



Cisco ACNS ソフトウェア API の概要

Cisco Application and Content Networking System (ACNS) 5.5 ソフトウェアには、次の 8 種類の API が用意されています。

- 複製状況
- チャンネル プロビジョニング
- ロケーション プロビジョニング
- Content Engine プロビジョニング
- プログラム
- リスト表示
- モニタリング統計情報
- ストリーミング統計情報

これらの HTTPS API は Java Servlet であり、結果を示す出力は XML 形式で生成されます。ACNS 5.5 ソフトウェアは、これらの 8 つの servlet を使用して、指定されたコンテンツの取得および配信パラメータをモニタリングし、変更します。表 1-1 に、これら 8 つの API を示します。いずれの場合も、チャンネルを特定できるように、API に固有の Web サイトとチャンネル名を提供する必要があります。

表 1-1 Cisco ACNS ソフトウェア API

API	説明
複製状況	チャンネル、Content Engine、またはコンテンツのリストを返し、指定のチャンネルのコンテンツの複製が完了したかどうかをチャンネルごとに示します。
プロビジョニング API	Content Distribution Manager に、ACNS チャンネル、ロケーション、Content Engine 情報を提供します。 <ul style="list-style-type: none"> チャンネルプロビジョニング API — ACNS ネットワーク チャンネルのモニタリングおよび変更を行います。 ロケーションプロビジョニング API — ACNS ネットワーク ロケーション オブジェクトの作成、変更、または削除を行います。 Content Engine プロビジョニング API — 指定された Content Engine のアクティブ化、検索、または削除を行います。 プログラム API — プログラムの作成、変更、検証、または削除を行います。また、プログラムに対して、Content Engine、デバイス グループ、およびチャンネルの割り当てや割り当て解除を行います。
リスト表示 API	ローカルの組み込みデータベースからオブジェクト情報を取得します。
モニタリング統計情報 API	単一の Content Engine または ACNS ネットワーク内のすべての Content Engine に関するモニタリング統計情報データを取得します。
ストリーミング統計情報 API	Content Engine またはデバイス グループから収集した WMT ¹ 、HTTP、RealProxy、および Cisco Streaming Engine のデータを報告し、このデータを Content Distribution Manager に送信します。ストリーミング統計情報 API で取得されたデータを保存して、カスタマイズされたレポートを生成できます。

1. WMT; Windows Media Technologies

また、ACNS 5.5 ソフトウェアでは、Authentication, Authorization, Accounting (AAA; 認証、許可、アカウントリング) 機能により、ユーザは外部のサーバとローカルのデータベースにアクセスすることができます。認証は、ユーザ ID と IP アドレスを検証します。許可は、ACNS ネットワーク内で認証されたユーザのアクセス権を許可または拒否します。アカウントリングは、許可されたネットワーク サービスの使用状況をログに記録します。これらの AAA 機能は API によって実行されるので、API を実行する前に、ユーザ証明書を有効化する必要があります。

この章の構成は、次のとおりです。

- [API コール \(p.1-3\)](#)
- [Java プログラムのサンプル \(p.1-4\)](#)
- [API エラー メッセージ \(p.1-8\)](#)
- [API タスク \(p.1-10\)](#)

API コール

ACNS 5.5 ソフトウェアの API は対話的に、またはコールプログラムを使用して実行できます。API コールは構文に従って正しく記述されていなければなりません。ユーザ証明書が無効だったり、構文に誤りがあったりすると、API は実行されません。ユーザ側にエラーがある場合、警告が返されます。警告には、エラーの性質およびエラーを起こした API の構文が記載されています。

対話型コール

ブラウザまたは“lynx”コマンドを使用すると、API を対話的に実行できます。認証と許可を行うために、ユーザ名とパスワードの入力を求めるプロンプトが表示されます。ユーザが確認されると API が実行されます。処理が完了すると出力結果がユーザに返されます。ブラウザを使用して API コールを行っていた場合、出力はブラウザに表示されます。または、“lynx”コマンドを使用した場合は、出力はファイルに転送できます。API の実行が失敗した場合は、エラーメッセージが表示されます。

プログラミング コール

API コールを行うには、HTTPS 要求を使用してコールプログラムを記述します。AAA 認証用の、ユーザ名とパスワードが HTTPS 要求内に設定されます。ユーザの確認とそれに続く API の実行が成功すると、結果を示す出力が返されます。返される内容は、出力または完了コードのいずれかです。API の実行が失敗した場合は、エラーコードが返されます。

Java プログラムのサンプル

次に API を構文解析する SAX (Simple API for XML) を 2 つ必要とする Java クライアントプログラムのサンプルを示します。このサンプルコードでは、構文解析用として org.xml.sax.* API および org.xml.sax.helpers.* API が、接続用として HTTPS URL パッケージが必要になります。

```
public class Client
{
    private static ApiParser parser_ = null;
    public static void main (String[] args)
    {
        parser_ = new ChannelApiServletParser();
        HttpUrlStreamHandlerFactory factory = new HttpUrlStreamHandlerFactory();
        factory.setSeed("test seed".getBytes());

        HttpUrlStreamHandler handler =
            (HttpUrlStreamHandler)factory.createURLStreamHandler("https");

        String sAuth = "admin:default";
        String sEncodedAuth = new sun.misc.BASE64Encoder().encode(sAuth.getBytes());

        Url url = new URL(null, "https://" + cdMAddress_ + ":" + cdMPort_ + "/servlet/" +
            "com.cisco.unicorn.ui.ChannelApiServlet?action=addManifest" +
            "&channel=" + channelId_ + "&manifest=http://www.cisco.com/test.xml"
+
            "&quota=10&ttl=20&user=user&password=password", handler);

        HttpURLConnection conn = (HttpURLConnection)url.openConnection();
        conn.setHeadProperty("Authorization: Basic " + sEncodedAuth);
        conn.setRequestMethod("GET");
        InputSource source = new InputSource(conn.getInputStream());
        parser_.parse(source);
    }
}
```

This program requires the parser to parse the output; a sample parser looks like this:

```
public class ApiParser extends DefaultHandler
{
    private XMLReader adapter_ = null;
    private String status_ = null;
    private String message_ = null;
    private String syntax_ = null;
    private String action_ = null;
    private int count_ = 0;
    private HashMap channels_ = new HashMap();
    private String thisChannel_ = null;
    private ArrayList clusters_ = new ArrayList();
    private ArrayList dgs_ = new ArrayList();

    public ChannelApiServletParser()
    {
        String parserName = "org.apache.xerces.parsers.SAXParser";

        try
        {
            adapter_ = XMLReaderFactory.createXMLReader(parserName);
            adapter_.setContentHandler(this);
            adapter_.setErrorHandler(this);
            adapter_.setEntityResolver(this);
            adapter_.setDTDHandler(this);
        }
        catch(SAXException saxe)
        {
            GlobalErrorHandler.recoverable(saxe);
        }
    }

    public void endElement (String aUri,
```

```
        String aLocalName,
        String aQName)
    throws SAXException
{
    if (aQName.equals("record") && action_.equals("getChannels"))
    {
        HashMap channelDetails = (HashMap)channels_.get(thisChannel_);

        if (clusters_.size() != 0)
        {
            channelDetails.put("Clusters", clusters_);
        }

        if (dgs_.size() != 0)
        {
            channelDetails.put("DeviceGroups", dgs_);
        }

        thisChannel_ = null;
        clusters_ = new ArrayList();
        dgs_ = new ArrayList();
        return;
    }
}

public void parse(InputSource source)
    throws IOException, SAXException
{
    adapter_.parse(source);
}

public void startElement(String aUri,
                        String aLocalName,
                        String aQName,
                        Attributes anAttributes)
    throws SAXException
{
    if (aQName.equals("message"))
    {
        status_ = anAttributes.getValue("status");
        message_ = anAttributes.getValue("message");
        return;
    }

    if (aQName.equals("syntax"))
    {
        syntax_ = anAttributes.getValue("syntax");
        return;
    }

    if (aQName.equals("channelProvisioning"))
    {
        action_ = anAttributes.getValue("action");
        return;
    }

    if (aQName.equals("listing"))
    {
        action_ = anAttributes.getValue("action");

        String countString = anAttributes.getValue("count");
        try
        {
            count_ = Integer.parseInt(countString);
        }
        catch (NumberFormatException nfe) {}
        return;
    }
}
```

```
        if (aQName.equals("record"))
        {
            if (action_.equals("getChannels"))
                processChannelDetails(anAttributes);

            return;
        }

        if (aQName.equals("entry"))
        {
            if (action_.equals("getChannels"))
                processAssignmentDetails(anAttributes);

            return;
        }
    }

    public HashMap getChannels()
    {
        return channels_;
    }

    public int getCount()
    {
        return count_;
    }

    public String getMessage()
    {
        return message_;
    }

    public String getStatus()
    {
        return status_;
    }

    private void processAssignmentDetails(Attributes anAttributes)
    {
        {
            String id = anAttributes.getValue("id");
            if (id.startsWith("Cluster"))
            {
                clusters_.add(id);
            }
            else if (id.startsWith("Device"))
            {
                dgs_.add(id);
            }
        }
    }

    private void processChannelDetails(Attributes anAttributes)
    {
        {
            HashMap channelDetails = new HashMap();

            String value = anAttributes.getValue("Name");
            channelDetails.put("Name", value);

            value = anAttributes.getValue("WebSiteId");
            channelDetails.put("WebSiteId", value);

            value = anAttributes.getValue("WeakCertification");
            channelDetails.put("WeakCertification", value);

            value = anAttributes.getValue("SkipEncryption");
            channelDetails.put("SkipEncryption", value);

            value = anAttributes.getValue("Description");
            channelDetails.put("Description", value);
        }
    }
}
```

```
value = anAttributes.getValue("ManifestUrl");
channelDetails.put("ManifestUrl", value);

value = anAttributes.getValue("CacheQuota");
channelDetails.put("CacheQuota", value);

value = anAttributes.getValue("ManifestTTL");
channelDetails.put("ManifestTTL", value);

value = anAttributes.getValue("ManifestUser");
channelDetails.put("ManifestUser", value);

value = anAttributes.getValue("ManifestPassword");
channelDetails.put("ManifestPassword", value);

value = anAttributes.getValue("CheckManifest");
channelDetails.put("CheckManifest", value);

value = anAttributes.getValue("RootCe");
channelDetails.put("RootCe", value);

value = anAttributes.getValue("Id");
channels_.put(value, channelDetails);
thisChannel_ = value;
}
}
```

API エラー メッセージ

API の呼び出し中にサーバエラーが発生した場合、XML 形式のメッセージが返されます。たとえば、Internal Server Error (サーバ内部のエラー) 500 が起きた場合、クライアントに次のメッセージが出力されます。

```
<?xml version="1.0"?>
<Error>
<message status="fail" message="Internal Server Error - 500"/>
</Error>
```

メッセージ構文でサポートされる共通エラーは、次のとおりです。

Bad Request — 400

Authorization Required — 401

Forbidden — 403

File Not Found — 404

Request Timeout — 408

Internal Server Error — 500

通常、API がエラーメッセージを返すのは、API の実行が失敗した場合です。API の実行が正常に行われた場合は、API はエラーメッセージを返しません。ただし、API の実行が正常に行われた場合でも、API は警告メッセージを返すことがあります。

API は、数値によるエラーコードおよび警告コードを返します。表 1-2 では、エラーおよび警告に使用される汎用の数値コードについて説明します。表 1-3 では、プログラム API のエラーおよび警告の数値コードについて説明します。

表 1-2 API のエラーと警告を示す数値コード

エラーコードまたは警告コード	説明
0	なし
1	構文エラー
2	入力エラー
3	制約エラー
4	入力警告

表 1-3 プログラム API のエラーと警告を示す数値コード

エラーコードまたは警告コード	説明
101	プログラム ファイルを取得できません。
102	プログラム ファイル構文エラー
103	プログラム ファイル内の無効な値
104	関連するシステムエラー
105	プログラム ファイル未使用入力 — 警告

たとえば、次の URL を入力して、選択されたタイプのプログラムを削除するために API を実行するとき、そのタイプのプログラムが存在しない場合

```
https://<cdm:port>/servlet/com.cisco.unicorn.ui.ProgramApiServlet?action=deletePrograms&program=type=tvout
```


API は次のような警告を返します。

```
<?xml version="1.0" ?>
<programApi action="deletePrograms">
<message status="success" message="The program(s) are deleted." />
<warning code="4" message="No Program(s) that matched the request were found" />
</programApi>
```

同様に、次の URL を入力して、無効なチャンネル ID を持つチャンネルを削除するために、API を実行する場合

```
https://<cdm:port>/servlet/com.cisco.unicorn.ui.ChannelApiServlet?action=deleteChannels&channel=Channel_333
```

API は次のようなエラーを返します。

```
<?xml version="1.0" ?>
<channelProvisioning action="deleteChannels">
<message status="fail" message="Input Error: Cannot locate channel using channel ID Channel_333" />
<error code="2" message="Input Error: Cannot locate channel using channel ID Channel_333" />
</channelProvisioning>
```

API タスク

次の項では、複製状況、プロビジョニング、プログラム、リスト表示および統計情報 API が実行するタスクについて説明します。

複製状況 API

複製状況 API は、次に示すタスクの 1 つまたは複数を実行します。

- 指定のチャンネル上のコンテンツの複製状況を取得する。
- 指定のチャンネルに割り当てられたすべての Content Engine のコンテンツの複製状況を取得する。
- 指定の Content Engine に割り当てられたすべてのチャンネルのコンテンツの複製状況を取得する。
- 検索基準に関係なく、指定のチャンネル上の指定の Content Engine の複製アイテムをすべてリスト表示する。
- 検索基準に関係なく、指定のチャンネル上の指定の Content Engine の非複製のアイテムをすべてリスト表示する。
- 検索基準に関係なく、指定のチャンネル上の Content Engine のコンテンツ アイテムをすべてリスト表示する。

プロビジョニング API

プロビジョニング API には、チャンネル プロビジョニング API、ロケーション プロビジョニング、Content Engine プロビジョニング API、およびプログラム API があります。

チャンネル プロビジョニング API

チャンネル プロビジョニング API は、次に示すタスクの 1 つまたは複数を実行します。

- チャンネルを作成する。
- 指定のチャンネルにマニフェスト ファイルを追加する。
- 指定のチャンネルに Content Engine を割り当てる。
- 指定のチャンネルにデバイス グループを割り当てる。
- マニフェスト ファイルを即時に取得する。
- チャンネル設定値を変更する。
- マニフェスト ファイルの設定値を変更する。
- 指定のチャンネルから Content Engine を削除する。
- 指定のチャンネルからデバイス グループを削除する。
- チャンネルを削除する。
- コンテンツ プロバイダーを作成する。
- コンテンツ プロバイダーの設定値を変更する。
- コンテンツ プロバイダーを削除する。
- Web サイトを作成する。
- Web サイトの設定値を変更する。
- Web サイトの設定値を削除する。

ロケーション プロビジョニング API

ロケーション プロビジョニング API は、次に示すタスクの 1 つまたは複数を実行します。

- 指定のロケーションを作成する。
- 指定のロケーションを変更する。
- 指定のロケーションを削除する。

Content Engine プロビジョニング API

Content Engine プロビジョニング API は、次に示すタスクの 1 つまたは複数を実行します。

- 指定の Content Engine をアクティブ化する。
- 指定の Content Engine のロケーションを変更する。
- 指定の Content Engine を削除する。

プログラム API

プログラム API は、次に示すタスクの 1 つまたは複数を実行します。

- プログラム ファイルを作成する。
- プログラム ファイルを検証する。
- 指定のプログラムにチャンネルを割り当てる。
- 指定のプログラムに Content Engine を割り当てる。
- 指定のプログラムにデバイス グループを割り当てる。
- プログラム ファイルを取得する。
- プログラム ファイルを変更する。
- 指定のプログラムからチャンネルを削除する。
- 指定のプログラムから Content Engine を削除する。
- 指定のプログラムからデバイス グループを削除する。

リスト表示 API

リスト表示 API は、次に示すタスクの 1 つまたは複数を実行します。

- 選択された Web サイト名と関連コンテンツ プロバイダー名、またはすべての Web サイトをリスト表示する。
- 選択されたチャンネル名と関連 Web サイト ID、またはすべてのチャンネルをリスト表示する。
- 選択された Content Engine 名、またはすべての Content Engine をリスト表示する。
- 指定の Content Engine のロケーションをリスト表示する。
- 選択されたクラスタ名、またはすべてのクラスタをリスト表示する。
- 選択されたコンテンツ プロバイダー名を、またはすべてのコンテンツ プロバイダーをリスト表示する。
- 選択されたデバイス グループ名、またはすべてのデバイス グループをリスト表示する。
- 1 つのデバイスまたはデバイス グループの状況をリスト表示する。
- スtring ID に基づいてオブジェクトをリスト表示する。
- オブジェクト名に基づいてオブジェクトをリスト表示する。
- 指定のすべてのプログラムをリスト表示する。
- プログラムによって現在使用されているすべてのマルチキャスト アドレスをリスト表示する。

- 現在使用中のすべてのマルチキャストアドレスをリスト表示する。
- プログラム用に予約されたマルチキャストアドレス範囲をリスト表示する。

統計情報 API

統計情報 API には、モニタリング統計情報 API とストリーミング統計情報 API があります。

モニタリング統計情報 API

モニタリング統計情報 API は、次に示すタスクの 1 つまたは複数を実行します。

- 各 Content Engine のモニタリング統計情報を取得する。
- 1 つのロケーション内のすべての Content Engine のモニタリング統計情報を取得する。
- ACNS ネットワーク内のすべての Content Engine のモニタリング統計情報を取得する。

ストリーミング統計情報 API

ストリーミング統計情報 API は、次に示すタスクの 1 つまたは複数を実行します。

- 各 Content Engine またはデバイス グループの HTTP 統計情報をレポートする。
- 各 Content Engine またはデバイス グループの Cisco Streaming Engine の統計情報をレポートする。
- 各 Content Engine またはデバイス グループの WMT 統計情報をレポートする。
- 各 Content Engine またはデバイス グループの RealProxy の統計情報をレポートする。