CHAPTER 1

UCS の XML API

この章では、XML の概要について簡単に説明し、その後、UCS の XML ベースの API の構造について説明します。まず、API のモデルと主なコマンド構造について説明し、さらにすべての API メソッドについても説明します。また、この章では、API 内でのフィルタの使用方法や、エラーが発生したときの API の対応についても説明します。

XML について

Extensible Markup Language（XML）とは、カスタム マークアップ言語を作成するための一般的な用途の仕様です。ユーザが独自の要素を定義できるため、拡張言語として分類されています。XML の第一の目的は、情報システム間で構造化されたデータを一部インターネットを介して共有しやすくすることであり、ドキュメントのエンコードと、データのシリアル化の両方に使用されています。

XML スキーマ

スキーマとは、XML 環境でデータを記述し、検証する方法です。スキーマは情報の構造を説明するためのモデルになります。UCS 用の XML スキーマは、API によって利用および配布できるようになっています。

XML API のマニュアル

このマニュアルに掲載されている例の中には一部が省略されているものがあります。UCS の XML API の全体は、API のパッケージに含まれている、『Model Documentation』（HTML）に記載されています。このマニュアルをお読みになる際には、この API の『Model Documentation』の詳細な記述も併せて参照することを推奨します。

UCS の XML API

XML API は、XML が UCS 内での通信のネイティブな形式であるため、Unified Computing System（UCS）との統合や相互にやり取りするのための強力な手段です。たとえば、CLI と GUI の両方で、UCS Manager との通信に同一の XML API が使用されています。また、UCS XML インターフェイスは、HTTP または HTTPS で送信される XML ドキュメント（API）に対応しています。クライアントの開発者は、好みのプログラム言語を使用して、API メソッドが含まれる XML ドキュメントを作成できます。

UCS Manager の API は、オブジェクト ベースの方式に従っています。トランザクショナルであり、1 つのデータ モデル上で完結します。これが、この API が従来の関数呼び出しをベースとする API と異なる点です。状態の変化に対する UCS の対応は、公開されているタスクごとに別々の API 関数を使用のではなく、API（XML ドキュメント）を通して行います。さらに、その総合的かつ標準的な構造によって、UCS の XML ベースの API は非常に強力なツールとなっており、学習や実装方法もシンプルです。

UCS Manager API のモデルは再帰的な方式であり、この方式で開発者に対して主要な機能が提供されています。たとえば、変更は、1 つのオブジェクト、オブジェクトのサブツリー、またはオブジェクト ツリー全体に対して行うことができます。このため、開発者は、1 回の API の呼び出しによって UCS 内のオブジェクトの属性 1 つを変更したり、また 1 回の API の呼び出しで UCS の全体構造（シャーシ、ブレード、アダプタ、ポリシーなどを含む）を設定したりすることができます。

UCS Manager API では、非同期モデルを活用して拡張性やパフォーマンスを実現しています。言い換えれば、プロセスの適用を、そのプロセスに対する応答が後のあらゆるタイミングで発生したときに行うことができます。この場合、API はプロセスが応答したときにいつでも対応します。API には、完全なイベント サブスクリプションも含まれます。これは、HTTP over TCP または HTTPS over TCP セッションを開くことができ、特定のイベントをメッセージとして転送することを意味します（このプロセスの詳細な説明と例については後述します）。

（注） このマニュアルでは、読みやすくすることを目的として、「<real cookie>」という語句で実際の Cookie を表します。たとえば、「1217377205/85f7ff49-e4ec-42fc-9437-da77a1a2c4bf」と表記する代わりに、その場所に <real cookie> と表記します。

用語集

DME	データ管理エンジン	UCS GUI の中核。トランザクション エンジンと情報リポジトリ（管理情報ツリー）で構成されます。
DN/dn	認定者名	すべての Managed Object（MO; 管理対象オブジェクト）の不変のプロパティ。MO の一意の完全修飾名を表します。
IM	情報モデル	MO クラス規則のドメイン固有の公式仕様（プロパティ、包含、継承など）、および管理フレームワークで使用するサービス API。
IMXML	情報モデル XML	ネイティブな Information Model（IM; 情報モデル）の XML 表現、および IM にアクセスして操作するための API。これには、MO の取得と設定を行うためのメカニズム（RPC）と、イベントを登録するメカニズムが含まれます。
LS	論理サーバ	サーバが使用する識別情報、Processing Node（PN; プロセッシング ノード）要件、接続要件、アソシエーション ポリシー、およびストレージ リソースの定義。Logical Server（LS; 論理サーバ）は PN でアクティブ化されます。
MIM	管理情報モデル	管理対象オブジェクトのツリー構造。
MIT	管理情報ツリー	すべての管理対象オブジェクト（MO）のインスタンスのリポジトリ。それぞれの認定者名（DN）で表します。Management Information Tree（MIT; 管理情報ツリー）は、「ファイル」のフル パス（この場合は MO）名を表すのにメイン フォルダから次の下位フォルダへと進んでいくファイル構造と似ています（RN を参照してください）。

MO	管理対象オブジェクト	管理フレームワークのすべてのオブジェクトのベースとなるクラス。すべての Managed Object（MO; 管理対象オブジェクト）クラスは、情報モデル（IM）によって指定されます。MO インスタンスは、すべて MIT に格納され、アクセスするにはそれぞれの認定者名（DN）または Relative Name（RN; 相対名）を使用します。
PN	プロセッシング ノード	ブレード サーバ クラスのプロセッサとメモリの複合体。統合された I/O 接続のためのアダプタ（物理サーバ）を持っています。
RBAC	ロール ベース アクセス コントロール	さまざまなユーザが実行できる操作の種類を決定する方法。システム内の認可されたユーザのさまざまな機能へのアクセスを制限します。
RN/rn	相対名	コンテナ オブジェクトの名前から相対的に付けられたオブジェクトの名前。ファイルの相対パス名に似ています（この意味で、DN はファイルのフル パス名に相当します）。
スティミュラス		キューイングや処理を行うデータ管理エンジン（DME）および Application Gateway（AG; アプリケーション ゲートウェイ）への入力イベント。
サービス プロファイル		サーバが使用する識別情報、PN 要件、接続要件、アソシエーション ポリシー、およびストレージ リソース。

Unified Computing System の概要

UCS の 1 ユニットは、最大 2 台の相互接続された Cisco UCS 6120XP ファブリック インターコネクトと、ブレード 1 枚を搭載した Cisco 5108 シャーシ 1 台以上で構成します。最大でブレードを 320 枚を搭載した 40 台のシャーシを接続して、1 つの UCS インスタンスから制御できます。

UCS インターフェイス

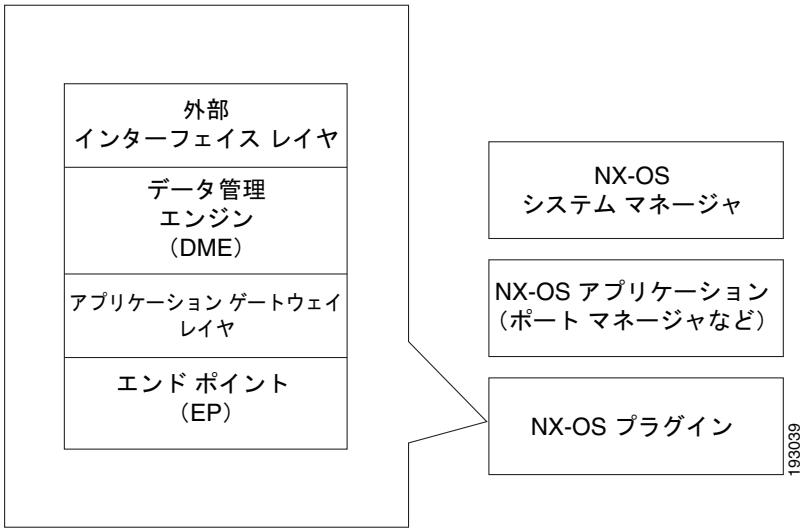
UCS には複数のインターフェイスがあります。これらの中には、Cisco UCS 6120XP ファブリック インターコネクトで終端するものと、Cisco 5108 シャーシで終端するものがあります。次の表を参照してください。

表 1-1　UCS のインターフェイス

ファブリック インターコネクトで終端	サーバのシャーシで終端
標準	CLI
SNMP	GUI
SMASH CLP	XML API
CIM XML	

他のインターフェイスと同様に、この XML API はサーバのバーチャル IP アドレスや、組み込まれた Unified Computing System Manager（UCSM）と結び付いています。このマネージャは 1 台または複数のスイッチ上に存在します。UCS へのすべての要求は、アクティブな UCSM で非同期に処理され、完結します。UCSM は、状態の更新など、すべてのエンド ノード通信を担当します。

図 1-1　UCSM はプラグインであり、NX-OS 内に存在



UCSM コンポーネントについての高度な概要

UCSM ではモデル駆動型アーキテクチャを使用しています。このアーキテクチャでは、変更は管理対象オブジェクトの形式の論理構造に適用されます。MO では、要求されたエンドポイントの状態を実現するために、エンドポイントをどのように設定するべきかを決定します。変更はすべて非同期に適用されます。

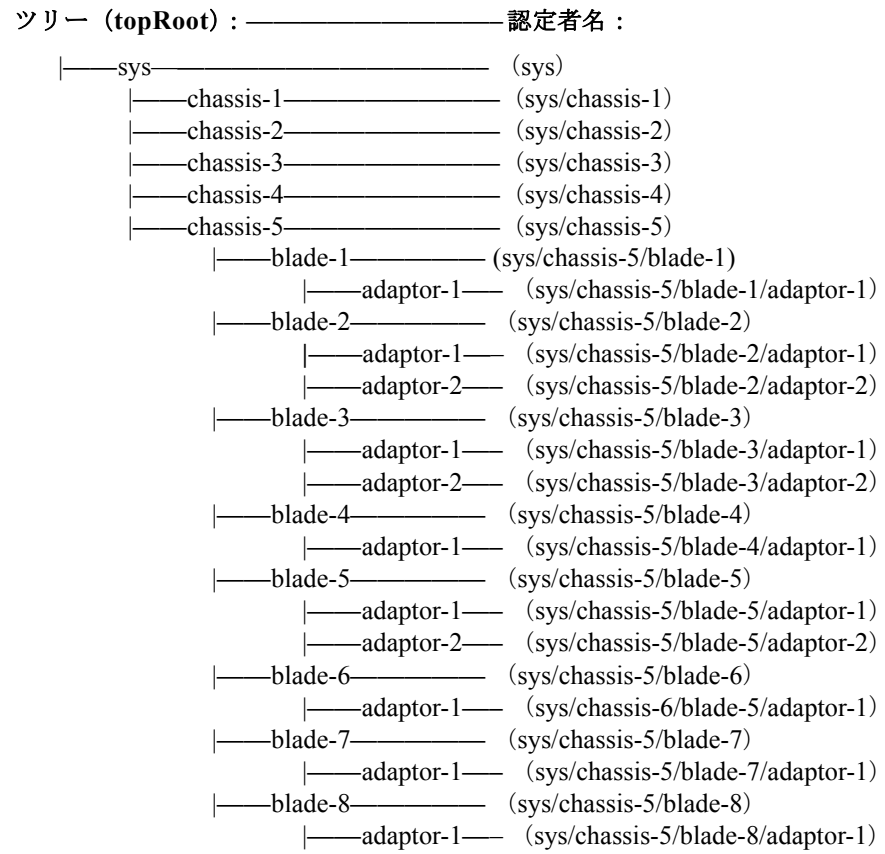
外部インターフェイス レイヤ	外側の世界に向けたノースバウンド インターフェイス XML API、CIM XML、WS-MAN、CLI、SMASH CLP、SNMP
データ管理エンジン （DME）	データ管理エンジンとは、UCS 内の「制御ソース」であり、次の項目の管理と維持を行います。 • オブジェクトのライフ サイクル管理 • モデル情報ツリー、データベースなど • トランザクション サービス • データの変化 • データや状態の複製 • UCSM 向けの HA サービス • ビジネス ロジックと動作のルール • AG に対する非同期配備 • 検証と修正 • RBAC 制御

アプリケーション ゲートウェイ（AG）	モデル中心の言語（メッセージ）とエンドポイント固有の言語間の変換レイヤ。AG は抽象的なレイヤであり、DME とエンドポイント間のトランスレータとして動作します。また、サウスバウンド インターフェイス（エンドポイント）に対する設定、ポーリング、イベントの処理も行います。
エンドポイント	管理対象のエンティティ、スイッチ、ファブリック エクステンダ、シャーシ、アダプタなど。

UCS 管理の情報モデル

UCS の設定には、運用の設定と管理の設定があります。データはツリー構造によって階層的に組織化され、最上位（ルート）から始まり、子ノードと親ノードが含まれます。ツリー内の各ノードは管理対象オブジェクトであり、UCS 内の各オブジェクトには一意の Distinguish Name（DN; 認定者名）があり、この名前によってオブジェクトとツリー内での位置が示されます。たとえば、次の図は、UCSM MIT の「**topRoot**」の下にある「**sys**」から始まるブランチを示しています。これは、シャーシが 5 台据え付けられており（この場合は各シャーシに 8 枚のブレード）、各ブレードには 1 つ以上のアダプタがあります（簡潔にするために、5 番目のシャーシだけを展開しています）。

図 1-2　　5 台のシャーシがある MIN 構造の図



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

管理対象オブジェクトとは、実世界のリソースを抽象化したものです。スイッチ、シャーシ、ブレードなど、UCS の物理コンポーネントおよび論理コンポーネントを表します。Managed Object（MO; 管理対象オブジェクト）のプロパティは、設定と動作で特徴付けられます。

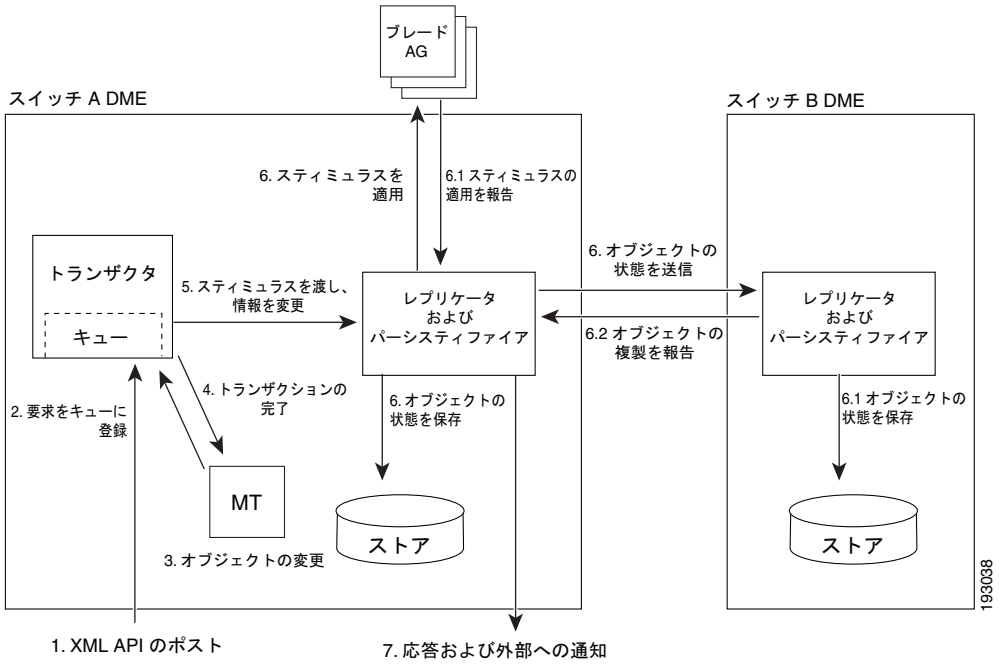
設定ポリシーとは、システム内のポリシーの大部分を占めるものであり、UCS のさまざまなコンポーネントの設定を表すために使用されます。ポリシーは、ある環境下でシステムがどのように動作するかを決定します。MO の中にはユーザが作成したものではなく、UCS によって自動的に作成されたものがあります。電源モジュール オブジェクトやファン オブジェクトなどがシステムによって自動的に作成されたものの例に相当します。重要なのは、XML API を使用して Management Information Model（MIM; 管理情報モデル）と相互にやり取りができるという点です。

XML API のフローの例

要求が DME（データ管理エンジン）に届き、FIFO 方式のトランザクタ キューに置かれます。次に、トランザクタがこのキューから要求を取り出し、要求を解釈して認可チェックを実行します。要求の確認が終了すると、トランザクタが メッセージ情報ツリー（MIT）を更新します。これは 1 つのトランザクションで行われます。

MIT が変更されると、トランザクションは完了です。トランザクタが「実行者」（レプリケータおよびパーシスティファイア）にスティミュラスと変更情報を渡します。実行者は、スタンバイ マネージャとして動作するセカンド スイッチへの更新の複製と、更新をローカルのフラッシュ ストアに保存すると同時にアプリケーション ゲートウェイ（AG）に非同期にスティミュラスを適用することに責任を持ちます。必要に応じて、イベント通知が外部のサブスクライバに送信されます。

図 1-3　　UCS での XML API のフロー



名前付け

ターゲット オブジェクトを識別するための方法には、認定者名（dn）による方法と、相対名（rn）による方法の 2 通りがあります。

認定者名

認定者名（dn）では、明確にターゲット オブジェクトを識別することができます。認定者名（dn）の形式は次のとおりです。

<…**dn** = "sys/chassis-5/blade-2/adaptor-1" />

dn は、オブジェクト ツリーの最上位からそのオブジェクトへと続く完全修飾パスを表します。DN は API 呼び出しによって影響を受ける管理対象オブジェクトを正確に表します。これはファイル システムに似ています。ファイル システムでは /etc/hosts などのフル パスを使用して 1 つのファイルを表します。

相対名

相対名（rn）は、所定の状況の中でオブジェクトを一意に識別します。dn は rn の並びによって構成されていることに注意してください。次に例を示します。

「sys/chassis-1/blade-1/adaptor-1/host-eth-2」という状況は、次の表現として考えることができます。

dn = <root object>/{rn}/{rn}/{rn}/{rn}/{rn}

また次の rn も、この状況でオブジェクトを特定するのに使用できます（たとえば、「adaptor-1」など）。

<… **rn** ="../" />

ファイル システム コマンドの「../」に似ています。これを使用して現在のディレクトリを変更することができます。

cd /etc/sysconfig

cat ../hosts

メソッドのカテゴリ

UCS とのやり取りに使用するメソッドは、4 つのカテゴリに分けられます。各 API はメソッドであり、各メソッドは XML ドキュメントに相当することに注意してください。次にさまざまなメソッドを一覧します。

- 「認証メソッド」（P.1-8）
  - ログイン
  - リフレッシュ
  - ログアウト
- 「クエリー メソッド」（P.1-11）
  - dn または複数の dn の指定による
  - クラスの指定による

  - 親または子の指定による
  - クラスおよび dn の指定による
  - フィルタ タイプ（Simple、Property、Composite、および Modifier）
- 「設定メソッド」（P.1-15）
  - 管理対象オブジェクト、1 つのサブツリー
  - 複数の管理対象オブジェクト、複数のサブツリー
  - 管理対象グループ、フィルタ基準に対応する複数のサブツリー
- 「イベント サブスクリプション メソッド」（P.1-16）
  - イベント、状態の変化

これらのメソッドについては、以降の項で説明します。

寛容モード

UCS の XML API は、「寛容モード」で動作します。これは、指定されていない属性に対して（可能であれば）DME のデフォルト値が使用され、DME が受信した不正な値は無視されるというモードです。さらに、複数の管理対象オブジェクト（仮想 NIC など）を設定する場合、いずれかの管理対象オブジェクトを設定できないときには、API が動作を中止し、設定を前の状態に戻した後、API の処理を中止します。

認証メソッド

認証メソッドは、セッションの認証と維持に使用されます。UCS ではデフォルトで TCP 80（セキュアな接続には 443）を使用します。要求は TCP を使用すると想定されています。UCS では、他の API 呼び出しを許可する前に、認証が正しく実行されることが必要です。

（注）　このマニュアルでは XML API に重点を置き、HTTP over TCP または HTTPS over TCP 接続がスクリプト言語またはプログラミング言語で処理されることを前提としています。

UCS に対して行われる XML API 要求は、cookie によって認証を受けます。有効な cookie を入手するためには、ユーザはログインして認証を受ける必要があります。これには、**aaaLogin** メソッドを使用します。セッションを正しく終了して、cookie を無効にするには、ログアウトを実行する必要があります。これには、**aaaLogout** メソッドを使用します。

接続が確立され、認証が正しく実行されると、応答で cookie が返されます。この cookie はデフォルトで 600 秒間（10 分間）有効であり、セッション期間中にリフレッシュして cookie が期限切れにならないようにする必要があります。cookie のリフレッシュの際には、デフォルトの間隔で新しい cookie を指定します。

この処理を説明するために、次の一般的な例では、標準的な Telnet クライアントを使用して UCS にログインし、認証を受けています。TCP セッションをポート 80 で確立した後、XML ドキュメントを作成して XML API メソッドの「**aaaLogin**」を呼び出します。このドキュメントには、キーボードを使用して HTTP メッセージに XML ドキュメントをカプセル化します。

```
#>telnet 10.10.10.10 80
```

要求のポスト：

```
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="mgmt02-dummy"
outChannel="noencssl" outEvtChannel="noencssl"> </aaaLogin>
```

上記の例では、次のことに注意してください。

1. 操作は HTTP の post メソッドによって行われる。
2. HTTP のエンベロープに XML の設定が含まれる。
3. URI/path は常に「/nuova」である。
4. UCSM は HTTP 要求と HTTPS 要求の両方に対応している。

ログインの例

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。

（注）　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 ドキュメントであり、この例にはルート 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 行目の XML エレメントは「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 行目の「inName」は属性名であり、属性値は「bob」です。

3 行目の「inPassword」は属性名であり、属性値は「abc123」です。

2 行目と 3 行目の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 エレメントの「aaaLogin」を閉じて XML ドキュメントを完了させるタグです。

上記の XML ドキュメントは、コンテンツをまったく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：

```
<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="mgmt4711 "
    outChannel="noencssl"
    outEvtChannel="noencssl">
/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：

```
<aaaLogin
    cookie=""
    response="yes"
    errorCode="551"
    invocationResult="unidentified-fail"
    errorDescr="Authentication failed">
</aaaLogin>
```

この API 応答は完全な XML ドキュメントです。この例では、複数の属性を持つルート エレメント「aaaLogin」だけが含まれています。

1 行目は、エレメント「aaaLogin」が UCS へのログインするために使用する API メソッドであることを示しています。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 行目は属性「outCookie」であり、値は「1217377205/85f7ff49-e4ec-42fc-9437-da77a1a2c4bf」です。

この属性はセッションの cookie です。

4 行目は属性「outRefreshPeriod」であり、値は「600」です。これは推奨される 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 行目は属性「outPriv」の値であり、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 行目は属性「outDomains」であり、値は「mgmt4711-dummy」です。これは UCS の一意の名前です。

7 行目は属性「outChannel」であり、値は「noencssl」です。これ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 行目は属性「outEvtChannel」であり、値は「noencssl」です。これ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。

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） 8 行目のイベント サブスクリプションについては、後述します（「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）。

クエリー メソッド

XML API インターフェイスには、豊富なクエリー インターフェイスがあり、ユーザが現在の設定状態を問い合せるのに使用できます。クエリーの種類を表 1-2 に示します。

表 1-2　クエリーの種類

クエリー メソッド	説明
configResolveDn	dn を指定してオブジェクトを取得する。
configResolveDns	複数の dn を指定してオブジェクトを取得する。
configResolveClass	クラスを指定してオブジェクトを取得する。
configResolveParent	オブジェクトの親のオブジェクトを取得する。
configResolveChildren	オブジェクトの子のオブジェクトを取得する。
configScope	ツリー内の dn からクラスのクエリーを実行する。

クエリーを実行するとき、ほとんどのメソッドは「**inHierarchical**」引数を使用します。この値は true または false です（「yes」または「no」も機能します）。「**inHierarchical**」引数の例は次のとおりです。

```
< configResolveDn … inHierarchical="false"></>
```

または

```
<configResolveDn … inHierarchical="true"></>
```

inHierarchical 引数は、UCS にすべての子とその情報を返すかどうかも指示します。多くのクエリー メソッドで、「**inRecursive**」引数も使用できます。この引数は、この呼び出しを再帰的に実行するかどうか、つまり他のオブジェクトまたは親オブジェクトを参照するオブジェクトに従うかどうかを、システムに示します。

configResolveDn クエリーの例

次のクエリー メソッドの例では、管理対象オブジェクトをその dn で取得します。

この XML は dn によるクエリーを実行するために使用します。

クライアントのポスト：

<**configResolveDn** ***cookie****="<real cookie>"* ***dn****="sys/chassis-1/blade-1"* ***inHierarchical****="false"*></**configResolveDn**>

応答：

<**configResolveDn dn**="*sys/chassis-1/blade-1*" **cookie**="<real cookie>" **response**="yes"> <**outConfig**> <**computeBlade adminPower**="policy" **adminState**="in-service" **assignedToDn**="" **association**="none" **availability**="available" **chassisId**="1" **checkPoint**="discovered" **connPath**="A" **connStatus**="A" **descr**="" **discovery**="complete" **dn**="sys/chassis-1/blade-1" **fltAggr**="12884901888" **fsmDescr**="" **fsmFlags**="" **fsmPrev**="PollSmbiosSuccess" **fsmProgr**="100" **fsmRmtInvErrCode**="none" **fsmRmtInvErrDescr**="" **fsmRmtInvRslt**="" **fsmStageDescr**="" **fsmStamp**="2009-01-24T21:01:30" **fsmStatus**="nop" **fsmTry**="0" **intId**="65031" **lc**="discovered" **managingInst**="A" **model**="N20-B6620-1" **name**="" **numOfAdaptors**="1" **numOfCores**="8" **numOfEthHostIfs**="0" **numOfFcHostIfs**="0" **numOfThreads**="16" **operPower**="on" **operQualifier**="" **operState**="unassociated" **operability**="unknown" **originalUuid**="FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF" **presence**="equipped" **revision**="0" **serial**="QCI12380009" **slotId**="1" **totalMemory**="12288" **uuid**="FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF" **vendor**="Cisco Systems Inc"/> </**outConfig**> </**configResolveDn**>

この応答からは configResolveDn メソッドによってクエリーを実行した管理対象オブジェクト（blade-1）について、相当な量の情報が入手できることに注意してください。

クエリー フィルタ

UCS の XML API には一連のフィルタがあり、クエリー メソッドを強力かつ使いやすくしています。これらのフィルタは、クエリーの一部として渡すことができ、必要な結果セットを特定するのに使用します。フィルタは次のように分類できます。

- 「Simple フィルタ」（P.1-12）
- 「Property フィルタ」（P.1-13）
- 「Composite フィルタ」（P.1-14）
- 「Modifier フィルタ」（P.1-15）

Simple フィルタ

Simple フィルタには、「True フィルタ」と「False フィルタ」の 2 つがあります。これら 2 つのフィルタは、それぞれに true と false の単純な状態に対して反応します。

Property フィルタ

Property フィルタでは、オブジェクトのプロパティの値を「結果セット」に含まれる内容の基準として使用します。

Greater than or Equal to フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティが指定したプロパティ値よりも大きいか等しいオブジェクトに限定します。Greater than or Equal to フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するための値が必要です。

Greater than フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティが指定したプロパティ値よりも大きいオブジェクトに限定します。Greater than フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するための値が必要です。

Inequality フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティが指定したプロパティ値と等しくないオブジェクトに限定します。Inequality フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するための値が必要です。

Less than フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティが指定したプロパティ値よりも小さいオブジェクトに限定します。Less than フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するための値が必要です。

Less than or Equal to フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティが指定したプロパティ値よりも小さいか等しいオブジェクトに限定します。Less than or Equal to フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するための値が必要です。

Equality フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティが指定したプロパティ値と等しいオブジェクトに限定します。Equality フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するための値が必要です。

Wildcard フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティが指定したプロパティ値に一致するものに限定します。Wildcard フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するためのワイルドカード値が必要です。サポートされているワイルドカードには、「%」または「*」（あらゆる文字の並び）、「?」または「-」（任意の 1 文字）があります。

All Bits フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティ値に渡されたビット セットがすべて含まれるオブジェクトに限定します。これはビットマスク プロパティにだけ使用できます。All Bits フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するための値が必要です。

Any Bits フィルタ

このフィルタは、結果セットの範囲を、特定のプロパティ値に渡されたビット セットの少なくとも 1 つが含まれるオブジェクトに限定します。これはビットマスク プロパティにだけ使用できます。Any Bits フィルタを作成するには、ターゲット オブジェクトまたはプロパティの classId および propertyId と、比較するための値が必要です。たとえば、このフィルタは、boot-order-pxe および mac-address-assignments の config-qualifier のいずれかを持つすべての論理サーバにクエリーを実行する場合に使用できます。

Composite フィルタ

Composite フィルタは、他のフィルタを 1 つ以上組み合わせたフィルタです。結果セットを得るための、より強力な該当基準を作成することができます。結果セットは、Composite フィルタの種類と、Composite フィルタから返される結果によって決まります。たとえば、Composite フィルタによって、含まれているフィルタの少なくとも 1 つに該当するオブジェクトだけに結果セットを制限することができます。

AND フィルタ

AND フィルタは、結果セットの内容を、Composite フィルタの個別のフィルタの基準を満たすオブジェクトだけに限定します。たとえば、Composite フィルタを使用して、totalMemory が 64MB より大きく、使用の可否が「operable」であるすべての計算ブレードを抽出することができます。この場合、この Composite フィルタには Greater than フィルタを 1 つと、Equality フィルタを 1 つ使用します。

OR フィルタ

OR フィルタは、結果セットの内容を、Composite フィルタの個別のフィルタの基準のうちの少なくとも 1 つを満たすオブジェクトだけに限定します。たとえば、Composite フィルタを使用して、「assignmentState」が「unassigned」、あるいはアソシエーション状態の値が「unassociated」である論理サーバすべてを抽出することができます。この場合はプロパティ値について 2 つの Equality フィルタを使用します。

XOR フィルタ

XOR フィルタは、結果セットの内容を、Composite フィルタの個別のフィルタの基準のうちの少なくとも 1 つを満たすものの、すべての基準は満たさないオブジェクトだけに限定します。

Between フィルタ

Between フィルタは、結果セットの内容を、Composite フィルタの個別のフィルタの基準のうち、少なくとも 1 つを満たすものの、すべての基準は満たさないオブジェクトだけに限定します。

Modifier フィルタ

Modifier フィルタは、含まれているフィルタの結果を変更します。現在サポートされている Modifier フィルタは 1 つだけです。

NOT フィルタ

このフィルタは、含まれているフィルタの結果を逆にします。含まれているフィルタに一致しないオブジェクトの抽出に使用できます。

フィルタの使用例

AND、OR、および NOT の組み合わせの例は次のとおりです。この例では、このクエリー メソッドを使用して、5 番を除くすべてのシャーシを対象として、スロット 1 またはスロット 8 にある「computeBlade」という種類のオブジェクトをすべて抽出しています。

```
<configResolveClass inHierarchical="false" cookie="<real cookie>" classId="computeBlade">
 <inFilter>
  <and>
   <or>
    <eq class="computeBlade" property="slotId" value="1"/>
    <eq class="computeBlade" property="slotId" value="8"/>
   </or>
   <not>
    <eq class="computeBlade" property="chassisId" value="5"/>
   </not>
  </and>
 </inFilter>
</configResolveClass>
```

設定メソッド

前述したように、UCS のオブジェクト モデルはシステムの最上位にルートを持つ設定ツリーになっています。XML API インターフェイスには、管理対象オブジェクトの設定変更を行うメソッドがいくつか用意されています。変更は個々のオブジェクトに対してではなく、1 つまたは複数のサブツリーに対して行うことに注意してください。

表 1-3 には、さまざまな設定の種類を表しています。

表 1-3　設定の種類

設定の種類	説明
configConfMo	1 つのサブツリー（dn）に影響
configConfMos	複数のサブツリー（dn）に影響
configMoGroup	このメソッドでは、複数のサブツリー構造（dn）または管理対象オブジェクトに対して同じ設定変更を行うことが可能。

設定の際、ほとんどのメソッドでは引数として「inHierarchical」を使用します。これには「true」または「false」（「yes」または「no」も同様に機能）の値を指定します。ただし、子オブジェクトはすべて XML ドキュメントに含まれており、DME は「寛容モード」で動作しているため、これらの値は設定中には大きな意味は持ちません。

ConfConfigMos の例

ConfigConfMos（Configure Managed Objects）メソッドでは、1 つの API コールで複数のサブツリーに影響を与えることができます。「pair key=<dn>」は 1 つのサブツリーの位置を指し示しますが、「pair key」の定義を 2 つ使用することにより、1 つのツリー内の異なる 2 つの場所を指し示すことができます。次の例では、2 つの組織である「HR」と「Finance」が、1 つの呼び出しでどのように削除されるかを表しています。

クライアントのポスト：

<**configConfMos cookie**="<real cookie>" **inHierarchical**="false"> <**inConfigs**> <**pair key**="org-root/org-HR"> <**orgOrg dn**="*org-root/org-HR*" **status**="*deleted*"> </**orgOrg**> </**pair**> <**pair key**="org-root/org-Finance"> <**orgOrg dn**="*org-root/org-Finance*" **status**="*deleted*"> </**orgOrg**> </**pair**> </**inConfigs**> </**configConfMos**>

応答：

<**configConfMos cookie**="<real cookie>" **response**="yes"> <**outConfigs**> <**pair key**="org-root/org-HR"> <**orgOrg dn**="*org-root/org-HR*" **fltAggr**="0" **level**="1" **name**="HR" **status**="*deleted*"/> </**pair**> <**pair key**="org-root/org-Finance"> <**orgOrg dn**="*org-root/org-Finance*" **fltAggr**="0" **level**="1" **name**="Finance" **status**="*deleted*"/> </**pair**> </**outConfigs**> </**configConfMos**>

この応答の中で、**dn**="*org-root/org-HR*" と **dn**="*org-root/org-Finance*" の両方で「**status**」が「*deleted*」として返されていることに注意してください。

イベント サブスクリプション メソッド

ユーザによる処理の開始やシステムの処理によってオブジェクトに変化（変更、作成、削除）が生じたときには、必ずイベントが生成されます。

状態の変化に関する情報を入手するプログラムには、一般的な 2 つの方法があります。1 つは定期的にポーリングを行う方法で、もう 1 つはイベント サブスクリプションを行う方法です。ポーリングはネットワーク リソースの消費量が非常に高いため、最後の手段として使用することを推奨します。

これとは逆に、イベント サブスクリプションではクライアントが UCS からのイベント通知を受けるよう登録できます。サブスクライブされている場合、イベントが発生すると、UCS からプログラムに対してイベントとそのタイプが通知されます。また、UCS は差分（実際の変更）だけを送信し、影響を受けたオブジェクトの属性は送信しません。これはシステム内のすべてのオブジェクトの変化に対応しています。

「eventSubscribe」メソッドは、イベントの登録に使用します（次の例を参照）。

<**eventSubscribe cookie**="<*real cookie*>"></**eventSubscribe**>

イベント サブスクリプションを使用するには、TCP 上で HTTP セッションまたは HTTPS セッションを開き、このセッションを開いたままにします。UCS からはサブスクリプションの確認について応答はありませんが、新たなイベントが発生した時点で送信を開始します。

各イベントには一意の「イベント ID」が付けられています。これらのイベント　ID はカウンタとして機能します。この UCS の場合、最後に発生したイベントは **174711** です。このイベント ID は、すべてのメソッドの応答に含まれます。イベントが発生するたびに、イベント ID のカウンタが増え、新しいイベントに新しいイベント ID が割り当てられます。この仕組みによって、サブスクライバはイベントの追跡を継続することができ、イベントの発生を見逃さないようになります。クライアントがイベントを見逃した場合には、「eventSendEvent」を使用して見逃したイベントを入手することができます。

UCS からサブスクライブ プロセスに対して送信される 3 つの異なるイベント（**174712**、**174713**、および **174714**）の例は次のとおりです。3 つのイベントが 1 つの XML 構造にラップされていることに注意してください。

```
<methodVessel cookie="<real cookie>">
    <inStimuli>
        <configMoChangeEvent cookie="<real cookie>" inEid="174712">
            <inConfig>
                <callhomeEp
                dn="call-home"
                fsmPrev="configCallhomeSetLocal"
                fsmStamp="2008-10-16T17:59:25"
                fsmTry="11"
                status="modified"
                />
            </inConfig>
        </configMoChangeEvent>
        <configMoChangeEvent cookie="<real cookie>" inEid="174713">
            <inConfig>
                <mgmtIf
                dn="sys/switch-A/mgmt/if-1"
                fsmPrev="SwMgmtOobIfConfigSwitch"
                fsmStamp="2008-10-16T17:59:25"
                fsmTry="9"
                status="modified"
                />
            </inConfig>
        </configMoChangeEvent>
        <configMoChangeEvent cookie="<real cookie>" inEid="174714">
            <inConfig>
                <eventRecord
                affected="sys/sysdebug/file-export"
                cause="transition"
                created="2008-10-16T17:59:25"
                descr="[FSM:STAGE:RETRY:8]: configuring automatic core file
                export service on local"
                dn="event-log/54344"
                id="54344"
                ind="state-transition"
                severity="info"
                status="created"
```

```
                                    trig="special"
                                    txId="24839"
                                    user="internal"/>
                            </inConfig>
                        </configMoChangeEvent>
                    </inStimuli>
</methodVessel>
```

成否についての XML の応答

要求の処理が正しく行われたときには、XML ドキュメントで要求された情報または変更が行われたことを示す確認事項が返されます。blade-1 に対する「dn の指定による解決」の例は次のとおりです。

```
<configResolveDn dn="sys/chassis-1/blade-1" cookie="<real cookie>" response="yes"> <outConfig>
<computeBlade adminPower="policy" adminState="in-service" assignedToDn=""
association="none" availability="available" chassisId="1" checkPoint="discovered"  connPath="A"
connStatus="A" discovery="complete" dn="sys/chassis-1/blade-1" fltAggr="0" fsmDescr=""
fsmFlags="" fsmPrev="DiscoverSuccess" fsmRmtInvErrCode="unspecified"
fsmRmtInvErrDescr="" fsmRmtInvRslt="" fsmStageDescr="" fsmStamp="2008-11-24T01:27:10"
fsmStatus="nop" fsmTry="0" lc="discovered" managingInst="A" model="Gooding" name=""
numOfAdaptors="1" numOfCores="4" numOfEthHostIfs="2" numOfFcHostIfs="2"
numOfThreads="0" operPower="off" operState="unassociated" operability="operable"
originalUuid="1b4e28ba-2fa1-11d2-0101-b9a761bde3fb" presence="equipped" revision=""
serial="1-1" slotId="1"  totalMemory="4096" uuid="" vendor="Nuova"/> </outConfig>
</configResolveDn>
```

成功したものの空の結果

存在しないオブジェクトに対するクエリー要求は、API エンジンによって失敗とは扱われません。オブジェクトが存在しない場合には、エンジンから成功のメッセージが返されますが、XML ドキュメントによって空のデータ「<outConfig> </outConfig>」が返されて、要求されたオブジェクトが見つからなかったことが示されます。

存在しない blade-4711 に対する「dn の指定による解決」の例は次のとおりです。

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

失敗した要求

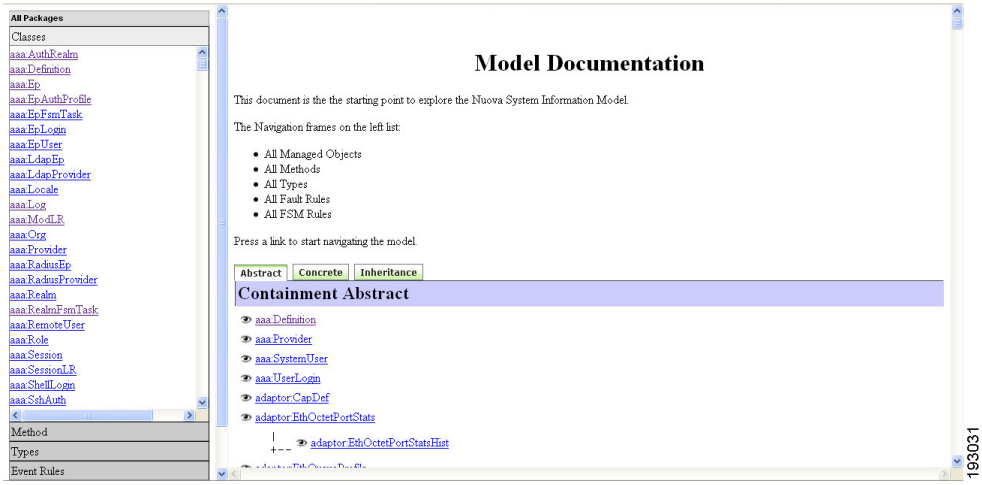
処理に失敗した要求の場合は、「errorCode」および「errorDescr」という XML 属性が含まれます。次に失敗した要求の例を示します。

```
<configConfMo dn="fabric/server" cookie="<real cookie>" response="yes" errorCode="103"
invocationResult="unidentified-fail" errorDescr="can't create; object already exists.">
<configConfMo>
```

API の『Model Documentation』

UCS Manager XML API は、すべてオンラインの API HTML ドキュメントで完全にマニュアル化されています。『Model Documentation』は、内容が豊富なハイパーテキスト方式の専門辞典となっており、API 全体に関するあらゆる情報が記載されています。図 1-4 は、API の『Model Documentation』の最初のページを表しています。

図 1-4　API の『Model Documentation』の最初のページ



API XML の『Model Documentation』では、次の論理グループについて、詳しく説明しています。

- クラス
- メソッド
- 種類
- イベント ルール
- 障害ルール
- FSM ルール

UCS Manager API XML の『Model Documentation』は、次の URL にあるシスコの新規および改訂版の技術マニュアルから入手できます。

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html