryArray">
    <inFilter>
      <gt class="memoryArray" property="currCapacity" value="1024" />
    </inFilter>
</configResolveClass>
```

Less than フィルタ

「Less than」フィルタの例では、1024MB より小さな容量を持つメモリ アレイを抽出します。

```
<configResolveClass cookie="1241527652/528daa5b-7cbf-47f2-a3c4-3bf74473f1c4"
inHierarchical="false" classId="memoryArray">
    <inFilter>
      <lt class="memoryArray" property="currCapacity" value="1024" />
    </inFilter>
</configResolveClass>
```

AND フィルタ

プールに割り当てられ、保持されているすべての「uuid」を確認するには、AND フィルタを使用した次のクエリーを実行できます。

送信：

```
<configResolveClass cookie="1239663439/c2283762-f189-406e-b36a-0ec56d71fb9b"
inHierarchical="false" classId="uuidpoolAddr">
    <inFilter>
      <and>
        <eq class="uuidpoolAddr" property="owner" value="pool" />
        <eq class="uuidpoolAddr" property="assigned" value="yes" />
      </and>
    </inFilter>
</configResolveClass>
```

応答：

```
<?xml version="1.0" encoding="UTF-8"?>
<configResolveClass
        classId="uuidpoolAddr"
        cookie="1239663464/26b69794-5d6d-433a-8d29-955f9a2100c9"
        response="yes">
    <outConfigs>
        <uuidpoolAddr
            assigned="yes"
            assignedToDn="org-root/ls-foo"
            dn="uuid/F000-00000000000F"
            id="F000-00000000000F"
            owner="pool">
        </uuidpoolAddr>
    </outConfigs>
</configResolveClass>
```

次の例では、AND フィルタを使用して、ベンダーが「Cisco Systems Inc」であり、かつ、シリアル番号が「CHS A04」であるシャーシを抽出します。

```
<configResolveClass cookie="1239838553/c1a4292e-5235-4907-b910-7d0dd25fd7ee"
inHierarchical="false" classId="equipmentChassis">
    <inFilter>
      <and>
        <eq property="vendor" value="Cisco Systems Inc" class="equipmentChassis"/>
        <eq class="equipmentChassis" property="serial" value="CHS A04"/>
      </and>
    </inFilter>
</configResolveClass>
```

NOT フィルタ

「NOT」フィルタは、含まれているフィルタの結果を逆にします。次の例では、connStatus が A で「ない」サーバを検索します （connStatus プロパティはビットマスクです）。

```
<configResolveClass cookie="null" inHierarchical="false" classId="computeBlade">
    <inFilter>
      <not>
        <anybit class="computeBlade" property="connStatus" value="A" />
      </not>
```

```
    </inFilter>
</configResolveClass>
```

Any Bits フィルタ

次の「Any Bits」フィルタの例では、connStatus が A か B のサーバをすべて抽出します （connStatus プロパティはビットマスクです）。

```
<configResolveClass cookie="null" inHierarchical="false" classId="computeBlade">
    <inFilter>
        <anybit class="computeBlade" property="connStatus" value="A,B" />
    </inFilter>
</configResolveClass>
```

All Bits フィルタ

次の「All Bits」フィルタの例では、configQualifier ビットマスクに vnic-capacity および vhba-capacity の両方が設定されている論理サーバをすべて抽出します。

```
<configResolveClass cookie="1241448379/5c51a994-2c48-47e9-8c53-a7b75964fd33"
inHierarchical="false" classId="lsServer">
    <inFilter>
        <allbits class="lsServer" property="configQualifier" value="vnic-capacity,vhba-capacity" />
    </inFilter>
</configResolveClass>
```

Between フィルタ

次の「Between」フィルタの例では、1023MB から 2048MB までの容量を持つメモリ アレイを抽出します。

```
<configResolveClass cookie="1239838553/c1a4292e-5235-4907-b910-7d0dd25fd7ee"
inHierarchical="false" classId="computeBlade">
    <inFilter>
        <bw class="memoryArray" property="populated" firstValue="1" secondValue="5"/>
    </inFilter>
</configResolveClass>
```

設定

このセクションでは、設定メソッドを使用した例を説明します。多くの操作は、既存の管理対象オブジェクトの管理プロパティの値を変更することに関連しています。このような変更や設定の処理は、設定メソッドを使用して行います。設定メソッドに関連する項目のいくつかを、次に一覧します。

- 障害ポリシーはシステムが作成したオブジェクトです（dn は 障害/障害ポリシー）。
- 保持ポリシー、および障害ポリシーの保持間隔を設定したい場合もあります。
- ドメインについての知識は必須：管理対象オブジェクトの中には内部だけで使用できるものがあり、イメージのアップグレードには複数のオブジェクトの変更が必要です。これについては後述します。

設定の変更

UCS 内の個々のプロパティの管理には、configConfMo メソッドを使用します。このメソッドを使用して、プロセスの開始や停止、個々のプロパティの状態の変更、UCS でのオブジェクトの作成などを行えます。

（注）　1 つの管理対象オブジェクトに変更を加えることにより、UCS では複数の変更が生じる場合があることに注意してください。このとき、これらの変更は、先の 1 つの設定変更からは目に見えない形で生じることがあります。

ConfigConfMo

次の例では、「fault-policy」のクリア間隔を 20 分にリセットします。

```
<configConfMo
   cookie="1214933755/7bc19b1f-332e-40b5-9520-d75f72ed3da9"
   inHierarchical="false">
   <inConfig>
      <faultPolicy
         dn="fault/fault-policy" clearInterval="00:20:00">
   </inConfig>
</configConfMo>
```

configConfMos

次の例では、サーバをシャットダウンする簡単な設定操作を表しています（同じタスクは configConfMo でも実行できます）。この操作は、属性の状態を設定するだけです。

```
<configConfMos cookie="2/10/2/24"  inHierarchical="no">
    <inConfigs>
        <pair key="org-root/ls-testLS1/power">
            <lsPower dn="org-root/ls-testLS1/power"
                state="hard-reset-immediate"
```

```
                status="modified"
            />
        </pair>
    </inConfigs>
</configConfMos>
```

configConfMos による削除

削除は設定操作です。次の例では、ターゲット オブジェクトを特定し、その状態を「deleted」に変更します。削除（delete）の処理は階層的（ツリー構造）であることに注意してください。また、システムによって作成されたオブジェクトは削除できません。

```
<configConfMos
    cookie="1221768831/29a075af-2c61-4776-9b85-079c8107d6b8"inHierarchical="false">
    <inConfigs>
        <pair key="org-root/ls-abc">
                <lsServer dn="org-root/ls-abc" status="deleted"/>
        </pair>
    </inConfigs>
</configConfMos>
```

UUID プールの作成

次の例では、UUID プール（サーバ プール）を作成し、そのプールにアドレスのブロックを割り当てます。このブロックの dn は次のとおりです。

org-root/uuid-pool-corpUuidPool/block-from-0000-000000000F72-to-0000-000000000F74

また、暗黙のオブジェクトとして、3 つの uuidpoolPooled オブジェクトが作成されます。

org-root/uuid-pool-corpUuidPool/0000-000000000F74

```
<configConfMo cookie="1221768831/29a075af-2c61-4776-9b85-079c8107d6b8"inHierarchical="no">
    <inConfig>
        <uuidpoolPool dn="org-root/uuid-pool-corpUuidPool"
            descr="Corporate uuid pool."
            name="corpUuidPool">
            <uuidpoolBlock
                from="0000-000000000F74"
                to="0000-000000000F78">
            </uuidpoolBlock>
        </uuidpoolPool>
    </inConfig>
</configConfMo>
```

MAC プール

次の例では、1 つのトランザクションで 2 つの MAC プールを作成します。いずれかのプールに対して範囲が無効なときには、サーバは全体の設定をロールバックすることに注意してください。これは 1 つのトランザクションの中で行われ、全体の操作は成功か失敗のいずれかになります。

```
configConfMo inHierarchical="true" cookie="<real cookie>" dn="org-root">
    <inConfig>
        <orgOrg name="root">
            <macpoolPool name="test">
              <macpoolBlock to="00:00:00:00:10:16" from="00:00:00:00:10:10"></macpoolBlock>
            </macpoolPool>
            <macpoolPool name="reserved">
              <macpoolBlock to="00:00:00:00:FF:10" from="00:00:00:00:FF:00"></macpoolBlock>
            </macpoolPool>
        </orgOrg>
    </inConfig>
</configConfMo>
```

論理サーバの作成

次の例では、サーバのプロファイルを作成します。これは最終的にはシャーシ内の論理サーバを定義するために使用されます。

（注） これはサーバ プロファイルの例としては比較的単純なものです。実際のプロファイルには、これより多数の属性やオブジェクトを含めることができます（必要があります）。

```
<configConfMos cookie="1221768831/29a075af-2c61-4776-9b85-079c8107d6b8"
inHierarchical="yes">
    <inConfigs>
      <pair key="org-root/ls-testLS1">
          <lsServer dn="org-root/ls-testLS1" name="testLS1"
              uuid="12345678-1234-1234-1234-123456789012">
              <vnicEther addr="01:02:03:04:05:06" name="vnicfinanceA" switchId="A">
                  <vnicEtherIf defaultNet="yes" name="finance"/>
              </vnicEther>
              <lsbootDef>
                      <lsbootStorage order="1">
                          <lsbootSanImage type="primary" vnicName="vhbaA">
                              <lsbootSanImagePath lun="1" type="primary"
wwn="20:00:00:00:00:00:00:10"/>
```

```
                    </lsbootSanImage>
                </lsbootStorage>
                <lsbootVirtualMedia order="2" access="read-only"/>)
            <lsbootLan order="3">
                <lsbootLanImagePath type="primary" vnicName="finance"/>
            </lsbootLan>
        </lsbootDef>
        <vnicFc addr="50:02:03:04:05:06:07:8D" name="fc1">
            <vnicFcIf name="primary"/>
        </vnicFc>
        <lsBinding pnDn="sys-machine/chassis-1/blade-1" status="created"/>
    </lsServer>
  </pair>
 </inConfigs>
</configConfMos>
```

（注）　紫色の下線を引いた部分は、現在説明の重点を置いている XML ドキュメントの部分です。

XML ドキュメントのこの部分では、サーバの名前が定義されています。この例の cookie は、コンパクトなプレースホルダとして使用されているだけの単純な数値です。実際の cookie は、複数桁の実数値です。

```
<configConfMos cookie="1221768831/29a075af-2c61-4776-9b85-079c8107d6b8"
inHierarchical="yes">
    <inConfigs>
      <pair key="org-root/ls-testLS1">
        <lsServer dn="org-root/ls-testLS1" name="testLS1"
          uuid="12345678-1234-1234-1234-123456789012">
            <vnicEther addr="01:02:03:04:05:06" name="vnicfinanceA" switchId="A">
                <vnicEtherIf defaultNet="yes" name="finance"/>
            </vnicEther>
            <lsbootDef>
                <lsbootStorage order="1">
                    <lsbootSanImage type="primary" vnicName="vhbaA">
                        <lsbootSanImagePath lun="1" type="primary"
wwn="20:00:00:00:00:00:00:10"/>
                    </lsbootSanImage>
                </lsbootStorage>
```

```
                    <lsbootVirtualMedia order="2" access="read-only"/>)
                    <lsbootLan order="3">
                        <lsbootLanImagePath type="primary" vnicName="finance"/>
                    </lsbootLan>
                </lsbootDef>
                <vnicFc addr="50:02:03:04:05:06:07:8D" name="fc1">
                    <vnicFcIf name="primary"/>
                </vnicFc>
                <lsBinding pnDn="sys-machine/chassis-1/blade-1" status="created"/>
            </lsServer>
        </pair>
    </inConfigs>
</configConfMos>
```

XML ドキュメントのこの部分では、サーバの virtual network card（VNIC; 仮想ネットワーク カード）を定義します。強調表示した部分では、VNIC の名前（vnicfinanceA）と MAC アドレス（01:02:03:04:05:06）を定義し、またこの VNIC に関連付けられている VLAN（finance）を定義しています。

```
<configConfMos cookie="1221768831/29a075af-2c61-4776-9b85-079c8107d6b8"
inHierarchical="yes">
    <inConfigs>
        <pair key="org-root/ls-testLS1">
            <lsServer dn="org-root/ls-testLS1" name="testLS1"
                uuid="12345678-1234-1234-1234-123456789012">
                <vnicEther addr="01:02:03:04:05:06" name="vnicfinanceA" switchId="A">
                    <vnicEtherIf defaultNet="yes" name="finance"/>
                </vnicEther>
                    <lsbootStorage order="1">
                        <lsbootSanImage type="primary" vnicName="vhbaA">
                            <lsbootSanImagePath lun="1" type="primary"
wwn="20:00:00:00:00:00:00:10"/>
                        </lsbootSanImage>
                    </lsbootStorage>
                    <lsbootVirtualMedia order="2" access="read-only"/>)
                    <lsbootLan order="3">
                        <lsbootLanImagePath type="primary" vnicName="finance"/>
                    </lsbootLan>
                </lsbootDef>
```

```
            <vnicFc addr="50:02:03:04:05:06:07:8D" name="fc1">
                <vnicFcIf name="primary"/>
            </vnicFc>
            <lsBinding pnDn="sys-machine/chassis-1/blade-1" status="created"/>
        </lsServer>
    </pair>
  </inConfigs>
</configConfMos>
```

強調表示されている部分では、論理サーバのブート定義を定義しています。ブート定義が受信されるインターフェイスは、「fc1」として、ワールド ワイド ポート名（wwn="20:00:00:00:00:00:00:10"）と一緒に定義されています。仮想メディア（2）が、ブート LAN（3）と一緒に定義されています。

```
<configConfMos cookie="1221768831/29a075af-2c61-4776-9b85-079c8107d6b8"
inHierarchical="yes">
  <inConfigs>
    <pair key="org-root/ls-testLS1">
      <lsServer dn="org-root/ls-testLS1" name="testLS1"
        uuid="12345678-1234-1234-1234-123456789012">
          <vnicEther addr="01:02:03:04:05:06" name="vnicfinanceA" switchId="A">
            <vnicEtherIf defaultNet="yes" name="finance"/>
          </vnicEther>
          <lsbootDef>
            <lsbootStorage order="1">
              <lsbootSanImage type="primary" vnicName="vhbaA">
                <lsbootSanImagePath lun="1" type="primary"
wwn="20:00:00:00:00:00:00:10"/>
              </lsbootSanImage>
            </lsbootStorage>
            <lsbootVirtualMedia order="2" access="read-only"/>)
            <lsbootLan order="3">
              <lsbootLanImagePath type="primary" vnicName="finance"/>
            </lsbootLan>
          </lsbootDef>
          <vnicFc addr="50:02:03:04:05:06:07:8D" name="fc1">
            <vnicFcIf name="primary"/>
          </vnicFc>
          <lsBinding pnDn="sys-machine/chassis-1/blade-1" status="created"/>
      </lsServer>
```

```
        </pair>
    </inConfigs>
</configConfMos>
```

ここで、強調表示されている部分は、VHBA インターフェイスを定義して、これをインターフェイスにバインドしている XML の部分です。プロセス ノード（pnDN）が、特定のシャーシの特定のブレードに相当するよう設定していることに注意してください（**<lsBinding pnDN="sys/chassis-1/blade-1" status="created"/>**）。

```
<configConfMos cookie="1221768831/29a075af-2c61-4776-9b85-079c8107d6b8"
inHierarchical="yes">
    <inConfigs>
      <pair key="org-root/ls-testLS1">
        <lsServer dn="org-root/ls-testLS1" name="testLS1"
          uuid="12345678-1234-1234-1234-123456789012">
            <vnicEther addr="01:02:03:04:05:06" name="vnicfinanceA" switchId="A">
              <vnicEtherIf defaultNet="yes" name="finance"/>
            </vnicEther>
            <lsbootDef>
              <lsbootStorage order="1">
                <lsbootSanImage type="primary" vnicName="vhbaA">
                  <lsbootSanImagePath lun="1" type="primary"
wwn="20:00:00:00:00:00:00:10"/>
                  </lsbootSanImage>
              </lsbootStorage>
                <lsbootVirtualMedia order="2" access="read-only"/>)
                <lsbootLan order="3">
                  <lsbootLanImagePath type="primary" vnicName="finance"/>
              </lsbootLan>
            </lsbootDef>
            <vnicFc addr="50:02:03:04:05:06:07:8D" name="fc1">
                <vnicFcIf name="primary"/>
            </vnicFc>
            <lsBinding pnDn="sys-machine/chassis-1/blade-1" status="created"/>
        </lsServer>
      </pair>
    </inConfigs>
</configConfMos>
```

イベント サブスクリプション

イベント チャネルをサブスクライブするには、次の行を使用します。

```
<eventSubscribe cookie="1239665195/33253282-6cf0-449b-ad0a-cc630dfa2804">
    <filter>
       <true/>
    </filter>
</eventSubscribe>
```

イベント チャネルを使用すると、オブジェクトの変更を、オブジェクトのローカル コピーに適用できます。イベントを見逃したり、ソケットが接続解除されたりしたときには、サーバに対してクエリーを実行して、イベントの正しい順序を調べる必要があります。これを行うには、「eventRecord」のインスタンスに対してクエリーを実行します。

（注） 現在、イベント チャネルに対するフィルタリングはサポートされていません。イベント チャネルのサブスクリプションにより、すべてのイベントが返されるようになります。