APPENDIX D

プロビジョニング API の使用例

この付録では、プロビジョニング Application Programming Interface（API; アプリケーション プログラミング インターフェイス）の最も一般的な使用例を取り上げます。詳細や、個々の API コールと機能を説明するサンプルの Java コード セグメントについては、Cisco Broadband Access Center (BAC) 4.0 の API Javadoc を参照してください。

次の使用例は、デバイスまたはサービス（またはその両方）のプロビジョニングに直接関連しています。サービス クラス、DHCP 基準、およびライセンスの管理など、多くの管理操作についてはここでは取り上げません。関連する API コールの詳細については、API Javadoc を参照することをお勧めします。これらのアクティビティのほとんどは、管理者のユーザ インターフェイスを使用して実行することもできます。

この付録では、次の使用例について説明します。

- API クライアントの作成方法（P.D-2）
- 使用例（P.D-5）

API クライアントの作成方法

各使用例を参照する前に、API クライアントの作成方法について十分に理解しておく必要があります。API クライアントを作成するには、この項で説明するワークフローを使用します。

ステップ 1　Provisioning API Command Engine（PACE）への接続を作成します。

```
// The PACE connection to use throughout the example. When
// executing multiple batches in a single process, it is advisable
// to use a single PACE connection that is retrieved at the start
// of the application. When done with the connection, YOU MUST
// explicitly close the connection with the releaseConnection()
// method call.

PACEConnection connection = null;

// Connect to the Regional Distribution Unit (RDU).
//
// The parameters defined at the beginning of this class are
// used here to establish the connection. Connections are
// maintained until releaseConnection() is called. If
// multiple calls to getInstance() are called with the same
// arguments, you must still call releaseConnection() on each
// connection you received.
//
// The call can fail for one of the following reasons:
// - The hostname / port is incorrect.
// - The authentication credentials are invalid.

try
{
    connection = PACEConnectionFactory.getInstance(

          // RDU host      rduHost,
          // RDU port      rduPort,
          // User name     userName,
          // Password       password);
}
catch (PACEConnectionException e)
{
// Handle connection error:

System.out.println("Failed to establish a PACEConnection to [" +
    userName + "@" + rduHost + ":" + rduPort + "]; " +
    e.getMessage());

System.exit(1);
}
```

ステップ 2　バッチの新しいインスタンスを作成します。

```
// To perform any operations in the Provisioning API, you must
// first create a batch. As you add commands to the batch,
// nothing gets executed until you post the batch.
// Multiple batches can be started concurrently against a
// single connection to the RDU.

Batch myBatch = connection.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);
```

ステップ 3　API コマンドをバッチに登録します。このステップの例では、*getDetails(...)* コールを使用しています。

```
// Use the Provisioning API to get all of the information for
// the specified MAC address. Since methods aren't actually
// executed until the batch is posted, the results are not
// returned until after post() completes. The getCommandStatus()
// followed by getData() calls must be used to access the results
// once the batch is posted.

final DeviceID modemMACAddress = DeviceID.getInstance("1,6,00:11:22:33:44:55",
    KeyType.MAC_ADDRESS);

List options = new ArrayList();
options.add(DeviceDetailsOption.INCLUDE_LEASE_INFO);

myBatch.getDetails(modemMACAddress, options);
```

ステップ 4　バッチを Regional Distribution Unit（RDU）サーバに送信します。

```
// Executes the batch against the RDU. All of the
// methods are executed in the order entered.

BatchStatus bStatus = null;
try
{
// Post batch in synchronous fashion without a timeout. This method will block
until
// results are returned. Other API calls are available to submit a batch with
timeout // or in asynchronous (non-blocking) fashion.

bStatus = myBatch.post();
}
catch (ProvisioningException pe)
{
System.out.println("Failed to query for modem with MAC address [" +
    modemMACAddress + "]; " + pe.getMessage());

System.exit(2);
}
```

ステップ 5　バッチのステータスを確認します。

```
// Check if any errors occurred during the execution of the
// batch. Exceptions occur during post() for truly exceptional
// situations such as failure of connectivity to RDU.
// Batch errors occur for inconsistencies such as no lease
// information for a device requiring activation. Command
// errors occur when a particular method has problems, such as
// trying to add a device that already exists.

//check batchStatus and commandStatus
//for any error

CommandStatus commandStatus = null;
if (batchStatus.getCommandCount() > 0)
{
     commandStatus = batchStatus.getCommandStatus(0);
}
if (batchStatus.isError()
    || commandStatus == null
    || commandStatus.isError())
{
        System.out.println("Failed to query for modem with MAC address [" +
           modemMACAddress + "]; " + bs.getStatusCode().toString() + ", "
           + bs.getErrorMessage());
        for (int i = 0; i < bs.getCommandCount(); i++)
        {
           CommandStatus cs = bs.getCommandStatus(i);
           System.out.println("Cmd " + i + ": status code "
                   + cs.getStatusCode().toString() + ", " + cs.getErrorMessage());
        }
}
```

エラーがなければ、バッチ コールは正常終了の結果を返します。

```
// Successfully queried for device.

System.out.println("Queried for DOCSIS modem with MAC address ["+
    modemMACAddress + "]");

// Display the results of the command (TreeMap is sorted). The
// data returned from the batch call is stored on a per-command
// basis. In this example, there is only one command, but if
// you had multiple commands all possibly returning results, you
// could access each result by the index of when it was added.
// The first method added is always index 0. From the status of
// each command, you can then access the accompanying data by
// using the getData() call. Since methods can return data of
// different types, you will have to cast the response to the
// type indicated in the Provisioning API documentation.

Map deviceData = (Map)bStatus.getCommandStatus(0).getData();

// Created a sorted map view

Map<String, Object> deviceDetails = new TreeMap(deviceData);
    for(String key: deviceDetails.keySet())
{
    System.out.println(" " + key + "=" + deviceDetails.get(key));
}
```

ステップ 6　接続を解放します。

```
// Once the last batch has been executed, the connection can
// be closed to the RDU. It is important to explicitly
// close connections since it helps ensure clean shutdown of
// the Java virtual machine.

connection.releaseConnection();
```

使用例

この項には、次の使用例を記載しています。

- 固定標準モードでセルフプロビジョニングされたモデムとコンピュータ（P.D-6）
- 固定標準モードでの新しいコンピュータの追加（P.D-9）
- 加入者のディセーブル化（P.D-11）
- モデムおよびセルフプロビジョニングされたコンピュータの事前プロビジョニング（P.D-13）
- 既存のモデムの修正（P.D-16）
- 加入者のデバイスの登録解除と削除（P.D-17）
- 無差別モードでの最初のアクティベーションのセルフプロビジョニング（P.D-20）
- 無差別モードでの 100 台のモデムの一括プロビジョニング（P.D-24）
- 無差別モードでの最初のアクティベーションの事前プロビジョニング（P.D-26）
- 既存のモデムの交換（P.D-28）
- 無差別モードでの 2 台目のコンピュータの追加（P.D-29）
- NAT を使用した最初のアクティベーションのセルフプロビジョニング（P.D-30）
- NAT を持つモデムの背後への新しいコンピュータの追加（P.D-31）
- 別の DHCP スコープへのデバイスの移動（P.D-32）
- イベントを使用したデバイス削除のロギング（P.D-33）
- イベントを使用した RDU 接続の監視（P.D-34）
- イベントを使用したバッチ完了のロギング（P.D-35）
- デバイスの詳細情報の取得（P.D-35）
- デバイス タイプを使用した検索（P.D-40）
- ベンダー プレフィックスまたはサービス クラスを使用したデバイスの検索（P.D-42）
- PacketCable eMTA の事前プロビジョニング（P.D-43）
- PacketCable eMTA 上での SNMP クローニング（P.D-44）
- PacketCable eMTA の差分プロビジョニング（P.D-46）
- 動的設定ファイルを使用した DOCSIS モデムの事前プロビジョニング（P.D-48）
- オプティミスティック ロッキング（P.D-50）
- 加入者の帯域幅の一時的なスロットリング（P.D-53）
- CableHome WAN-MAN の事前プロビジョニング（P.D-54）
- ファイアウォール設定を持つ CableHome（P.D-56）
- CableHome WAN-MAN のデバイス機能の取得（P.D-58）
- CableHome WAN-MAN のセルフプロビジョニング（P.D-59）

固定標準モードでセルフプロビジョニングされたモデムとコンピュータ

加入者は、戸建住宅内にコンピュータを設置し、DOCSIS ケーブル モデムを購入しました。コンピュータには Web ブラウザがインストールされています。

目的

次のワークフローを使用して、プロビジョニングされていない新しい DOCSIS ケーブル モデムとコンピュータをオンラインにし、適切なレベルのサービスを受けられるようにします。

ステップ 1　加入者は DOCSIS ケーブル モデムを購入して住宅内に設置し、コンピュータをケーブル モデムに接続します。

ステップ 2　加入者はモデムとコンピュータの電源をオンにし、BAC が制限付きアクセス権をモデムに付与します。コンピュータとモデムには、制限付きアクセス プールから IP アドレスが割り当てられます。

ステップ 3　加入者は Web ブラウザを起動します。スプーフィング DNS サーバにより、Web ブラウザがサービス プロバイダーの登録サーバ（OSS ユーザ インターフェイスやメディエータなど）にアクセスします。

ステップ 4　加入者はサービス プロバイダーのユーザ インターフェイスを使用して、サービス クラスの選択など、登録に必要な手順を終了します。

ステップ 5　サービス プロバイダーのユーザ インターフェイスは、加入者の情報（選択されたサービス クラスやコンピュータの IP アドレスなど）を BAC に渡します。次に、BAC が加入者のモデムとコンピュータを登録します。

```
// First we query the computer's information to find the
// modem's MAC Address. We use the computer IP Address (the web browser
// received this when the subscriber opened the service provider's
// web interface

PACEConnection connection = PACEConnectionFactory.getInstance(
            "localhost", 49187, "admin", "changeme");

// NO_ACTIVATION is the activation mode because this is a query.
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device.
// NO_PUBLISHING is the publishing mode because we are not attempting
// to publish to external database.

Batch batch = connection.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// register getAllForIPAddress to the batch

batch.getAllForIPAddress("10.0.14.38");

BatchStatus batchStatus = null;

// post the batch to RDU server
```

```
try
{
batchStatus = batch.post();
}
catch(ProvisioningException e)
{
e.printStackTrace();
}
// Get the LeaseResults object after posting a batch.

CommandStatus commandStatus = batchStatus.getCommandStatus(0);

LeaseResults computerLease = (LeaseResults)commandStatus.getData();

// Derive the modem MAC address from computer's network
// information. The "1,6" is a standard prefix for an Ethernet
// device. The fully qualify MAC Address is required by BACC

StringBuffer modemMACAddress = new StringBuffer();
modemMACAddress.append("1,6,");
modemMACAddress.append(computerLease.getSingleLease().get("relay-agent-remote-id"))
;

// Create MacAddress object from the string

MACAddress modemMACAddressObject = new MACAddress(modemMACAddress.toString());

List<DeviceID> modemDeviceIDList = new ArrayList<DeviceID>();
    modemDeviceIDList.add(modemMACAddressObject);

// Create a new batch to add modem device

batch = connection.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// Register add API to the batch

batch.add(DeviceType.DOCSIS, modemDeviceIDList,
    null, null, "0123-45-6789", "silver", "provisioned-cm", null);

// post the batch to RDU server

// Derive computer MAC address from computer's network information.

String computerMACAddress =
    (String)computerLease.getSingleLease().get(DeviceDetailsKeys.MAC_ADDRESS);

// Create a map for computer property.

Map<String, Object> properties = new HashMap<String, Object>();
properties.put(IPDeviceKeys.MUST_BE_BEHIND_DEVICE, modemMACAddress.toString());

List<DeviceID> compDeviceIDList = new ArrayList<DeviceID>();
MACAddress computerMACAddressObject = new MACAddress(computerMACAddress);
compDeviceIDList.add(computerMACAddressObject);

// Register add API to the batch
```

```
    batch.add(DeviceType.COMPUTER, compDeviceIDList,
        null, null, "0123-45-6789", null, "provisioned-cpe", properties);
    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
```

ステップ 6　プロビジョニング クライアントは *performOperation(DeviceOperation deviceOperation, DeviceID deviceID, Map<String,Object> parameters* を呼び出してモデムをリブートし、プロビジョニングされたアクセス権をモデムに付与します。

```
    // Reset the computer
    // Create a new batch

    batch = connection.newBatch(

        // No reset

        ActivationMode.AUTOMATIC,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION);

    // Register performOperation command to the batch

    batch.performOperation(DeviceOperation.RESET, modemMACAddressObject, null);

    // Post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
        }
    }
}
```

ステップ 7　ユーザ インターフェイスは、加入者にコンピュータをリブートするように求めます。

リブート後、コンピュータは新しい IP アドレスを受信します。これで、ケーブル モデムとコンピュータの両方が、プロビジョニングされたデバイスとなります。コンピュータは、サービス プロバイダーのネットワークを介してインターネットにアクセスできます。

固定標準モードでの新しいコンピュータの追加

Mmultiple System Operator（MSO; マルチプル システム オペレータ）は、加入者に対し、ケーブル モデムの背後に 2 台のコンピュータを接続することを許可しています。加入者は 1 台のコンピュータをすでに登録しています。今度は、職場から自宅にラップトップ コンピュータを持ち帰ってアクセスを設定しようとしています。加入者はハブを設置し、ラップトップをハブに接続します。

目的

次のワークフローを使用して、プロビジョニングされていない新しいコンピュータを、すでにプロビジョニングされているケーブル モデムを使用してオンラインにし、新しいコンピュータが適切なレベルのサービスを受けられるようにします。

ステップ 1　加入者は新しいコンピュータの電源をオンにし、BAC が制限付きアクセス権をコンピュータに付与します。

ステップ 2　加入者は新しいコンピュータの Web ブラウザを起動します。スプーフィング DNS サーバにより、Web ブラウザがサービス プロバイダーの登録サーバ（OSS ユーザ インターフェイスやメディエータなど）にアクセスします。

ステップ 3　加入者はサービス プロバイダーのユーザ インターフェイスを使用して、新しいコンピュータの追加に必要な手順を終了します。

ステップ 4　サービス プロバイダーのユーザ インターフェイスは、加入者の情報（選択されたサービス クラスやコンピュータの IP アドレスなど）を BAC に渡します。次に、BAC が加入者のモデムとコンピュータを登録します。

```
// First we query the computer's information to find the
// modem's MAC Address. We use the computer IP address (the web browser
// received this when the subscriber opened the service provider's
// web interface.
PACEConnection connection = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "changeme");

// NO_ACTIVATION is the activation mode because this is a query
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device
// NO_PUBLISHING is the publishing mode because we are not attempting
// to publish to external database.

Batch batch = connection.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// register getAllForIPAddress to the batch

batch.getAllForIPAddress("10.0.14.39");
BatchStatus batchStatus = null;

// post the batch to RDU server
```

```
try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
{
    e.printStackTrace();
}
// Get the LeaseResults object after posting a batch.

CommandStatus commandStatus = batchStatus.getCommandStatus(0);

LeaseResults computerLease = (LeaseResults)commandStatus.getData();

// derive the modem MAC address from computer's network
// information. The "1,6" is a standard prefix for an Ethernet
// device. The fully qualify MAC Address is required by BACC

StringBuffer modemMACAddress = new StringBuffer();
modemMACAddress.append("1,6,");
modemMACAddress.append(computerLease.getSingleLease().get("relay-agent-remote-id"))
;

// derive computer MAC address from computer's network information.

String computerMACAddress =
    (String)computerLease.getSingleLease().get(DeviceDetailsKeys.MAC_ADDRESS);

//Create a map for computer property.

Map<String, Object> properties = new HashMap<String, Object>();

// setting IPDeviceKeys.MUST_BE_BEHIND_DEVICE on the computer ensures
// that when the computer boots, it will only receive its provisioned
// access when it is behind the given device. If it is not behind
// the given device, it will receive default access (unprovisioned)
// and hence fixed mode.

properties.put(IPDeviceKeys.MUST_BE_BEHIND_DEVICE, modemMACAddress);

// the IPDeviceKeys.MUST_BE_IN_PROV_GROUP ensures that the computer
// will receive its provisioned access only when it is brought up in
// the specified provisioning group. This prevents the computer
// (and/or) the modem from moving from one locality to another
// locality.

properties.put(IPDeviceKeys.MUST_BE_IN_PROV_GROUP, "bostonProvGroup");

List<DeviceID> compDeviceIDList = new ArrayList<DeviceID>();
MACAddress computerMACAddressObject = new MACAddress(computerMACAddress);
compDeviceIDList.add(computerMACAddressObject);

batch = connection.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// register add API to the batch
```

```
    batch.add(
        DeviceType.COMPUTER,    // deviceType: Computer
        compDeviceIDList,       // compDeviceIDList: the list of DeviceIDs derived from
                                // computerLease
        null,                   // hostName: not used in this example
        null,                   // domainName: not used in this example
        "0123-45-6789",         // ownerName
        null,                   // class of service: get the default COS
        "provisionedCPE",       // dhpcCriteria: Network Registrar uses this to
                                  // select a modem lease granting provisioned IP
        address properties                // device properties
    );


    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
        }
    }
}
```

ステップ 5　ユーザ インターフェイスは、加入者にコンピュータをリブートするように求めます。リブートすると、BAC が登録済みのサービス レベルをコンピュータに付与します。

これで、コンピュータはプロビジョニングされたデバイスとなり、適切なレベルのサービスにアクセスできます。

加入者のディセーブル化

加入者が滞納を繰り返した場合、サービス プロバイダーは加入者のインターネット アクセスをディセーブルにする必要があります。

目的

次のワークフローを使用して、動作中のケーブル モデムとコンピュータをディセーブルにします。その結果、デバイスがユーザのインターネット アクセスを一時的に制限します。また、この使用例では、ユーザのブラウザを特別なページにリダイレクトして、次のように通知することがあります。

```
You haven't paid your bill so your Internet access has been disabled.
```

ステップ 1　サービス プロバイダーのアプリケーションはプロビジョニング クライアント プログラムを使用して、加入者のデバイスすべてのリストを BAC に要求します。

ステップ 2　次に、サービス プロバイダーのアプリケーションはプロビジョニング クライアントを使用して、加入者のデバイスを個々にディセーブルにするか、または制限します。

```
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");

//get all for owner ID

Batch batch = conn.newBatch();
batch.getAllForOwnerID("0123-45-6789");

BatchStatus batchStatus = null;
try
{
    batchStatus = batch.post();
}
catch(Exception e)
{
    e.printStackTrace();
}
CommandStatus commandStatus = batchStatus.getCommandStatus(0);


//batch success without error, retrieve the result

RecordSearchResults rcSearchResult = (RecordSearchResults)commandStatus.getData();
List<RecordData> resultList = rcSearchResult.getRecordData();

if (resultList != null)
{
// getting the data

for (int i=0; i<resultList.size(); i++)
{
    RecordData rd = resultList.get(i);
    Map<String, Object> detailMap = rd.getDetails();

    //get the deviceType from the detail map

    String deviceType =
        (String)detailMap.get(DeviceDetailsKeys.DEVICE_TYPE);

    Key primaryKey = rd.getPrimaryKey();

    //only interest in DOCSIS
    if (DeviceType.getDeviceType(deviceType)
        .equals(DeviceType.DOCSIS))
    {

    //change COS

    batch = conn.newBatch();
    batch.changeClassOfService((DeviceID)primaryKey, "DisabledCOS");

    //change DHCPCriteria

    batch.changeDHCPCriteria((DeviceID)primaryKey, "DisabledDHCPCriteria");

    batchStatus = null;
    try
    {
         batchStatus = batch.post();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
```

```
    }
    //disable computer

    else if (DeviceType.getDeviceType(deviceType)
        .equals(DeviceType.COMPUTER))
    {
        //change DHCPCriteria

        batch = conn.newBatch();
        batch.changeClassOfService((DeviceID)primaryKey,
            "DisabledComputerCOS");
        batch.changeDHCPCriteria((DeviceID)primaryKey,
            "DisabledComputerDHCPCriteria");

        batchStatus = null;
        try
        {
            batchStatus = batch.post();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

（注）　DisabledCOS の特性を定義し、モデムをリセットするときに、モデムの背後の CPE に及ぼす影響を考慮しなければならないことがあります。これは、モデムの背後に音声エンドポイントが接続されている場合に特に重要です。ケーブル モデムの動作を中断すると、その時点で進行中の通話に影響を及ぼす可能性があるためです。

これで、加入者がディセーブルになります。

モデムおよびセルフプロビジョニングされたコンピュータの事前プロビジョニング

新しい加入者はサービス プロバイダーに連絡し、サービスを要求します。加入者は、戸建住宅内にコンピュータを設置しています。サービス プロバイダーは、すべてのケーブル モデムをまとめて事前プロビジョニングします。

目的

次のワークフローを使用して、事前プロビジョニングされたケーブル モデムとプロビジョニングされていないコンピュータを、ローミング標準モードでオンラインにします。この手順は、両方のデバイスが適切なレベルのサービスを受け、登録されるために必要です。

ステップ 1　サービス　プロバイダーは、課金システムで使用される加入者のユーザ名とパスワードを選択します。

ステップ 2　サービス プロバイダーは、加入者がアクセスできるサービスを選択します。

ステップ 3　サービス プロバイダーの現場技術者は、新しい加入者の住宅内に物理ケーブルを敷設し、事前プロビジョニングされたデバイスを設置して、加入者のコンピュータに接続します。

ステップ 4　モデムの電源をオンにすると、BAC はプロビジョニングされた IP アドレスをモデムに付与します。

ステップ 5　コンピュータの電源をオンにすると、BAC はプライベート IP アドレスをコンピュータに付与します。

ステップ 6　コンピュータのブラウザ　アプリケーションを起動すると、ブラウザはサービス　プロバイダーのユーザ インターフェイスにアクセスします。

ステップ 7　サービス　プロバイダーのユーザ　インターフェイスにアクセスすると、プロビジョニングされたケーブル モデムの背後にあるコンピュータの登録に必要な手順を終了します。

```
// First we query the computer's information to find the
// modem's MAC Address. We use the computer IP address (the web browser
// received this when the subscriber opened the service provider's
// web interface

PACEConnection connection = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "changeme");

// NO_ACTIVATION is the activation mode because this is a query
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device
// NO_PUBLISHING is the publishing mode because we are not attempting
// to publish to external database.

Batch batch = connection.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// register getAllForIPAddress to the batch

batch.getAllForIPAddress("10.0.14.38");

BatchStatus batchStatus = null;

// post the batch to RDU server

try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
{
    e.printStackTrace();
}

// Get the LeaseResults object after posting a batch.

CommandStatus commandStatus = batchStatus.getCommandStatus(0);

LeaseResults computerLease = (LeaseResults)commandStatus.getData();

// derive computer MAC address from computer's network information.
String computerMACAddress =
    (String)computerLease.getSingleLease().get(DeviceDetailsKeys.MAC_ADDRESS);
```

```
        List<DeviceID> compDeviceIDList = new ArrayList<DeviceID>();
        MACAddress computerMACAddressObject = new MACAddress(computerMACAddress);
        compDeviceIDList.add(computerMACAddressObject);


    // NO_ACTIVATION will generate new configuration for the computer,
    // however it will not attempt to reset it.
    // NO_CONFIRMATION is the confirmation mode because we are not
    // attempting to reset the computer because this cannot be done.

    batch = connection.newBatch(
        // No reset

        ActivationMode.NO_ACTIVATION,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION,

        // No publishing to external database

        PublishingMode.NO_PUBLISHING);

    // register add API to the batch

    batch.add(
        DeviceType.COMPUTER,  // deviceType: Computer
        compDeviceIDList,     // compDeviceIDList: the list of DeviceIDs derived from
                              // computerLease
        null,                 // hostName: not used in this example
        null,                 // domainName: not used in this example
        "0123-45-6789",       // ownerName
        null,                 // class of service: get the default COS
        "provisionedCPE",     // dhpcCriteria: Network Registrar uses this to
                              // select a modem lease granting provisioned IP address
        null                  // properties: not used
    );

    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

（注） *IPDeviceKeys.MUST_BE_BEHIND_DEVICE* プロパティはコンピュータに設定されませんが、このプロパティを使用すると、ケーブル モデムの背後から別のケーブル モデムへのローミングが可能になります。

ステップ 8　コンピュータを再起動すると、コンピュータはプロビジョニングされた新しい IP アドレスを受信します。

これで、ケーブル モデムとコンピュータはいずれもプロビジョニングされたデバイスとなります。コンピュータは、サービス プロバイダーのネットワークを介してインターネットにアクセスできます。

既存のモデムの修正

サービス プロバイダーの加入者は、現在 **Silver** というレベルのサービスを受けていますが、**Gold** サービスにアップグレードすることにしました。加入者は、住宅内にコンピュータを設置しています。

（注）　この使用例の目的は、デバイスの修正方法を示すことです。この例は、ローミング標準以外のモードでプロビジョニングされたデバイスに適用できます。

目的

次のワークフローを使用して、既存のモデムのサービス クラスを修正し、そのサービスの変更をサービス プロバイダーの外部システムに渡します。

ステップ 1　加入者はサービス プロバイダーに電話をかけて、サービスのアップグレードを依頼します。サービス プロバイダーはユーザ インターフェイスを使用して、サービス クラスを **Silver** から **Gold** に変更します。

ステップ 2　サービス プロバイダーのアプリケーションは、BAC で次の API コールを行います。

```
    // NO_ACTIVATION is the activation mode because this is a query
    // NO_CONFIRMATION is the confirmation mode because we are not
    // attempting to reset the device
    // NO_PUBLISHING is the publishing mode because we are not attempting
    // to publish to external database.

    Batch batch = connection.newBatch(

        // No reset

        ActivationMode.NO_ACTIVATION,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION,

        // No publishing to external database

        PublishingMode.NO_PUBLISHING);

    // replace changeClassOfService to this. Make sure the comment
    // on top of this line is still there.

    batch.changeClassOfService(new MACAddress("1,6,00:11:22:33:44:55")

        // the MACAddress object

        , "Gold");


    // post the batch to the RDU

    BatchStatus batchStatus = null;
    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

これで、加入者はサービス プロバイダーのネットワークにアクセスし、*Gold* サービスを受けられるようになります。

加入者のデバイスの登録解除と削除

サービス プロバイダーは、サービスの利用を中止した加入者を削除する必要があります。

目的

次のワークフローを使用して、加入者のデバイスすべてをサービス プロバイダーのネットワークから完全に削除します。

ステップ 1　サービス プロバイダーのユーザ インターフェイスで、加入者へのサービスを停止します。

ステップ 2　次のステップでは、加入者のデバイスの登録解除方法と削除方法について説明します。サービス プロバイダーによっては、故障の場合を除いてケーブル モデムをデータベースに残す場合があるため、デバイスの削除はオプションです。また、ステップ 2-a に従ってデバイスを登録解除した場合は、ステップ 2-b に従ってデバイスを削除することはできません。

a. デバイスを登録解除するには、サービス プロバイダーのアプリケーションがプロビジョニング クライアント プログラムを使用して、加入者のデバイスすべてのリストを BAC に要求します。次に、各デバイスを登録解除してリセットします。その結果、各デバイスがデフォルトの（プロビジョニングされていない）サービス レベルに低下します。

（注）「unregister」API へのパラメータに指定されたデバイスがすでに登録解除された状態にある場合は、API コールからのステータス コードが `CommandStatusCodes.CMD_ERROR_DEVICE_UNRE- GISTER_UNREGISTERED_ERROR` に設定されます。これは通常および所定の動作です。

```
// MSO admin UI calls the provisioning API to get a list of
// all the subscriber's devices.

// Create a new connection

PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");


// NO_ACTIVATION is the activation mode because this is a query
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device
// NO_PUBLISHING is the publishing mode because we are not attempting
// to publish to external database.

Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,
```

```
        // No publishing to external database

        PublishingMode.NO_PUBLISHING);

    batch.getAllForOwnerID("0123-45-6789"

    //  query all the devices for this account number
    );

    BatchStatus batchStatus = null;
    try
    {
        batchStatus = batch.post();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    CommandStatus commandStatus = batchStatus.getCommandStatus(0);


    //batch success without error, retrieve the result

    RecordSearchResults rcSearchResult = (RecordSearchResults)commandStatus.getData();
    List<RecordData> resultList = rcSearchResult.getRecordData();

    //  We need to unregister all the devices behind each modem(s) or else the
    //  unregister call for that modem will fail.

    if (resultList != null)
    {
      //Unregister the COMPUTER
      for (int i=0; i<resultList.size(); i++)
    {
        RecordData rd = resultList.get(i);
        Map<String, Object> detailMap = rd.getDetails();

        //get the deviceType from the detail map

        String deviceType = (String)detailMap.get(DeviceDetailsKeys.DEVICE_TYPE);


        //only interest in DOCSIS

        if (DeviceType.getDeviceType(deviceType) .equals(DeviceType.COMPUTER))
        {
            Key primaryKey = rd.getPrimaryKey();

            batch = conn.newBatch();
            batch.unregister((DeviceID)primaryKey);

            batchStatus = null;
            try
            {
               batchStatus = batch.post();
            }
            catch(ProvisioningException e)
            {
               e.printStackTrace();
            }
         }
    }
}
    //  for each modem in the retrieved list:

    for (int i=0; i<resultList.size(); i++)
    {
    RecordData rd = resultList.get(i);
    Map<String, Object> detailMap = rd.getDetails();
```

```
    //get the deviceType from the detail map

    String deviceType = (String)detailMap.get(DeviceDetailsKeys.DEVICE_TYPE);


    //only interest in DOCSIS

    if (DeviceType.getDeviceType(deviceType) .equals(DeviceType.DOCSIS))
    {
        Key primaryKey = rd.getPrimaryKey();
        batch = conn.newBatch();
        batch.unregister((DeviceID)primaryKey);
        batchStatus = null;
        try
        {
            batchStatus = batch.post();
        }
        catch(ProvisioningException e)
        {
            e.printStackTrace();
        }
    }
}
```

b. デバイスを削除するには、サービス プロバイダーのアプリケーションがプロビジョニング クライアント プログラムを使用して、加入者の残りのデバイスをデータベースから個別に削除します。

```
// Create a new connection

PACEConnection conn =
    PACEConnectionFactory.getInstance("localhost", 49187, "admin", "admin123");


// NO_ACTIVATION is the activation mode because this is a query
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device
// NO_PUBLISHING is the publishing mode because we are not attempting
// to publish to external database.

Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publisining to external database

    PublishingMode.NO_PUBLISHING);

    batch.getAllForOwnerID("0123-45-6789"   //  query all the devices for this
    account                                      // number
    );

    BatchStatus batchStatus = null;
    try
    {
        batchStatus = batch.post();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
```

```
        CommandStatus commandStatus = batchStatus.getCommandStatus(0);


        //batch success without error, retrieve the result

        RecordSearchResults rcSearchResult =
        (RecordSearchResults)commandStatus.getData();
        List<RecordData> resultList = rcSearchResult.getRecordData();

        if (resultList != null)
        {

        //  for each modem in the retrieved list, delete it

        for (int i=0; i<resultList.size(); i++)
        {
            RecordData rd = resultList.get(i);
            Map<String, Object> detailMap = rd.getDetails();

            //get the deviceType from the detail map

            String deviceType = (String)detailMap.get(DeviceDetailsKeys.DEVICE_TYPE);


            //only interest in DOCSIS

            if (DeviceType.getDeviceType(deviceType) .equals(DeviceType.DOCSIS))
            {
                 Key primaryKey = rd.getPrimaryKey();

                 //change COS

                 batch = conn.newBatch();
                 batch.delete((DeviceID)primaryKey, true);

                 batchStatus = null;
                 try
                {
                     batchStatus = batch.post();
                }
                 catch(ProvisioningException e)
                {
                     e.printStackTrace();
                 }
            }
    }
}
```

無差別モードでの最初のアクティベーションのセルフプロビジョニング

加入者は、戸建住宅内にコンピュータ（ブラウザ アプリケーションがインストール済み）を設置し、DOCSIS ケーブル モデムを購入しました。

目的

次のワークフローを使用して、プロビジョニングされていない新しい DOCSIS ケーブル モデムとコンピュータをオンラインにし、適切なレベルのサービスを受けられるようにします。

ステップ 1　加入者は DOCSIS ケーブル モデムを購入し、住宅内に設置します。

ステップ 2　加入者はモデムの電源をオンにし、BAC は制限付きアクセス権をモデムに付与します。

ステップ 3　加入者はコンピュータのブラウザ アプリケーションを起動します。スプーフィング DNS サーバにより、ブラウザがサービス プロバイダーの登録サーバ（OSS ユーザ インターフェイスやメディエータなど）にアクセスします。

ステップ 4　加入者はサービス プロバイダーのユーザ インターフェイスを使用して、サービス クラスの選択など、登録に必要な手順を終了します。

サービス プロバイダーのユーザ インターフェイスは、加入者の情報（選択されたサービス クラスやコンピュータの IP アドレスなど）を BAC に渡します。加入者のケーブル モデムとコンピュータが BAC に登録されます。

ステップ 5　ユーザ インターフェイスは、加入者にコンピュータをリブートするように求めます。

```
// Create a new connection
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");


Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);


// NO_ACTIVATION is the activation mode because this is a
// query. NO_CONFIRMATION is the confirmation mode because
// we are not attempting to reset the device.
// First we query the computer's information to find the
// modem’s MAC address.
// We use the computer’s IP address (the web browser
// received this when the subscriber opened the service
// provider’s web interface).
// We also assume that "bostonProvGroup"
// is the provisioning group used in that locality.

List<String> provGroupList = new ArrayList<String>();

provGroupList.add("bostonProvGroup");


batch.getAllForIPAddress("10.0.14.38",

    // ipAddress: restricted access computer lease
    provGroupList
    // provGroups: List containing provgroup
    );

BatchStatus batchStatus = null;

// post the batch to RDU server

try
{
    batchStatus = batch.post();
}
```

```
catch(ProvisioningException e)
{
    e.printStackTrace();
}

// Get the LeaseResults object after posting a batch.

CommandStatus commandStatus = batchStatus.getCommandStatus(0);

LeaseResults computerLease = (LeaseResults)commandStatus.getData();

// Derive the modem MAC address from the computer's network
// information. The 1,6, is a standard prefix for an Ethernet
// device. The fully qualified MAC address is required by BACC

StringBuffer modemMACAddress = new StringBuffer();
modemMACAddress.append("1,6,");
modemMACAddress.append(computerLease.getSingleLease().get("relay-agent-remote-id"))
;


//create MacAddress object from the string

MACAddress modemMACAddressObject = new MACAddress(modemMACAddress.toString());

List<DeviceID> modemDeviceIDList = new ArrayList<DeviceID>();
modemDeviceIDList.add(modemMACAddressObject);


// NO_ACTIVATION is the activation mode because this is a query
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device
// NO_PUBLISHING is the publishing mode because we are not attempting
// to publish to external database.

batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

Map<String, Object> properties = new HashMap<String, Object>();

// Set the property PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED
// to enable promiscuous mode on modem

properties.put(PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED, Boolean.TRUE);

properties.put(PolicyKeys.COMPUTER_DHCP_CRITERIA, "provisionedCPE");


// enable promiscuous mode by changing the technology default

batch.changeDefaults(DeviceType.DOCSIS,
    properties, null);


// post the batch to RDU server

try
{
```

```
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
    batch = conn.newBatch(

        // No reset

        ActivationMode.NO_ACTIVATION,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION,

        // No publishing to external database

        PublishingMode.NO_PUBLISHING);


    batch.add(
        DeviceType.DOCSIS,     // deviceType: DOCSIS
        modemDeviceIDList,     // macAddress: derived from computer lease
        null,                  // hostName: not used in this example
        null,                  // domainName: not used in this example
        "0123-45-6789",        // ownerID: here, account number from billing system
        "Silver",              // ClassOfService
        "provisionedCM",       // DHCP Criteria: Network Registrar uses this to
                               // select a modem lease granting provisioned IP address
        null                   // properties:
    );
```

ステップ 6　プロビジョニング クライアントは *performOperation(...)* を呼び出してモデムをリブートし、プロビジョニングされたアクセス権をモデムに付与します。

```
    // Reset the computer
    // create a new batch
    batch = conn.newBatch(

        // No reset

        ActivationMode.AUTOMATIC,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION);

    // register performOperation command to the batch

    batch.performOperation(DeviceOperation.RESET,
        modemMACAddressObject, null);

    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 7　コンピュータをリブートすると、コンピュータは新しい IP アドレスを受信します。

これで、ケーブル モデムはプロビジョニングされたデバイスとなります。コンピュータは BAC に登録されていませんが、サービス プロバイダーのネットワークを介してインターネットにアクセスできます。無差別モードのモデムの背後でオンラインになっているコンピュータは、引き続き、プロビジョニング API を使用して利用することができます。

無差別モードでの 100 台のモデムの一括プロビジョニング

サービス プロバイダーのカスタマー サービス担当者が、サービス センターで、配送する 100 台のケーブル モデムを事前プロビジョニングします。

目的

次のワークフローを使用して、すべてのモデムのモデム データを新しい加入者に配送します。カスタマー サービス担当者は、割り当て可能なモデムのリストを持っています。

ステップ 1　サービス プロバイダーの発送センターで、新しいケーブル モデムまたは再利用するケーブル モデムの MAC アドレス データがリストにまとめられます。

ステップ 2　特定のサービス センターに割り当てられたモデムが BAC に一括ロードされ、そのサービス センターの識別名が付けられます。

ステップ 3　サービス　センターでモデムを新しい加入者に配送する際、カスタマー　サービス担当者が新しいサービス パラメータを入力し、モデムの Owner ID フィールドを新しい加入者のアカウント番号に合せて変更します。

```
// Create a new connection
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");


Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);



// The activation mode for this batch should be NO_ACTIVATION.
// NO_ACTIVATION should be used in this situation because no
// network information exists for the devices because they
// have not booted yet. A configuration can't be generated if no
// network information is present. And because the devices
// have not booted, they are not online and therefore cannot
// be reset. NO_CONFIRMATION is the confirmation mode because
// we are not attempting to reset the devices.
// Create a Map for the properties of the modem
```

```
Map properties;

// Set the property PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED to
// enable promiscuous mode on modem.
// This could be done at a system level if promiscuous mode
// is your default provisioning mode.

properties.put(PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED, Boolean.TRUE);

// The PolicyKeys.CPE_DHCP_CRITERIA is used to specify the DHCP
// Criteria to be used while selecting IP address scopes for
// CPE behind this modem in the promiscuous mode.

properties.put(PolicyKeys.COMPUTER_DHCP_CRITERIA, "provisionedCPE");

// enable promiscuous mode by changing the technology default

batch.changeDefaults(DeviceType.DOCSIS,properties, null);

BatchStatus batchStatus = null;

// post the batch to RDU server

try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
{
    e.printStackTrace();
        }

// for each modem MAC-address in list:

ModemLoop:
{
    batch = conn.newBatch(

       // No reset

       ActivationMode.NO_ACTIVATION,

       // No need to confirm activation

       ConfirmationMode.NO_CONFIRMATION,

       // No publishing to external database

       PublishingMode.NO_PUBLISHING);


batch.add(
    DeviceType.DOCSIS,   // deviceType: DOCSIS
    modemMACAddressList, // modemMACAddressList: the list of deviceID
    null,                // hostName: not used in this example
    null,                // domainName: not used in this example
    "0123-45-6789",      // ownerID: here, account number from billing system
    "Silver",            // ClassOfService
    "provisionedCM",     // DHCP Criteria: Network Registrar uses this to
                         // select a modem lease granting provisioned IP address
    properties           // properties:
);

try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
```

```
    {
        e.printStackTrace();
    }
    // end ModemLoop.
}
```

無差別モードでの最初のアクティベーションの事前プロビジョニング

新しい加入者はサービス プロバイダーに連絡し、サービスを要求します。加入者は、戸建住宅内にコンピュータを設置しています。

目的

次のワークフローを使用して、プロビジョニングされていない新しいケーブル モデムとコンピュータをオンラインにし、適切なレベルのサービスを受けられるようにします。

ステップ 1　サービス　プロバイダーは、課金システムで使用される加入者のユーザ名とパスワードを選択します。

ステップ 2　サービス プロバイダーは、加入者がアクセスできるサービスを選択します。

ステップ 3　サービス プロバイダーは、独自のユーザ インターフェイスを使用して、デバイスを登録します。

ステップ 4　サービス プロバイダーのユーザ インターフェイスは、モデムの MAC アドレスやサービス クラスなどの情報を BAC に渡します。また、モデムが CPE DHCP 基準設定を取得します。この基準設定により、Network Registrar はモデムの背後に接続するコンピュータに対して、プロビジョニングされたアドレスを選択できます。次に、新しいモデムが BAC に登録されます。

ステップ 5　サービス プロバイダーの現場技術者は、新しい加入者の住宅内に物理ケーブルを敷設し、事前プロビジョニングされたデバイスを設置して、加入者のコンピュータに接続します。

```
//  MSO admin UI calls the provisioning API to pre-provision
//  an HSD modem.

// Create a new connection
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");


Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);


// The activation mode for this batch should be NO_ACTIVATION.
// NO_ACTIVATION should be used in this situation because no
// network information exists for the modem because it has not
```

```
// booted. A configuration cannot be generated if no network
// information is present. And because the modem has not booted,
// it is not online and therefore cannot be reset.
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the modem.
// Create a map for the properties of the modem.


Map<String, Object> properties = new HashMap<String, Object>();

// Set the property PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED
// to enable promiscuous mode on modem

properties.put(PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED, Boolean.TRUE);

properties.put(PolicyKeys.COMPUTER_DHCP_CRITERIA, "provisionedCPE");


// enable promiscuous mode by changing the technology default

batch.changeDefaults(DeviceType.DOCSIS, properties, null);

BatchStatus batchStatus = null;

// post the batch to RDU server

try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
{
    e.printStackTrace();
}
batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);


MACAddress macAddressObject = new MACAddress("1,6,00:11:22:33:44:55");
List<DeviceID> modemDeviceIDList = new ArrayList<DeviceID>();
modemDeviceIDList.add(macAddressObject);

batch.add(
    DeviceType.DOCSIS,      // deviceType: DOCSIS
    modemDeviceIDList,      // macAddress: derived from computer lease
    null,                   // hostName: not used in this example
    null,                   // domainName: not used in this example
    "0123-45-6789",      // ownerID: here, account number from billing system
    "Silver",            // ClassOfService
    "provisionedCM",     // DHCP Criteria: Network Registrar uses this to
                         // select a modem lease granting provisioned IP address
    null                 // properties:
);


// post the batch to RDU server

try
```

```
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 6　ケーブル モデムの電源をオンにすると、BAC はプロビジョニングされたアクセス権をケーブル モデムに付与します。

ステップ 7　コンピュータの電源をオンにすると、BAC はプロビジョニングされたアクセス権をコンピュータに付与します。

これで、ケーブル モデムとコンピュータはいずれもプロビジョニングされたデバイスとなります。コンピュータは、サービス プロバイダーのネットワークを介してインターネットにアクセスできます。

既存のモデムの交換

サービス プロバイダーは、故障したモデムを交換します。

（注）　モデムから別のモデムへのローミングを制限するオプションがコンピュータに設定されている場合、モデムを交換したときは、コンピュータに設定されているモデムの MAC アドレスも変更する必要があります。

目的

次のワークフローを使用して、加入者に提供するサービスのレベルを変更することなく、既存のケーブル モデムを新しいモデムに物理的に交換します。

ステップ 1　サービス プロバイダーは、既存のモデムの MAC アドレスを新しいモデムの MAC アドレスに変更します。

```
// Create a new connection
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");

batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database
```

```
        PublishingMode.NO_PUBLISHING);


    MACAddress macAddressObject = new MACAddress("1,6,00:11:22:33:44:55");
    List<DeviceID> modemDeviceIDList = new ArrayList<DeviceID>();
    modemDeviceIDList.add(macAddressObject);

    //  old macAddress: unique identifier for the old modem
    MACAddress oldMacAddress = new MACAddress("1,6,00:11:22:33:44:55");

    // new macAddress: unique identifier for the new modem
    MACAddress newMacAddress = new MACAddress("1,6,00:11:22:33:44:66");
    List<DeviceID> newDeviceIDs = new ArrayList<DeviceID>();
    newDeviceIDs.add(newMacAddress);

    batch.changeDeviceID(oldMacAddress, newDeviceIDs);

    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 2　サービス プロバイダーは、ケーブル モデムを交換し、その電源をオンにします。

ステップ 3　コンピュータの電源もオンにする必要があります。

これで、ケーブル モデムは完全にプロビジョニングされたデバイスとなり、適切なレベルのサービスを受けられるようになります。ケーブル モデムの背後にあるコンピュータも同様です。

無差別モードでの 2 台目のコンピュータの追加

加入者は、設置されているケーブル モデムの背後に 2 台目のコンピュータを接続します。この例では、プロビジョニング API の呼び出しは必要ありません。

目的

次のワークフローを使用して、加入者が選択したサービスが複数の CPE セットの接続を許可すること、および接続された両方のコンピュータから加入者がネットワークにアクセスできることを確認します。

ステップ 1　加入者はケーブル モデムの背後に 2 台目のコンピュータを接続します。

ステップ 2　加入者はコンピュータの電源をオンにします。

加入者が選択したサービスが複数の CPE セットの接続を許可している場合、BAC はインターネットへのアクセス権を 2 台目のコンピュータに付与します。

NAT を使用した最初のアクティベーションのセルフプロビジョニング

大学で、Network Address Translation（NAT; ネットワークアドレス変換）および DHCP 機能を持つ DOCSIS ケーブル モデムが購入されました。建物の利用者 5 人はそれぞれ、ブラウザ アプリケーションを持つコンピュータを設置しています。

目的

次のワークフローを使用して、プロビジョニングされていない新しいケーブル モデム（NAT を持つ）と、モデムの背後にあるコンピュータをオンラインにし、適切なレベルのサービスを受けられるようにします。

ステップ 1　加入者は NAT および DHCP 機能を持つケーブル モデムを購入し、集合住宅内に設置します。

ステップ 2　加入者はモデムの電源をオンにし、BAC は制限付きアクセス権をモデムに付与します。

ステップ 3　加入者はラップトップ コンピュータをケーブル モデムに接続し、モデム内の DHCP サーバは IP アドレスをラップトップに付与します。

ステップ 4　加入者はコンピュータのブラウザ アプリケーションを起動します。スプーフィング DNS サーバにより、ブラウザがサービス プロバイダーの登録サーバ（OSS ユーザ インターフェイスやメディエータなど）にアクセスします。

ステップ 5　加入者はサービス プロバイダーのユーザ インターフェイスを使用して、モデムのケーブル モデムの登録に必要な手順を終了します。登録用のユーザ インターフェイスは、モデムで NAT が使用されていることを検出し、モデムを登録して、モデムが NAT 互換のサービス クラスを受けられることを確認します。詳細については、P.D-6 の「固定標準モードでセルフプロビジョニングされたモデムとコンピュータ」を参照してください。

（注）　NAT を持つ特定のケーブル モデムでは、新しいサービス クラス設定を取得するためにコンピュータをリブートするように求められる場合があります。ケーブル モデムと NAT デバイスが別々のデバイスである場合は、NAT デバイスもコンピュータの登録と同じように登録する必要があります。

NAT を持つモデムの背後への新しいコンピュータの追加

アパートには 4 人の借家人がいて、モデムを共有し、サービス プロバイダーのネットワークにアクセスしています。このアパートの大家が新しい借家人に対し、建物のモデムを共有したインターネット アクセスを提供します。モデムには NAT および DHCP 機能が含まれています。新しい借家人は、自分のコンピュータをモデムに接続しています。

（注）　この例では、プロビジョニング API の呼び出しは必要ありません。

目的

次のワークフローを使用して、プロビジョニングされていない新しいコンピュータを、すでにプロビジョニングされているケーブル モデムを使用してオンラインにし、新しいコンピュータが適切なレベルのサービスを受けられるようにします。

ステップ 1　加入者はコンピュータの電源をオンにします。

ステップ 2　これで、コンピュータはプロビジョニングされたデバイスとなり、適切なレベルのサービスにアクセスできます。

プロビジョニングされた NAT モデムは、モデムの背後にあるコンピュータをネットワーク上で非表示にします。

別の DHCP スコープへのデバイスの移動

サービス プロバイダーは、ネットワークに番号を再割り当てします。これに伴い、登録済みケーブル モデムに、別の Network Registrar スコープの IP アドレスが必要になります。

目的

プロビジョニング クライアントは DHCP 基準を変更し、ケーブル モデムは対応する DHCP スコープから IP アドレスを受信します。

ステップ 1　DOCSIS モデムの DHCP 基準を「newmodemCriteria」に変更します。

```
    // Create a new connection
    PACEConnection conn = PACEConnectionFactory.getInstance(
        "localhost", 49187, "admin", "admin123");

    Batch batch = conn.newBatch(

        // No reset

        ActivationMode.AUTOMATIC,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION,

        // No publishing to external database

        PublishingMode.NO_PUBLISHING);

    // AUTOMATIC is the Activation mode because we are attempting
    // to reset the modem so that a phone line is disabled
    // NO_CONFIRMATION is the Confirmation mode because we don't
    // want the batch to fail if we can't reset the modem.
    // This use case assumes that the DOCSIS modem has been
    // previously added to the database

    batch.changeDHCPCriteria(
        new MACAddress("1,6,ff:00:ee:11:dd:22"),  // Modem's MAC address or FQDN
              "newmodemCriteria"
    );

    // post the batch to RDU server

    BatchStatus batchStatus = null;
    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 2　モデムは、「newmodemCriteria」によって対象とされたスコープから IP アドレスを取得します。

イベントを使用したデバイス削除のロギング

サービス プロバイダーは、複数のプロビジョニング クライアントを保有しており、デバイス削除をログに記録します。

目的

どのプロビジョニング クライアントがデバイスを削除しても、プロビジョニング クライアントはイベントを 1 か所のログに記録します。

ステップ 1　デバイス削除イベントのリスナーを作成します。このクラスは、*DeviceAdapter* 抽象クラスを拡張するか、または *DeviceListener* インターフェイスを実装する必要があります。また、イベントをログに記録するために *deletedDevice(DeviceEvent ev)* メソッドを無効にする必要もあります。

```
public DeviceDeletionLogger
        extends DeviceAdapter

        //Extend the DeviceAdapter class.
{
    public void deletedDevice(DeviceEvent ev)

    //Override deletedDevice.
    {
        logDeviceDeletion(ev.getDeviceID());

        //Log the deletion.
    }
}
```

ステップ 2　*PACEConnection* インターフェイスを使用して、イベントのリスナーと修飾子を登録します。

```
DeviceDeletionLogger deviceDeletionLogger =
    new DeviceDeletionLogger();

        // Modem's MAC address or FQDN "newmodemCriteria"

DeviceEventQualifier qualifier = new DeviceEventQualifier();

// We are interested only in device deletion.

qualifier.setDeletedDevice ();

// Add device listener using PACEConnection

connection.addDeviceListener(deviceDeletionLogger, qualifier
);
```

ステップ 3　システムからデバイスが削除されると、イベントが生成され、リスナーに通知されます。

イベントを使用した RDU 接続の監視

サービス プロバイダーは 1 つのプロビジョニング クライアントを実行しており、プロビジョニング クライアントと RDU 間の接続が切断された場合に通知されるようにします。

目的

次のワークフローを使用して、接続が切断された場合にイベント インターフェイスからサービス プロバイダーに通知されるように設定します。

ステップ 1　メッセージ イベントのリスナーを作成します。このクラスは、*MessagingAdapter* 抽象クラスを拡張するか、または *MessagingListener* インターフェイスを実装する必要があります。さらに、このクラスは *connectionStopped(MessagingEvent ev)* メソッドを無効にする必要もあります。

```
// Extend the service provider's Java program using the
// provisioning client to receive Messaging events.
public MessagingNotifier
    extends MessagingAdapter

    //Extend the MessagingAdapter class.
{
    public void connectionStopped(MessagingEvent ev)

    //Override connectionStopped.
    {
        doNotification(ev.getAddress(), ev.getPort());

        //Do the notification.
    }
}
```

ステップ 2　*PACEConnection* インターフェイスを使用して、イベントのリスナーと修飾子を登録します。

```
MessagingQualifier qualifier =new MessagingQualifier();
qualifier.setConnectionDown();
MessagingNotifier messagingNotifier = new MessagingNotifier();
connnection.addMessagingListener(messagingNotifier, qualifier
);
```

ステップ 3　接続が切断されると、イベントが生成され、リスナーに通知されます。接続が中断した場合は常に、PACEConnection が自動的に RDU に再接続します。

イベントを使用したバッチ完了のロギング

サービス プロバイダーは、複数のプロビジョニング クライアントを保有しており、バッチ完了をログに記録します。

目的

どのプロビジョニング クライアントがバッチを完了しても、イベントを 1 か所のログに記録します。

ステップ 1　イベントのリスナーを作成します。このクラスは、*BatchAdapter* 抽象クラスを拡張するか、または *BatchListener* インターフェイスを実装する必要があります。また、イベントをログに記録するために *completion(BatchEvent ev)* メソッドを無効にする必要があります。

```
public BatchCompletionLogger
    extends BatchAdapter

    //Extend the BatchAdapterclass.
{
    public void completion(BatchEvent ev)
    //Override completion.
    {
        logBatchCompletion(ev.BatchStatus().getBatchID());
        //Log the completion.
    }
}
```

ステップ 2　*PACEConnection* インターフェイスを使用して、イベントのリスナーと修飾子を登録します。

```
BatchCompletionLogger batchCompletionLogger = new BatchCompletionLogger();
BatchEventQualifier qualifier = new BatchEventQualifier();
connection.addBatchListener(batchCompletionLogger , qualifier
);
```

ステップ 3　バッチが完了すると、イベントが生成され、リスナーに通知されます。

デバイスの詳細情報の取得

サービス プロバイダーは管理者に対して、特定のデバイスの詳細情報を表示することを許可します。

目的

サービス プロバイダーの管理アプリケーションは、特定のデバイスに関する既知の詳細をすべて表示します。この詳細には、MAC アドレス、リース情報、デバイスのプロビジョニング ステータス、およびデバイス タイプ（既知の場合）などがあります。

ステップ 1　管理者は、サービス プロバイダーの管理者のユーザ インターフェイスに、クエリーするデバイスの MAC アドレスを入力します。

ステップ 2　BAC は、デバイスの詳細を組み込みデータベースにクエリーします。

```
// The host name or IP address of the RDU. It is
// recommended that you normally use a fully-qualified domain name
// since it lends itself to the greatest flexibility going forward.
// For example, you could change the host running RDU without
// having to reassign IPs. For that reason, having an alias for
// the machine is better than a specific name.

final String rduHost = "localhost";

// The port number of RDU on the server.

final int rduPort = 49187;

// The user name for connecting to RDU.

final String userName = "admin";

// The password to use with the username.

final String password = "changeme";

// ------------------------------------------------------------------------
// DEVICE PARAMETERS, see IPDevice.getDetails()
// ------------------------------------------------------------------------

// The MAC address of the modem to be queried. MAC addresses in BAC
// must follow the simple "1,6,XX:XX:XX:XX:XX:XX" format.

final DeviceID modemMACAddress = DeviceID.getInstance("1,6,00:11:22:33:44:55",
    KeyType.MAC_ADDRESS);

// The PACE connection to use throughout the example. When
// executing multiple batches in a single process, it is advisable
// to use a single PACE connection that is retrieved at the start
// of the application. When done with the connection, YOU MUST
// explicitly close the connection with the releaseConnection()
// method call.

PACEConnection connection = null;

// 1) Connect to the Regional Distribution Unit (RDU).
//
// The parameters defined at the beginning of this class are
// used here to establish the connection. Connections are
// maintained until releaseConnection() is called. If
// multiple calls to getInstance() are called with the same
// arguments, you must still call releaseConnection() on each
// connection you received.
//
// The call can fail for one of the following reasons:
// - The hostname / port is incorrect.
// - The authentication credentials are invalid.

try
{
connection = PACEConnectionFactory.getInstance(
// RDU host   rduHost,
// RDU port   rduPort,
// User name  userName,
// Password   password
);
}
catch (PACEConnectionException e)
{
// failed to get a connection
```

```
System.out.println("Failed to establish a PACEConnection to [" +
    userName + "@" + rduHost + ":" + rduPort + "]; " +
    e.getMessage());

System.exit(1);
}
// 2) Create a new batch instance.
//
// To perform any operations in the Provisioning API, you must
// first start a batch. As you make commands against the batch,
// nothing will actually start until you post the batch.
// Multiple batches can be started concurrently against a
// single connection to the RDU.

Batch myBatch = connection.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// 3) Register the getDetails(...) with the batch.

// Use the Provisioning API to get all of the information for
// the specified MAC address. Since methods aren't actually
// executed until the batch is posted, the results are not
// returned until after post() completes. The getCommandStatus()
// followed by getData() calls must be used to access the results
// once the batch is posted.

final DeviceID modemMACAddress = DeviceID.getInstance("1,6,00:11:22:33:44:55",
        KeyType.MAC_ADDRESS);

List options = new ArrayList();
     options.add(DeviceDetailsOption.INCLUDE_LEASE_INFO);

myBatch.getDetails(modemMACAddress, options);

// 4) Post the batch to the server.
//
// Executes the batch against the RDU. All of the
// methods are executed in the order entered and the data
// changes are applied against the embedded database in RDU.

BatchStatus bStatus = null;
try
{
    bStatus = myBatch.post();
}
catch (ProvisioningException pe)
{
System.out.println("Failed to query for modem with MAC address [" +
    modemMACAddress + "]; " + pe.getMessage());

System.exit(2);
}
// 5) Check to see if the batch was successfully posted.
//
// Verify if any errors occurred during the execution of the
// batch. Exceptions occur during post() for truly exception
// situations such as failure of connectivity to RDU.
// Batch errors occur for inconsistencies such as no lease
```

```
// information for a device requiring activation. Command
// errors occur when a particular method has problems, such as
// trying to add a device that already exists.

if (bStatus.isError())
{
// Batch error occurred.

System.out.println("Failed to query for modem with MAC address [" +
    modemMACAddress + "]; " + bStatus.getErrorMessage());

System.exit(3);
}
```

ステップ 3　サービス プロバイダーのアプリケーションは、デバイス データの詳細に関するページを表示します。このページには、要求されたデバイスに関する既知の情報をすべて表示できます。デバイスがサービス プロバイダーのネットワークに接続された場合、このデータにはリース情報（IP アドレスやリレー エージェントの ID など）が含まれます。データは、デバイスがプロビジョニングされたかどうかを示します。プロビジョニングされた場合、データにはデバイス タイプも含まれます。

```
// Successfully queried for device.
System.out.println("Queried for DOCSIS modem with MAC address ["+
    modemMACAddress + "]");

// Display the results of the command (TreeMap is sorted). The
// data returned from the batch call is stored on a per-command
// basis. In this example, there is only one command, but if
// you had multiple commands all possibly returning results, you
// could access each result by the index of when it was added.
// The first method added is always index 0. From the status of
// each command, you can then access the accompanying data by
// using the getData() call. Since methods can return data of
// different types, you will have to cast the response to the
// type indicated in the Provisioning API documentation.

Map<String, Object> deviceDetails = new HashMap<String,
    Object>((Map)bStatus.getCommandStatus(0).getData());

String deviceType = (String)deviceDetails.get(DeviceDetailsKeys.DEVICE_TYPE);
String macAddress = (String)deviceDetails.get(DeviceDetailsKeys.MAC_ADDRESS);
String fqdn = (String)deviceDetails.get(DeviceDetailsKeys.FQDN);
String duid = (String)deviceDetails.get(DeviceDetailsKeys.DUID);
String host = (String)deviceDetails.get(DeviceDetailsKeys.HOST);
String domain = (String)deviceDetails.get(DeviceDetailsKeys.DOMAIN);

// if the device is DocsisModem, get the COS

String cos   = (String)deviceDetails.get(DeviceDetailsKeys.CLASS_OF_SERVICE);
String dhcpCriteria   = (String)deviceDetails.get(DeviceDetailsKeys.DHCP_CRITERIA);
String provGroup = (String)deviceDetails.get(DeviceDetailsKeys.PROV_GROUP);
Boolean isProvisioned =
(Boolean)deviceDetails.get(DeviceDetailsKeys.IS_PROVISIONED);
String ownerID = (String)deviceDetails.get(DeviceDetailsKeys.OWNER_ID);
Boolean isRegistered = (Boolean)deviceDetails.get(DeviceDetailsKeys.IS_REGISTERED);
String oidNumber =
(String)deviceDetails.get(GenericObjectKeys.OID_REVISION_NUMBER);

// if the device is a modem, get the device behind

String relayAgentMacAddress =
    (String)deviceDetails.get(DeviceDetailsKeys.RELAY_AGENT_MAC);
String relayAgentDUID =
(String)deviceDetails.get(DeviceDetailsKeys.RELAY_AGENT_DUID);

// get the map of Device property
```

```
Map deviceProperties = (Map)deviceDetails.get(DeviceDetailsKeys.PROPERTIES);

// get the map of discovery data v4

Map dhcpdiscovermapv4 =
    (Map)deviceDetails.get(DeviceDetailsKeys.DISCOVERED_DATA_DHCPV4);

// if discovery data is not null, get the inform, response, request and environment
// map from discovery data map

Map dhcpInformMap = (Map)dhcpdiscovermapv4.get("INFORM");
Map dhcpRespMap = (Map)dhcpdiscovermapv4.get("RESPONSE");
Map dhcpReqMap = (Map)dhcpdiscovermapv4.get("REQUEST");
Map dhcpEnvMap = (Map)dhcpdiscovermapv4.get("ENVIRONMENT");

// get the map of lease query v4

Map leasemapv4 = (Map)deviceDetails.get(DeviceDetailsKeys.LEASE_QUERY_DATA_DHCPV4);
String leaseTime = (String)leasemapv4.get(CNRNames.DHCP_LEASE_TIME.toString());
String rebindingTime =
    (String)leasemapv4.get(CNRNames.DHCP_REBINDING_TIME.toString());

String clientLastTransTime =
    (String)leasemapv4.get(CNRNames.CLIENT_LAST_TRANSACTION_TIME.toString());
String clientIPAddress=
(String)leasemapv4.get(CNRNames.CLIENT_IPADDRESS.toString());
String relayAgentRemoteID=
    (String)leasemapv4.get(CNRNames.RELAY_AGENT_REMOTE_ID.toString());
String relayAgentCircuitID=
    (String)leasemapv4.get(CNRNames.RELAY_AGENT_CIRCUIT_ID.toString());

// get the map of discovery DHCP v6

Map dhcpdiscovermapv6 =
    (Map)deviceDetails.get(DeviceDetailsKeys.DISCOVERED_DATA_DHCPV6);

// if discovery data is not null , get the inform, response, request and
environment
// map from discovery data map

Map dhcpv6InformMap = (Map)dhcpdiscovermapv6.get("INFORM");

Map dhcpv6RespMap = (Map)dhcpdiscovermapv6.get("RESPONSE");

Map dhcpv6ReqMap = (Map)dhcpdiscovermapv6.get("REQUEST");

Map dhcpv6RelReqMap = (Map)dhcpdiscovermapv6.get("RELAY_REQUEST");

Map dhcpv6EnvMap = (Map)dhcpdiscovermapv6.get("ENVIRONMENT");

// get the map of lease query V6

Map leasemapv6 = (Map)deviceDetails.get(DeviceDetailsKeys.LEASE_QUERY_DATA_DHCPV6);

String iaprefixkey = (String)leasemapv6.get(CNRNames.IAPREFIX.toString());
String iaaddrkey = (String)leasemapv6.get(CNRNames.IAADDR.toString());
String leasetimev6 = (String)leasemapv6.get(CNRNames.VALID_LIFETIME.toString());
String renewaltimev6 =
(String)leasemapv6.get(CNRNames.PREFERRED_LIFETIME.toString());
String dhcplasttranstimev6 =
    (String)leasemapv6.get(CNRNames.CLIENT_LAST_TRANSACTION_TIME);
String clientIpAddressv6 = (String)leasemapv6.get(CNRNames.CLIENT_IPADDRESS);
String relayagentremoteidv6 =
(String)leasemapv6.get(CNRNames.RELAY_AGENT_REMOTE_ID);
String relayagentcircuitidv6 =
    (String)leasemapv6.get(CNRNames.RELAY_AGENT_CIRCUIT_ID);
```

デバイス タイプを使用した検索

サービス プロバイダーは管理者に対して、すべての DOCSIS モデムのデータを表示することを許可します。

目的

サービス プロバイダーの管理アプリケーションは、DOCSIS デバイスのリストを返します。

ステップ 1　管理者は、サービス プロバイダーの管理者のユーザ インターフェイスで、検索オプションを選択します。

ステップ 2　BAC は、DOCSIS モデムの MAC アドレスすべてのリストを、組み込みデータベースにクエリーします。

```
public static void getAllDevicesByDeviceType() throws Exception  {
      DeviceSearchType dst = DeviceSearchType.getByDeviceType(
          DeviceType.getDeviceType(DeviceTypeValues.DOCSIS_MODEM),
          ReturnParameters.ALL);

      RecordSearchResults rs = null;

      SearchBookmark sb = null;

      rs = searchDevice(dst, sb);
      sb = rs.getSearchBookmark();

      while (sb != null)
      {
          // print out the data in the record search result.
          sb = printRecordSearchResults(rs);

          // call the search routine again
          rs = searchDevice(dst, sb);
      }
 }
 private static RecordSearchResults searchDevice(DeviceSearchType dst,
                SearchBookmark sb) throws Exception  {
      RecordSearchResults rs = null;
      final Batch batch = s_conn.newBatch();
      final int numberOfRecordReturn = 10;

      //calling the search API
      batch.searchDevice(dst, sb, numberOfRecordReturn);

      // Call the RDU.
      BatchStatus batchStatus = batch.post();

      // Check for success.
      CommandStatus commandStatus = null;

      if (0 < batchStatus.getCommandCount())
      {
          commandStatus = batchStatus.getCommandStatus(0);
      }
      //check to see if there is an error
      if (batchStatus.isError()
      || batchStatus.isWarning()
      || commandStatus == null
      || commandStatus.isError())
      {
          System.out.println("report batch error.");
          return null;
      }
```

```
      //batch success without error, retrieve the result
      //this is a list of devices
      rs = (RecordSearchResults)commandStatus.getData();
      return rs;
 }


 private static SearchBookmark printRecordSearchResults(RecordSearchResults rs)
throws Exception  {

      SearchBookmark sb = rs.getSearchBookmark();

      List<RecordData> rdlist = rs.getRecordData();
      Iterator<RecordData> iter = rdlist.iterator();

      while (iter.hasNext())
      {
          RecordData rdObj = iter.next();
          Key keyObj = rdObj.getPrimaryKey();

           System.out.println("DeviceOID: " + ((DeviceID)keyObj).getDeviceId());

          //this is for secondary keys.
          List<Key> deviceList = rdObj.getSecondaryKeys();

          if (deviceList != null && !deviceList.isEmpty())
          {
              for (int i=0; i<deviceList.size(); i++)
              {
                  Key key = deviceList.get(i);
                  System.out.println("DeviceID : " + key.toString());
              }
          }
      }
      return sb;

 }
```

ベンダー プレフィックスまたはサービス クラスを使用したデバイスの検索

サービス プロバイダーは管理者に対して、特定のベンダー プレフィックスまたはサービス クラスに一致するすべてのデバイスを検索することを許可します。

目的

サービス プロバイダーの管理アプリケーションは、要求されたベンダー プレフィックスまたはサービス クラスに一致するデバイスのリストを返します。

ステップ 1　管理者は、サービス プロバイダーの管理者のユーザ インターフェイスに、目的のベンダー プレフィックスに一致する部分文字列を入力します。

ステップ 2　BAC は、要求されたベンダー プレフィックスまたはサービス クラスに一致するデバイスの MAC アドレスすべてのリストを、組み込みデータベースにクエリーします。この例では、検索クエリーを組み立て、MAC アドレスを使用してデバイスを取得する方法を示します。P.D-40 の「デバイス タイプを使用した検索」も参照してください。

```
DeviceIDPattern pattern = new MACAddressPattern("1,6,22:49:*");

DeviceSearchType dst = DeviceSearchType.getDevices(pattern, ReturnParameters.ALL);

// To set up search for class of service:

DeviceSearchType searchType = DeviceSearchType.getByClassOfService(

new ClassOfServiceName(name), AssociationType

.valueOf(association), ReturnParameters.ALL);
```

ステップ 3　サービス プロバイダーのアプリケーションは、これらのデバイスの詳細を BAC に要求し、デバイス データのページを表示します。デバイスごとに、デバイス タイプ、MAC アドレス、クライアント クラス、およびデバイスのプロビジョニング ステータスが表示されます。1 行にデバイスが 1 つ表示されます。

```
// calling the search procedure

rs = searchDevice(connection, dst, sb);
sb = processRecordSearchResults(rs);

if (rs != null)
{
    while (sb != null)
         {
            // The search returns a search bookmark, which can be used to make
            // the next search call that would return next set of results

            rs = searchDevice(connection, dst, sb);
            sb = processRecordSearchResults(rs);
        }
    }
}
}
```

PacketCable eMTA の事前プロビジョニング

新しい顧客は、サービス プロバイダーに連絡して PacketCable 音声サービスを注文します。顧客は、プロビジョニングされた組み込み型 MTA を受け取ります。

目的

次のワークフローを使用して、組み込み型 MTA を事前プロビジョニングし、モデムの MTA コンポーネントがオンラインになったときに適切なレベルのサービスを受けられるようにします。

（注）　次の使用例では、eMTA から電話をかける際に必要なコール エージェントのプロビジョニングを省略しています。

ステップ 1　サービス　プロバイダーは、課金システムで使用される加入者のユーザ名とパスワードを選択します。

ステップ 2　サービス プロバイダーは、モデム コンポーネントに適切なサービス クラスと DHCP 基準を選択し、そのコンポーネントを BAC に追加します。

```
// Create a new connection

PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");


Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION);

// Let's provision the modem and the MTA component in the same
// batch. This can be done because the activation mode of this
// batch is NO_ACTIVATION. More than one device can be operated
// on in a batch if the activation mode does not lead to more
// than one device being reset.
// To add a DOCSIS modem:

List<DeviceID> modemDeviceIDList = new ArrayList<DeviceID>();
modemDeviceIDList.add(new MACAddress("1,6,01:02:03:04:05:06"));
batch.add(
    DeviceType.DOCSIS,      // deviceType: DOCSIS
    modemDeviceIDList,      // macAddress: scanned from the label
    null,                   // hostName: not used in this example
    null,                   // domainName: not used in this example
    "0123-45-6789",         // ownerID: here, account number from billing system
    "Silver",               // classOfService
    "provisionedCM",        // DHCP Criteria: Network Registrar uses this to
                            // select a modem lease granting provisioned IP address
    null                    // properties: not used
);
```

ステップ 3　サービス プロバイダーは、MTA コンポーネントに適切なサービス クラスと DHCP 基準を選択し、そのコンポーネントを BAC に追加します。

```
    List<DeviceID> packetcableMTADeviceIDList = new ArrayList<DeviceID>();
    packetcableMTADeviceIDList.add(new MACAddress("1,6,01:02:03:04:05:07"));

    // Continuation of the batch in Step2
    // To add the MTA component:

    batch.add(
        DeviceType.PACKET_CABLE_MTA,  // deviceType: PACKET_CABLE_MTA
        packetcableMTADeviceIDList,   // macAddress: scanned from the label
        null,                         // hostName: not used in this example, will be
        auto                                // generated
        null,                         // domainName: not used in this example, will be
                                      // auto generated. The FqdnKeys.AUTO_FQDN_DOMAIN
                                      // property must be set somewhere in the property
                                      // hierarchy.
        "0123-45-6789",               // ownerID: here, account number from billing
        system
        "Silver",                     // ClassOfService
        "provisionedMTA",             // DHCP Criteria: Network Registrar uses this to
                                      // select an MTA lease granting provisioned IP
                                      // address
        null                          // properties: not used
    );

    BatchStatus batchStatus = null;

    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 4　組み込み型 MTA が顧客に出荷されます。

ステップ 5　顧客は組み込み型 MTA をオンラインにして、電話をかけます。

PacketCable eMTA 上での SNMP クローニング

管理者は SNMP Element Manager アクセス権を PacketCable eMTA に付与します。

目的

外部の Element Manager に、PacketCable eMTA へのセキュアな SNMPv3 アクセス権を付与します。

（注）　RW MIB 変数に加えた変更は永続的なものではなく、BAC の eMTA に関する設定では更新されません。eMTA MIB に書き込まれた情報は、次回 MTA を電源オフにするか、またはリセットしたときに失われます。

ステップ 1　プロビジョニング API メソッドである *performOperation(...)* を呼び出します。その際、MTA の MAC アドレスと、MTA 上に作成する新しいユーザのユーザ名を渡します。このユーザ名は、後で Element Manager が SNMP コールを行うときに使用されます。

```
    // Create a new connection
    PACEConnection conn = PACEConnectionFactory.getInstance(
        "localhost", 49187, "admin", "admin123");

    Batch batch = conn.newBatch(

        // No reset

        ActivationMode.NO_ACTIVATION,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION);

    // NO_ACTIVATION is the activation mode because we don't want to
    // reset the device.
    // NO_CONFIRMATION is the confirmation mode because we are
    // not attempting to reset the device.
    // The goal here is to create a new user on the MTA indicated
    // by the MAC address. The other parameter needed here is the new
    // user name, which is passed in the Map.
    // Create a map that contains one element - the name of
    // the new user to be created on the MTA

    HashMap<String, Object> map = new HashMap<String, Object>();
    map.put( SNMPPropertyKeys.CLONING_USERNAME, "newUser" );

    // The first param is the actual device operation to perform.

    batch.performOperation(
        DeviceOperation.ENABLE_SNMPV3_ACCESS,    // deviceOperation :
        ENABLE_SNMPV3_ACCESS
        new MACAddress("1,6,00:00:00:00:00:99"), // macORFqdn : MAC Address of the
        modem
        map                                      // parameters: operation specific
                                                 // parameters
    );

    BatchStatus batchStatus = null;

    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 2　プロビジョニング API は、ステップ 1 で渡された新しいユーザに対応するエントリを MTA 上に作成するため、SNMPv3 クローニング操作の実行を試みます。新しいユーザのエントリ行で使用されるキーは、BAC 内で定義された 2 つのパスワードの関数です。顧客はこれらのパスワードを使用できるようになります。また、RDU コマンドは、これらのパスワード（auth および priv パスワード）をキー ローカリゼーション アルゴリズムに渡して、auth キーおよび priv キーを作成します。これらのキーは、新しいユーザとセットで、eMTA のユーザ テーブルに格納されます。

（注）　このステップに記載されている auth および priv パスワードを変更するには、*rdu.properties* 設定ファイル内の SNMPPropertyKeys.CLONING_AUTH_PASSWORD（*/snmp/cloning/auth/password*）および SNMPPropertyKeys.CLONING_PRIV_PASSWORD（*/snmp/cloning/priv/password*）プロパティをそれぞれ設定変更します。

ステップ 3　顧客は、指定されたユーザ名、パスワード、およびキー ローカリゼーション アルゴリズムを使用して SNMPv3 要求を発行し、MTA とのセキュアな通信を可能にします。

PacketCable eMTA の差分プロビジョニング

顧客は、PacketCable eMTA を使用しており、その 1 本目の回線（エンドポイント）をイネーブルにしています。この顧客は、eMTA 上の 2 本目の電話回線（エンドポイント）をイネーブルにし、その回線に電話を接続します。

目的

顧客は eMTA 上の 2 本目の回線（エンドポイント）に電話を接続し、どのサービスも中断することなく正常に電話をかけられるようになる必要があります。

（注）　eMTA 上で 2 本目の回線を使用するには、コール エージェントを適切に設定する必要があります。コール エージェントのプロビジョニングについては、この使用例では取り上げません。

ステップ 1　サービス プロバイダーのアプリケーションは、BAC API を起動して、eMTA のサービス クラスを変更します。新しいサービス クラスは、eMTA 上の 2 つのエンドポイントをサポートします。このサービス クラスの変更は、eMTA がリセットされると有効になります。eMTA を中断することは好ましくありません。そのため、次のステップで差分プロビジョニングが行われます。

```
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");

Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION);

// NO_ACTIVATION is the activation mode because we don't want to
// reset the device.
// NO_CONFIRMATION is the Confirmation mode because we are not
// disrupting the device.

batch.changeClassOfService(
    new MACAddress("1,6,ff:00:ee:11:dd:22"),  / eMTA's MAC address or FQDN
    "twoLineEnabledCOS"                      // This COS supports two lines.
);
```

```
BatchStatus batchStatus = null;

// post the batch to RDU server

try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
{
    e.printStackTrace();
}
```

ステップ 2　サービス プロバイダーのアプリケーションは、BAC 差分更新機能を使用して、eMTA に SNMP オブジェクトを設定します。そのため、eMTA を中断することなく、サービスがイネーブルになります。

```
// The goal here is to enable a second phone line, assuming one
// phone line is currently enabled.   We will be adding a new
// row to the pktcNcsEndPntConfigTable.

batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION);

// NO_ACTIVATION is the activation mode because we don't want to
// reset the device.
// NO_CONFIRMATION is the confirmation mode because we are
// not attempting to reset the device.
// Create a map containing one element - the list of SNMP
// variables to set on the MTA

HashMap<String, Object> map = new HashMap<String, Object>();

// Create an SnmpVarList to hold SNMP varbinds

SnmpVarList list = new SnmpVarList();

// An SnmpVariable represents an oid/value/type triple.
// pktcNcsEndPntConfigTable is indexed by the IfNumber, which in this case we will
// assume is interface number 12 (this is the last number in each of the oids
below).
// The first variable represents the creation of a new row in
// pktcNcsEndPntConfigTable we are setting the RowStatus
// column (column number 26). The value of 4 indicates that
// a new row is to be created in the active state.

SnmpVariable variable = new SnmpVariable( ".1.3.6.1.4.1.4491.2.2.2.1.2.1.1.26.12",
    "4", SnmpType.INTEGER );
list.add( variable );

// The next variable represents the call agent id for this new
// interface, which we'll assume is 'test.com'

variable = new SnmpVariable( ".1.3.6.1.4.1.4491.2.2.2.1.2.1.1.1.12", "test.com",
    SnmpType.STRING );
list.add( variable );

// The final variable represents the call agent port
```

```
    variable = new SnmpVariable( ".1.3.6.1.4.1.4491.2.2.2.1.2.1.1.2.12", "2728",
        SnmpType.INTEGER );
    list.add( variable );

    // Add the SNMP variable list to the Map to use in the API call

    map.put( SNMPPropertyKeys.SNMPVAR_LIST, list );

    // Invoke the BACC API to do incremental update on the eMTA.

    batch.performOperation(
        DeviceOperation.INCREMENTAL_UPDATE,      // device operation
        new MACAddress("1,6,00:00:00:00:00:99"), // MAC Address
        map                                      // Parameters for the operation
    );

    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 3　eMTA で 2 本目の電話回線を使用できるようになります。ステップ 1 でサービス クラスが変更されているので、eMTA はリセット後も引き続き同じサービスを受けられます。

動的設定ファイルを使用した DOCSIS モデムの事前プロビジョニング

新しい顧客は、サービス プロバイダーに連絡して、モデムの背後に接続された 2 台の CPE を対象とする高速な *Gold* データ サービス付きの DOCSIS モデムを注文します。

目的

次のワークフローを使用して、DOCSIS テンプレートを使用するサービス クラスで DOCSIS モデムを事前プロビジョニングします。テンプレートから生成される動的設定ファイルは、モデムがオンラインになるときに使用されます。

ステップ 1　サービス　プロバイダーは、課金システムで使用される加入者のユーザ名とパスワードを選択します。

ステップ 2　サービス プロバイダーは、Gold サービス クラスと適切な DHCP 基準を選択し、ケーブル モデムを BAC に追加します。

```
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");

Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION);


Map<String, Object> properties = new HashMap<String, Object>();

// Set the property PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED to enable
// promiscuous mode on modem

properties.put(PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED, Boolean.TRUE);

// enable promiscuous mode by changing the technology default

batch.changeDefaults(DeviceType.DOCSIS,properties, null);

BatchStatus batchStatus = null;

// post the batch to RDU server

try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
{
    e.printStackTrace();
}
// No CPE DHCP Criteria is specified.
// The CPE behind the modem will use the default provisioned
// promiscuous CPE DHCP criteria specified in the system defaults.
// This custom property corresponds to a macro variable in the
// DOCSIS template for "gold" class of service indicating the
// maximum number of CPE allowed behind this modem. We set it
// to two sets of CPE from this customer.

properties = new HashMap<String, Object>();
properties.put("docsis-max-cpes", "2");

batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// To add a DOCSIS modem:

List<DeviceID> deviceIDList = new ArrayList<DeviceID>();
```

```
    deviceIDList.add(new MACAddress("1,6,01:02:03:04:05:06"));
    batch.add(
        DeviceType.DOCSIS,      // deviceType: DOCSIS
        deviceIDList,           // macAddress: scanned from the label
        null,                   // hostName: not used in this example
        null,                   // domainName: not used in this example
        "0123-45-6789",         // ownerID: here, account number from billing system
        "gold",                 // classOfService:
        "provisionedCM",        // DHCP Criteria: Network Registrar uses this to
                                // select a modem lease granting provisioned IP address
        properties              // properties:
    );

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 3　ケーブル モデムが顧客に出荷されます。

ステップ 4　顧客はケーブル モデムをオンラインにし、モデムの背後にコンピュータを接続します。

オプティミスティック ロッキング

サービス プロバイダー アプリケーションのインスタンスは、同じアプリケーションの別のインスタンスによって行われた変更を上書きしないようにします。

目的

次のワークフローを使用して、BAC API が提供するオプティミスティック ロッキング機能を実行します。

（注）　オブジェクトのロッキングは、マルチユーザのシステムで変更の整合性を維持するために実行されます。その結果、ユーザの変更は別のユーザによって不用意に上書きされなくなります。オプティミスティック ロッキングを使用してプログラムを記述する場合、コミットしようとするオブジェクトが 1 つでも処理の開始後に別のユーザによって変更されていたときは、コミットが失敗する可能性があります。

ステップ 1　サービス担当者は、サービス プロバイダーのユーザ インターフェイスで検索オプションを選択し、ケーブル モデムの MAC アドレスを入力します。

ステップ 2　BAC は、組み込みデータベースにクエリーし、デバイスの詳細を取得します。次に、その情報が MSO ユーザ インターフェイスに表示されます。

```
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");

Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);


final DeviceID modemMACAddress = DeviceID.getInstance("1,6,00:11:22:33:44:55",
        KeyType.MAC_ADDRESS);
List<DeviceDetailsOption> options = new ArrayList<DeviceDetailsOption>();

options.add(DeviceDetailsOption.INCLUDE_LEASE_INFO);

// MSO admin UI calls the provisioning API to query the details
// for the requested device. Query may be performed based on MAC
// address or IP address, depending on what is known about the
// device.

batch.getDetails(modemMACAddress, options);

// post the batch to RDU server

BatchStatus batchStatus = null;

try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
{
    e.printStackTrace();
}
```

ステップ 3　サービス担当者は、ユーザ インターフェイスを使用して、モデムのサービス クラスと DHCP 基準の変更を試みます。その結果、BAC API が起動します。

```
    Map<String, Object> deviceDetails = new TreeMap((Map<String,
        Object>)batchStatus.getCommandStatus(0).getData());

    //  extract device detail data from the map

    String deviceType = (String)deviceDetails.get(DeviceDetailsKeys.DEVICE_TYPE);
    String macAddress = (String)deviceDetails.get(DeviceDetailsKeys.MAC_ADDRESS);
    String relayAgentID = (String)deviceDetails.get(DeviceDetailsKeys.RELAY_AGENT_MAC);
    Boolean isProvisioned =
    (Boolean)deviceDetails.get(DeviceDetailsKeys.IS_PROVISIONED);

    //  Let's save the OID_REVISION_NUMBER property so that we can set it in
    //  step 3.

    String oidRevisionNumber =
        (String)deviceDetails.get(GenericObjectKeys.OID_REVISION_NUMBER);

    // We need a reference to Batch instance so that ensureConsistency()
    // method can be invoked on it.

    batch = conn.newBatch() ;
    List<String> oidList = new ArrayList<String>();

    // Add the oid-rev number saved from step 2 to the list

    oidList.add(oidRevisionNumber);

    // Sends a list of OID revision numbers to validate before processing the
    // batch. This ensures that the objects specified have not been modified
    // since they were last retrieved.

    batch.ensureConsistency(oidList);
    batch.changeClassOfService (
        new MACAddress("1,6,00:11:22:33:44:55"),  // macORFqdn: unique identifier for
        the                                           // device.
        "gold"                                    // newCOSName : Class of service
        name.
    );

    batch.changeDHCPCriteria (
        new MACAddress("1,6,00:11:22:33:44:55"),  // macORFqdn: unique identifier for
        the                                           // device.
        "specialDHCPCriteria"                     // newDHCPCriteria : New DHCP
        Criteria.
    );

    // This batch fails with BatchStatusCodes.BATCH_NOT_CONSISTENT,
    // in case if the device is updated by another client in the meantime.
    // If a conflict occurs, then the service provider client
    // is responsible for resolving the conflict by querying the database
    // again and then applying changes appropriately.
    }
}
```

ステップ 4　これで、ユーザは適切な DHCP 基準の Gold サービス クラスを受けられるようになります。

加入者の帯域幅の一時的なスロットリング

MSO は、加入者に許可するダウンロードのデータ量を 1 か月あたり 10 MB に制限するサービスを提供します。その限度に達すると、加入者のダウンストリーム帯域幅が 10 MB から 56 KB に制限されます。月が替わると 10 MB に戻ります。

（注）　ピアツーピア ユーザや Web サイトを運営するユーザはアップロード帯域幅の使用量が非常に高くなる傾向があるため、アップストリーム帯域幅の変更も考慮することが必要になる場合があります。

目的

次のワークフローを使用して、加入者の帯域幅をその契約条件に従って増減させます。

ステップ 1　MSO は *NetFlow* などのレート追跡システムを使用します。このシステムは、MAC アドレスに基づいて各顧客の使用率を追跡します。最初に、顧客は、1 MB のダウンストリームを持つ *Gold* サービス クラスのレベルでプロビジョニングされます。

ステップ 2　レート追跡ソフトウェアは、加入者が 10 MB の限度に達したことを特定すると、OSS に通知します。OSS は BAC API を呼び出して、加入者のサービス クラスを *Gold* から *Gold-throttled* に変更します。

```
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");

Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);


// AUTOMATIC is the activation mode because we are
// attempting to reset the modem so that it
// receives low bandwidth service.
// NO_CONFIRMATION is the confirmation mode
// because we do not want the batch to fail if we cannot
// reset the modem. If the modem is off, then it will
// be disabled when it is turned back on.
// Let's change the COS of the device so that it restricts
// bandwidth usage of the modem.

batch.changeClassOfService(
    new MACAddress("1,6,00:11:22:33:44:55"),  // macAddress: unique identifier for
                                              // this modem
    "Gold-throttled"                          // newClassOfService: restricts
                                              // bandwidth usage to 56k
);

BatchStatus batchStatus = null;
```

```
    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 3　請求対象期間の終了時に、OSS は BAC API を呼び出して、加入者のサービス クラスを *Gold* に戻します。

CableHome WAN-MAN の事前プロビジョニング

新しい顧客は、サービス プロバイダーに連絡してホーム ネットワーキング サービスを注文します。顧客は、プロビジョニングされた CableHome デバイスを受け取ります。

目的

次のワークフローを使用して、CableHome デバイスを事前プロビジョニングし、ケーブル モデムとその WAN-MAN コンポーネントがオンラインになったときに適切なレベルのサービスを受けられるようにします。

ステップ 1　サービス　プロバイダーは、課金システムで使用される加入者のユーザ名とパスワードを選択します。

ステップ 2　サービス プロバイダーは、モデム コンポーネントに適切なサービス クラスと DHCP 基準を選択し、そのコンポーネントを BAC に追加します。

```
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");

Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation
    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// Let's provision the modem and the WAN-Man component in the same
// batch.
// To add a DOCSIS modem:

List<DeviceID> docisDeviceIDList = new ArrayList<DeviceID>();
docisDeviceIDList.add(new MACAddress("1,6,01:02:03:04:05:06"));
batch.add(
    DeviceType.DOCSIS,      // deviceType: DOCSIS
    docisDeviceIDList,      // macAddress: scanned from the label
    null,                   // hostName: not used in this example
    null,                   // domainName: not used in this example
    "0123-45-6789",         // ownerID: here, account number from billing system
    "Silver",               // classOfService
    "provisionedCM",        // DHCP Criteria: Network Registrar uses this to
                            // select a modem lease granting provisioned IP address
    null                    // properties: not used
);
```

ステップ 3　サービス プロバイダーは、WAN-MAN コンポーネントに適切なサービス クラスと DHCP 基準を選択し、そのコンポーネントを BAC に追加します。

```
List<DeviceID> wanManDeviceIDList = new ArrayList<DeviceID>();
wanManDeviceIDList.add(new MACAddress("1,6,01:02:03:04:05:07"));
batch.add(
    DeviceType.CABLEHOME_WAN_MAN,   // deviceType: CABLEHOME_WAN_MAN
    wanManDeviceIDList,             // macAddress: scanned from the label
    null,                           // hostName: not used in this example
    null,                           // domainName: not used in this example
    "0123-45-6789",                 // ownerID: here, account number from billing
                                    // system
    "silverWanMan",                 // classOfService
    "provisionedWanMan",            // DHCP Criteria: Network Registrar uses this
    to
                                   // select a modem lease granting provisioned IP
                                    // address
    null                            // properties: not used
);
}
```

ステップ 4　CableHome デバイスが顧客に出荷されます。

ステップ 5　顧客は CableHome デバイスをオンラインにします。

ファイアウォール設定を持つ CableHome

顧客はサービス プロバイダーに連絡して、ファイアウォール機能がイネーブルになったホーム ネットワーキング サービスを注文します。顧客は、プロビジョニングされた CableHome デバイスを受け取ります。

目的

次のワークフローを使用して、CableHome デバイスを事前プロビジョニングし、ケーブル モデムとその WAN-MAN コンポーネントがオンラインになったときに適切なレベルのサービスを受けられるようにします。

ステップ 1　サービス　プロバイダーは、課金システムで使用される加入者のユーザ名とパスワードを選択します。

ステップ 2　サービス プロバイダーは、ケーブル モデム コンポーネントに適切なサービス クラスと DHCP 基準を選択し、そのコンポーネントを BAC に追加します。

```
PACEConnection conn = PACEConnectionFactory.getInstance(
    "localhost", 49187, "admin", "admin123");

Batch batch = conn.newBatch(

    // No reset

    ActivationMode.NO_ACTIVATION,

    // No need to confirm activation

    ConfirmationMode.NO_CONFIRMATION,

    // No publishing to external database

    PublishingMode.NO_PUBLISHING);

// Let's provision the modem and the WAN-Man component in the same
// batch.
// To add a DOCSIS modem:

List<DeviceID> docisDeviceIDList = new ArrayList<DeviceID>();
docisDeviceIDList.add(new MACAddress("1,6,01:02:03:04:05:06"));
batch.add(
    DeviceType.DOCSIS,      // deviceType: DOCSIS
    docisDeviceIDList,      // macAddress: scanned from the label
    null,                   // hostName: not used in this example
    null,                   // domainName: not used in this example
    "0123-45-6789",         // ownerID: here, account number from billing system
    "Silver",               // classOfService
    "provisionedCM",        // DHCP Criteria: Network Registrar uses this to
                            // select a modem lease granting provisioned IP address
    null                    // properties: not used
);
```

ステップ 3　サービス プロバイダーは、WAN-MAN コンポーネントに適切なサービス クラスと DHCP 基準を選択し、そのコンポーネントを BAC に追加します。

```
//      Continuation of the batch in Step 2
// To add the WAN-Man component:
// Create a Map to contain WanMan's properties

Map<String, Object> properties = new HashMap<String, Object>();

// The fire wall configuration for the Wan Man component is specified
// using the CableHomeKeys.CABLEHOME_WAN_MAN_FIREWALL_FILE property.
// This use case assumes that the firewall configuration file named
// "firewall_file.cfg" is already present in the RDU database and the
// firewall configuration is enabled in the Wan Man configuration file
// specified with the corresponding class of service.

properties.put(CableHomeKeys.CABLEHOME_WAN_MAN_FIREWALL_FILE, "firewall_file.cfg");

List<DeviceID> wanManDeviceIDList = new ArrayList<DeviceID>();
wanManDeviceIDList.add(new MACAddress("1,6,01:02:03:04:05:07"));
batch.add(
    DeviceType.CABLEHOME_WAN_MAN,    // deviceType: CABLEHOME_WAN_MAN
    wanManDeviceIDList,              // macAddress: scanned from the label
    null,                          // hostName: not used in this example
    null,                          // domainName: not used in this example
    "0123-45-6789",                // ownerID: here, account number from billing
    system
    "silverWanMan",                // classOfService
    "provisionedWanMan",           // DHCP Criteria: Network Registrar uses this to
                                   // select a modem lease granting provisioned IP
                                   // address
    null                           // properties: not used
);

BatchStatus batchStatus = null;

// post the batch to RDU server

try
{
    batchStatus = batch.post();
}
catch(ProvisioningException e)
{
    e.printStackTrace();
}
}
```

ステップ 4　CableHome デバイスが顧客に出荷されます。

ステップ 5　顧客は CableHome デバイスをオンラインにします。次に、ケーブル モデムと WAN-MAN コンポーネントは、プロビジョニングされた IP アドレスと正しい設定ファイルを取得します。

CableHome WAN-MAN のデバイス機能の取得

サービス プロバイダーは管理者に対して、CableHome WAN-MAN デバイスの機能の情報を表示することを許可します。

目的

サービス プロバイダーの管理アプリケーションは、特定の CableHome WAN-MAN コンポーネントに関する既知の詳細をすべて表示します。この詳細には、MAC アドレス、リース情報、プロビジョニング ステータス、およびデバイス機能の情報などがあります。

ステップ 1　管理者は、サービス プロバイダーのユーザ インターフェイスに、クエリーする WAN-MAN の MAC アドレスを入力します。

ステップ 2　BAC は、入力された MAC アドレスを使用して特定したデバイスの詳細を、組み込みデータベースにクエリーします。

```
    PACEConnection conn = PACEConnectionFactory.getInstance(
        "localhost", 49187, "admin", "admin123");


    Batch batch = conn.newBatch(

        // No reset

        ActivationMode.NO_ACTIVATION,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION,

        // No publishing to external database

        PublishingMode.NO_PUBLISHING);


    final DeviceID modemMACAddress = DeviceID.getInstance("1,6,00:11:22:33:44:55",
            KeyType.MAC_ADDRESS);

    List<DeviceDetailsOption> options = new ArrayList<DeviceDetailsOption>();
    options.add(DeviceDetailsOption.INCLUDE_LEASE_INFO);

    // MSO admin UI calls the provisioning API to query the details
    // for the requested device. Query may be performed based on MAC
    // address or IP address, depending on what is known about the
    // device.

    batch.getDetails(modemMACAddress, options);

    // post the batch to RDU server

    BatchStatus batchStatus = null;
    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 3　サービス プロバイダーのアプリケーションは、デバイス データの詳細に関するページを表示します。このページには、要求されたデバイスに関する既知の情報をすべて表示できます。デバイスがサービス プロバイダーのネットワークに接続された場合、このデータにはリース情報（IP アドレスやリレー エージェントの ID など）が含まれます。このデータは、デバイスがプロビジョニングされているかどうかを示します。プロビジョニングされている場合、データにはデバイス タイプとデバイス機能の情報も含まれます。

```
    Map<String, Object> deviceDetails = new TreeMap((Map<String,
        Object>)batchStatus.getCommandStatus(0).getData());

    //  extract device detail data from the map

    String deviceType = (String)deviceDetails.get(DeviceDetailsKeys.DEVICE_TYPE);
    String macAddress = (String)deviceDetails.get(DeviceDetailsKeys.MAC_ADDRESS);
    String relayAgentID = (String)deviceDetails.get(DeviceDetailsKeys.RELAY_AGENT_MAC);
    Boolean isProvisioned =
    (Boolean)deviceDetails.get(DeviceDetailsKeys.IS_PROVISIONED);

    String deviceID = (String) deviceDetails.get(CNRNames.DEVICE_ID.toString());
    String serNum = (String)
    deviceDetails.get(CNRNames.DEVICE_SERIAL_NUMBER.toString());
    String hwVer = (String)
        deviceDetails.get(CNRNames.HARDWARE_VERSION_NUMBER.toString());
    String swVer = (String)
        deviceDetails.get(CNRNames.SOFTWARE_VERSION_NUMBER.toString());
    String brVer = (String) deviceDetails.get(CNRNames.BOOT_ROM_VERSION.toString());
    String vendorOui = (String) deviceDetails.get(CNRNames.VENDOR_OUI.toString());
    String modelNum = (String) deviceDetails.get(CNRNames.MODEL_NUMBER.toString());
    String vendorNum = (String) deviceDetails.get(CNRNames.VENDOR_NAME.toString());

    //  The admin UI now formats and prints the detail data to a view page
    }
}
```

CableHome WAN-MAN のセルフプロビジョニング

加入者は、戸建住宅内にブラウザ アプリケーションを持つコンピュータを設置し、組み込み CableHome デバイスを購入しました。

目的

次のワークフローを使用して、プロビジョニングされていない新しい組み込み CableHome デバイスをオンラインにし、適切なレベルのサービスを受けられるようにします。さらに、組み込み CableHome デバイスに接続されたコンピュータから加入者がインターネットにアクセスできるようにします。

ステップ 1　加入者は組み込み CableHome デバイスを購入し、住宅内に設置します。

ステップ 2　加入者は、組み込み CableHome デバイスの電源をオンにします。BAC は、制限付きのアクセス権を組み込みケーブル モデムに付与します。つまり、アクセス権は 2 台の CPE（CableHome WAN-MAN とコンピュータ）に制限されます。

（注）　この使用例は、プロビジョニングされていない DOCSIS モデムの背後に 2 台の CPE を接続できることを前提としています。特に設定しない限り、BAC では、プロビジョニングされていない DOCSIS モデムの背後に接続できるデバイスは 1 台のみです。この動作を変更するには、2 台の CPE をサポートする適切なサービス クラスを定義してから、そのサービス クラスを DOCSIS デバイスのデフォルト サービス クラスとして使用します。

ステップ 3　BAC は CableHome WAN-MAN を設定します。この設定には、IP 接続や、デフォルトの CableHome ブート ファイルのダウンロードなどがあります。デフォルトの CableHome ブート ファイルは、CableHome デバイスをパススルー モードに設定します。CableHome デバイスはまだプロビジョニングされていません。

ステップ 4　加入者は、コンピュータを CableHome デバイスに接続します。コンピュータは、プロビジョニングされていない（制限された）IP アドレスを取得します。加入者はコンピュータ上でブラウザ アプリケーションを起動します。スプーフィング DNS サーバにより、ブラウザがサービス プロバイダーの登録サーバ（OSS ユーザ インターフェイスやメディエータなど）にアクセスします。

ステップ 5　加入者はサービス プロバイダーのユーザ インターフェイスを使用して、ケーブル モデムの登録に必要な手順（サービス クラスの選択など）を終了します。また、加入者は CableHome のサービス クラスも選択します。

ステップ 6　サービス プロバイダーのユーザ インターフェイスは、加入者の情報（ケーブル モデムと CableHome に対して選択されたサービス クラスやコンピュータの IP アドレスなど）を BAC に渡します。次に、加入者が BAC に登録されます。

ステップ 7　ユーザ インターフェイスは、加入者にコンピュータをリブートするように求めます。

ステップ 8　プロビジョニング クライアントは *performOperation(...)* を呼び出してモデムをリブートし、プロビジョニングされたアクセス権をモデムに付与します。

```
    // create a new batch

    batch = conn.newBatch(

        // No reset

        ActivationMode.AUTOMATIC,

        // No need to confirm activation

        ConfirmationMode.NO_CONFIRMATION);


    // register performOperation command to the batch

    batch.performOperation(DeviceOperation.RESET, modemMACAddressObject, null);

    // post the batch to RDU server

    try
    {
        batchStatus = batch.post();
    }
    catch(ProvisioningException e)
    {
        e.printStackTrace();
    }
}
```

ステップ 9　コンピュータをリブートすると、コンピュータは CableHome デバイスの DHCP サーバから新しい IP アドレスを受信します。これで、ケーブル モデムと CableHome デバイスはいずれもプロビジョニングされます。この結果、加入者は CableHome デバイスのイーサネット ポートに複数のコンピュータを接続し、インターネットにアクセスすることができます。

（注）　WAN-MAN コンポーネントに提供された設定ファイルによってボックスの WAN-Data コンポーネントがイネーブルにされる場合、コンポーネントは無差別モードでプロビジョニングされます。無差別モードは、DeviceType.CABLEHOME_WAN_DATA デバイス タイプのテクノロジー デフォルト レベルでイネーブルになることが前提となっています。