CHAPTER 1

概要

Cisco IP Interoperability and Collaboration System（IPICS）の 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）は、Web サービス ベースの API であり、プログラマティック インターフェイスとカスタム アプリケーションを使用して、Cisco IPICS のさまざまなオペレーションを管理および制御できます。この API には、Cisco IPICS のオペレーション（VTG の作成、ポリシーの呼び出し、ユーザ管理など）を制御する関数セットが含まれています。

この章では、API に関する一般的な概念を紹介し、API 要求を実行するためのオプションについて説明します。次のトピックについて説明します。

API 要求の実行

API 要求を実行するには、目的の API 関数を Cisco IPICS サーバに送信します。Cisco IPICS API 要求は、URL を使用して実行するか、あるいは Pure Java オブジェクトを指定した Web サービスを使用して実行できます。この章では、これらの方法について例を示しながら説明します。次のトピックについて説明します。

URL を使用した API 要求の実行

API 要求は、あらゆるブラウザから実行できます。ここでは、これらの要求の実行について説明します。Cisco IPICS API の Web Services Description Language（WSDL）ファイルにも、役立つ情報が含まれています。第 4 章「Web Services Description Language（WSDL）ファイル」を参照してください。

ガイドライン

URL 方式を使用して API 要求を実行する場合は、次のガイドラインに留意してください。

- ブラウザ セッションで初めて要求を実行すると場合は、ユーザ名とパスワードを要求するメッセージが表示されます。Cisco IPICS のユーザ名とパスワードを入力してください。
- API 応答の出力形式は、使用しているブラウザによって異なります。

- API からのエラー メッセージが Internet Explorer に直接表示されるようにするには、Internet Explorer を起動して [Internet Options] を選択します。[Advanced] タブで、[Browsing] オプション リストの [Show friendly HTTP error messages] チェックボックスをオンにします。
- Google Chrome では、復帰改行文字と XML タグが API 応答から削除されます。

URL 形式

URL を介して API 要求を Cisco IPICS サーバに送信するには、次の URL をブラウザに入力します。

https://<*Cisco_IPICS_Server*>/ipics_server/services/NorthboundService/<*function*>[?<*parameter*>=<*value*>][&<*function*>[?<*parameter*>=<*value*>]...]

この URL の説明は、次のとおりです。

- <*Cisco_IPICS_Server*> : Cisco IPICS サーバの IP アドレスまたはホスト名を指定する。
- <*function*> : 使用する API 関数を指定する。
- <*parameter*> : 必要に応じて、使用する関数パラメータを指定する。
- <*value*> : 必要に応じて、使用するパラメータ値を指定する。

例

ここでは、さまざまな API 関数を実行する URL 要求の例を示します。この例では、<*Cisco_IPICS_Server*> に Cisco IPICS サーバの IP アドレスまたはホスト名を指定します。

例 1 : VTG を取得する

Cisco IPICS サーバに設定されている VTG をすべて取得するには、次の URL をブラウザに入力します。

https://<*Cisco_IPICS_Server*>/ipics_server/services/NorthboundService/getAllVtgs

応答例 :

```
<ns:getAllVtgsResponse>
        <ns:getAllVtgsReturn>
                <ns:vtgVOs>
                        <ns:vtgVO>
                                <ns:name>vtg1</ns:name>
                                <ns:id>260</ns:id>
                        </ns:vtgVO>
                        <ns:vtgVO>
                                <ns:name>vtg2</ns:name>
                                <ns:id>268</ns:id>
                        </ns:vtgVO>
                </ns:vtgVOs>
        </ns:getAllVtgsReturn>
</ns:getAllVtgsResponse
```

例 2 : VTG をアクティブにする

ID が 260 の VTG をアクティブにするには、次の URL をブラウザに入力します。

https://<*Cisco_IPICS_Server*>/ipics_server/services/NorthboundService/activateVtg?vtgId=260

応答 :

```
<ns:activateVtgResponse>
        <ns:activateVtgReturn>true</ns:activateVtgReturn>
</ns:activateVtgResponse>
```

例 3：ユーザを作成する

ログイン ID が test でパスワードが abc123 の Cisco IPICS ユーザを作成するには、次の URL をブラウザに入力します。

https://<*Cisco_IPICS_Server*>/ipics_server/services/NorthboundService/createUser?userLogin=test&userPassword=abc123

応答例：

```
<ns:createUserResponse>
        <ns:createUserReturn>6881</ns:createUserReturn>
</ns:createUserResponse>
```

例 4：強制エラーにする

Cisco IPICS API がエラー メッセージを返すようにするには、次の URL をブラウザに入力します。

https://my-ipics/ipics_server/services/NorthboundService/createUser?userLogin=test

応答例：

```
<faultstring>
Required element {http://com.cisco.ipics.issymo.northbound.service}userPassword defined in
the schema can not be found in the request
</faultstring>
```

Java を使用した API 要求の実行

SOAP、Axis2、JiBX のライブラリを指定してクライアント アプリケーションを Java に実装すると、API 要求を実行できます。これには、次の一般的な手順に従います。

1. issymo-northbound-client-2.2-SNAPSHOT-jar-with-dependencies.jar ファイルをローカル ドライブに置きます。

   これには、scp ユーティリティを使用して、Cisco IPICS の /opt/cisco/ipics/examples フォルダからクライアント jar ファイルをダウンロードします（*Cisco_IPICS_Server* には Cisco IPICS サーバの IP アドレスまたはホスト名を指定します）。

   scp root@<*Cisco_IPICS_Server*>:/opt/cisco/ipics/examples/issymo-northbound-client-2.2-SNAPSHOT-jar-with-dependencies.jar

2. API 方式を呼び出すアプリケーションを記述します。

次のリソースを参考にできます。

- JiBX マッピング ファイル：応答で使用される Java クラスを定義し、XML 応答のどの部分が Java オブジェクトのどのフィールドにマップされるかを定義する。JiBX マッピング ファイルとその他のリソースは、issymo-northbound-common-2.2-SNAPSHOT.jar という名前のファイルに含まれています。このファイルの詳細とロケーションについては、第 3 章「JiBX マッピング ファイル」を参照してください。
- binding.xml ファイル：複雑なオブジェクト タイプの処理方法を JiBX に提供する。このファイルも issymo-northbound-common-2.2-SNAPSHOT.jar ファイルに含まれています。
- 値オブジェクト：複雑な実装タイプを定義する。これらも issymo-northbound-common-2.2-SNAPSHOT.jar ファイルに含まれています。
- WISDL ファイル：呼び出すことが可能な方式と、それらの方式から返されるオブジェクト タイプが含まれる。一部のオブジェクトは複雑なタイプです。詳細については、第 4 章「Web Services Description Language（WSDL）ファイル」を参照してください。
- northboundservice.class：クライアント スタブ。issymo-northbound-client-2.2-SNAPSHOT-jar-with-dependencies.jar ファイルに含まれています。

例

Web サービスには、いろいろな方式を利用してアクセスできます。次の例では、Axis2 を使用してクライアント API 要求をビルドします。Axis2 はスタブ生成機能を使用します。この機能によって WSDL2Java ツールが WSDL の記述からコードを生成し、JiBX を使用して XML を構築するクライアント アプリケーションを生成します。

詳細については、Apache Axis2 のマニュアルと第 3 章「JiBX マッピング ファイル」を参照してください。

次の例では、Cisco IPICS に付属しているリファレンス Web サービス クライアント スタブ、com.cisco.ipics.issymo.northbound.client.NorthboundServiceStub を使用します。このスタブは、issymo-northbound-client-2.2-SNAPSHOT-jar-with-dependencies.jar ファイルに含まれています。

このスタブは Axis2 と JiBX を使用し、wsdl2java によって生成されました。また、wsdl2java ユーティリティによって、API が使用する複雑なタイプである「値オブジェクト」も生成されました。ファイル issymo-northbound-common-2.2-SNAPSHOT.jar には、次の値オブジェクトのリファレンス実装が含まれています。

```
import java.rmi.RemoteException;
import org.apache.axis2.AxisFault;
import org.apache.axis2.client.Options;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.cisco.ipics.issymo.northbound.client.NorthboundServiceStub;
import com.cisco.ipics.issymo.northbound.client.NorthboundService;
import com.cisco.ipics.issymo.northbound.service.UserContainerVO;
import com.cisco.ipics.issymo.northbound.service.UserVO;

public class IpicsApiClient {

    /**
     * @param args The program arguments are as follows:
     * 1. Hostname or IP address of your Cisco IPICS server
     * 2. The Java trust store containing the Cisco IPICS server's SSL
     *    certificate(s).
     * 3. A Cisco IPICS user login name
     * 4. A Cisco IPICS user password
     */
    public static void main(String[] args) {
        if(args.length!=4) {
            System.err.println("ERROR: Insufficient arguments");
            System.exit(1);
        }

        final String HOSTNAME   = args[0];
        final String TRUSTSTORE = args[1];
        final String USERNAME   = args[2];
        final String PASSWORD   = args[3];

        try {
            System.setProperty("javax.net.ssl.trustStore", TRUSTSTORE);

            NorthboundService svc = initService(HOSTNAME, USERNAME, PASSWORD);

            //
            // Invoke the createUser API method which returns a simple int
            //
            try {
                final String userLogin =
                    "testuser-"+System.currentTimeMillis();
                final String userPassword = "secret";

                System.out.println("Create Users:");
```

```
            int newUserId = svc.createUser(userLogin, userPassword);
            System.out.println("  user id= " + newUserId);
        } catch (RemoteException e) {
            System.err.println("ERROR: "+e.getMessage());
        }
        System.out.println();

        //
        // Invoke the getAllUsers API method which returns a complex type,
        // the UserContainerVO "value object".
        //
        try {
            System.out.println("All Users:");
            System.out.println("----------------------- ");
            UserContainerVO userContainer = svc.getAllUsers();
        if(userContainer.getUserVOs() != null) {
            for (UserVO user : userContainer.getUserVOs()) {
                System.out.println(user.toString());
                System.out.println("  id    : "
                        + user.getId());
                System.out.println("  login : "
                        + user.getUserLogin());
                System.out.println("  name  : "
                        + user.getFirstName()
                        + user.getLastName());
                System.out.println();
            }
        } else {
            System.out.println("No users found");
        }
        } catch (RemoteException e) {
            System.err.println("ERROR: "+e.getMessage());
        }

    } catch (AxisFault ex) {
        System.err.println(ex);
    }
}

/**
 * Get service stub for target end point URL and setup the client for
 * BASIC authentication.
 * @param hostname Cisco IPICS server hostname or IP address
 * @param username Cisco IPICS user login name
 * @param password Cisco IPICS user password
 * @return An authenticated service stub, or an AxisFault exception.
 * @throws AxisFault If authentication fails, or the service is
 * unreachable.
 */
public static NorthboundService initService(String hostname,
        String username, String password) throws AxisFault
{
    NorthboundServiceStub stub;
    HttpTransportProperties.Authenticator auth =
        new HttpTransportProperties.Authenticator();
    auth.setPreemptiveAuthentication(true);
    auth.setUsername(username);
    auth.setPassword(password);

    final String ws_url =
        "https://"+hostname+"/ipics_server/services/NorthboundService";

    stub = new NorthboundServiceStub(ws_url);
    Options options = stub._getServiceClient().getOptions();
```

```
            options.setProperty(HTTPConstants.AUTHENTICATE, auth);

            return stub;
        }
}
```

API セキュリティ

Cisco IPICS サーバと Cisco IPICS API は SSL を使用して、サーバとクライアント間の安全な通信を確立します。サーバは、クライアントがサーバに接続を試みると、X.509 証明書（*SSL 証明書*とも呼ぶ）を使用してその ID を確認します。

Cisco IPICS サーバはデフォルトで、一般的にクライアントによって拒否される自己署名証明書を提示します。クライアントがこの証明書を拒否しないようにするには、表 1-1 に示す処理の 1 つを実行します。

表 1-1　Cisco IPICS サーバの自己署名証明書をクライアントが拒否しないようにする方法

方法	メモ
Cisco IPICS サーバの自己署名証明書を、VeriSing や組織内のローカル Certificate Authority（CA; 認証局）など、一般的な CA の署名付き証明書に置き換える。	詳細については、『*Cisco IPICS Server インストレーション アップグレード ガイド*』の「Cisco IPICS サーバへのサードパーティ証明書のインストール」の項を参照してください。
Java クライアントを使用している場合は、Java キーツールを使用して証明書をクライアントのトラストストアにインポートし、自己署名証明書を信頼するように SSL ライブラリを設定する。	手順： 1. SSH を使用して、ルート ユーザとして Cisco IPICS サーバにログインします。 2. 次のコマンドを入力します。 **cd /opt/cisco/ipics/tomcat/current/conf** 3. 次のコマンドを入力します。 <*certificate_file*> には、証明書を保持しているファイルの名前を指定します。 **keytool -export -alias tomcat** **-keystore tomcat.keystore** **-keypass changeit -file <***certificate_file***>** 4. 自己署名証明書をクライアント システムにダウンロードします。 5. 次のコマンドを入力します。<*name*> にはエイリアスの一意の名前を、<*certificate_file*> にはクライアントにダウンロードしたファイルを、<*path*> にはクライアント キーストアのパスを指定します。 **keytool -import -alias** <*name*> **-file <***certificate_file***> - keystore** <*path*>

表 1-1　Cisco IPICS サーバの自己署名証明書をクライアントが拒否しないようにする方法 （続き）

方法	メモ
Java 以外のクライアントを使用している場合は、自己署名証明書を信頼するようにクライアントの SSL ライブラリを設定する。	自己署名証明書をクライアント システムの特別なディレクトリにコピーできる場合があります。 詳細については、ライブラリのマニュアルを参照してください。
クライアント コードが証明書の信頼チェーンを検証する方法を変更する。	一部の言語では、デフォルトの証明書検証動作を変更できる設定可能な SSL ファイルが提供されています。

クライアントは、クライアント アプリケーションからサーバに送信されたユーザ名とパスワードを使用して ID を検証します。

API ロギング

Cisco IPICS サーバは、すべての Web サービス呼び出しをログに記録します。呼び出しの日時、使用された API 関数、応答のステータスが記録されます。

この情報は Cisco IPICS のログ ファイルに保存されます。ログ ファイルの詳細については、『*Cisco IPICS Administration Guide*』を参照してください。