CHAPTER 4

Component APIs

In addition to the primary phone XSI API, the following two additional component APIs are available:

- Application Management API
- RTP Streaming API

This section contains information on the following topics:

Supported Phone Models

Table 1 lists the Cisco Unified IP Phone models that support the component APIs.

Table legend:

- —Supported
- —Not supported

Table 1 Phone Models that Support the Component APIs

Phone Model	Supported/Not Supported	Firware Supported[1]
9900 Series Phones		
9971		9.1(1) or later—
9951		9.1(1) or later—
8900 Series Phones		
8961		9.1(1) or later—
7900 Series Phones		
7905G		—
7906G		8.3(2) or later
7911G		8.3(2) or later
7912G		—
7931G		8.3(2) or later
7937G		—

Table 1 Phone Models that Support the Component APIs

Phone Model	Supported/Not Supported	Firware Supported[1]
7940G	✗	—
7941G/7941G-GE	✓	8.3(2) or later
7942G	✓	8.3(2) or later
7945G	✓	8.3(2) or later
7960G	✗	—
7961G/7961G-GE	✓	8.3(2) or later
7962G	✓	8.3(2) or later
7965G	✓	8.3(2) or later
7970G	✓	8.3(2) or later
7971G-GE	✓	8.3(2) or later
7975G	✓	8.3(2) or later
7985G	✗	—
7900 Series Wireless Phones		
7920G	✗	—
7921G	✗	—
7925G/7925G-EX	✗	—
7926G	✗	—
6900 Series Phones		
6921	✗	—
6941	✗	—
6945	✗	—
6961	✗	—
Other Devices		
Cisco IP Phone Communicator	✓	7.0 or later

1. Cisco recommends the use of latest firmware. The firmware can be downloaded from the following location (requires login and/or service contract): http://tools.cisco.com/support/downloads/pub/Redirect.x?mdfid=278875240.

Application Management API

To address the limited application management, the Application Management API provides a smoother hand-off between the call mode and the application mode. The Application API consists of two primary components:

- Application URI—see the "Application" section on page 5-20
- Application Event Handlers—see the "Application Event Handlers" section on page 3-31

Note Support for the Application Management API requires an updated XML Parser (see "Updated XML Parser and Schema Enforcement" section on page B-1 for details).

RTP Streaming API

This XML-based RTP Streaming API allows applications to initiate and observe RTP audio streams. It extends capabilities beyond the legacy RTP streaming URIs by providing support for stream start/stop event listeners and the ability to specify other extended stream attributes, such as codec type.

Note Support for the RTP Streaming API requires an updated XML Parser (see "Updated XML Parser and Schema Enforcement" section on page B-1 for details).

The event handlers typically use the standard Notification framework (see "Notify" section on page 5-18), but they can also invoke most other URIs, with the exception of HTTP URLs.

Interaction Rules with Legacy RTP URI Streams

The RTP Streaming API allows a full-duplex stream (mode=sendReceive) to be setup as a single stream request which simplifies the usage of the API. However, in some cases, this creates some interoperability issues with the legacy RTP URIs because the legacy RTP URIs send and receive streams separately. The interaction rules between legacy RTP URI streams and the new RTP Streaming API are as follows:

- If an RTP Stop URI is invoked, and an RTP Streaming API stream is currently streaming in that same direction, then the entire RTP Streaming API stream is stopped.

  For example, if a full-duplex stream is setup through the RTP Streaming API (mode=sendReceive) and then an RTPTx:Stop URI is invoked, the stream will be stopped in both the send and receive directions (and the onStopped event handler will be called, if present).

- If the **stopMedia** request (from the RTP Streaming API) does not specify a stream ID, then the request will stop all services RTP streams, in any direction (send or receive) and of any type (multicast and unicast). This allows applications using the RTP Streaming API to stop media streams which may have been started by the legacy RTP URIs or by other applications for which a stream ID is not known.

RTP Streaming Schema

Note The **port** number parameter of the `startMedia` request is optional and if it is not specified, the phone selects an available port and returns it in the `startMediaResponse` object. The **port** parameter, if specified, must be an even number in the range of 20480-32768.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Cisco Systems, Inc. (Cisco
Systems, Inc.) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="startMedia">
```

```
  <xs:complexType>
    <xs:all>
      <xs:element name="mediaStream" type="mediaStream"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="stopMedia">
  <xs:complexType>
    <xs:all>
      <xs:element name="mediaStream">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="optional"/>
        </xs:complexType>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="startMediaResponse">
  <xs:complexType>
    <xs:all>
      <xs:element name="mediaStream" type="mediaStream"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="notifyMediaEvent">
  <xs:complexType>
    <xs:all>
      <xs:element name="mediaStream">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:all>
    <xs:attribute name="origin" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="user"/>
          <xs:enumeration value="application"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="type" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="stopped"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:complexType name="mediaStream">
  <xs:all>
    <xs:element name="type">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="audio"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="codec">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="G.711"/>
          <xs:enumeration value="G.722"/>
```

```
            <xs:enumeration value="G.723"/>
            <xs:enumeration value="G.728"/>
            <xs:enumeration value="G.729"/>
            <xs:enumeration value="GSM"/>
            <xs:enumeration value="Wideband"/>
            <xs:enumeration value="iLBC"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="mode">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="send"/>
            <xs:enumeration value="receive"/>
            <xs:enumeration value="sendReceive"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="address">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:minLength value="7"/>
            <xs:maxLength value="15"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="port" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:unsignedShort">
            <xs:minInclusive value="20480"/>
            <xs:maxInclusive value="32768"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:all>
    <xs:attribute name="onStopped" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:minLength value="1"/>
          <xs:maxLength value="256"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="receiveVolume" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="100"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:schema>
```

Error Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="errorResponse">
```

```
    <xs:complexType>
      <xs:all>
        <xs:element name="type">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="InvalidURL"/>
              <xs:enumeration value="InvalidResource"/>
              <xs:enumeration value="InvalidResourceID"/>
              <xs:enumeration value="UnavailableResource"/>
              <xs:enumeration value="InvalidXML"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="data" nillable="true">
          <xs:simpleType>
            <xs:restriction base="xs:string"/>
          </xs:simpleType>
        </xs:element>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Examples

Start Media

- Request

```
HTTP POST /CGI/Execute
<startMedia>
  <mediaStream
      onStopped=”Notify:http:server:80:path/page”
      receiveVolume=”50”>
        <type>audio</type>
        <codec>G.729</codec>
        <mode>sendReceive</mode>
        <address>239.1.2.3</address>
        <port>20480</port>
  </mediaStream>
</startMedia>
```

- Response

```
HTTP200 OK
<mediaStream id=”abc123”/>
```

Stop Media

- Request

```
HTTP POST CGI/Execute
<stopMedia>
  <mediaStream id=”abc123”/>
</stopMedia>
```

- Response

```
HTTP 200 OK
```

If the user terminates the media stream by placing the active audio path on-hook, the following notification is sent:

```
HTTP POST /server/path/page
DATA=<notifyMediaEvent type="stopped" origin="user">
    <mediaStream id="abc123"/>
</notifyMediaEvent>
```

Errors and Responses

Error conditions and responses for the RTP Streaming API include:

Condition	Applicable Methods	HTTP Result Code	Type	Data
Authorization failed	all	401 (Authorization Failed)	N/A	N/A
Request object does not comply with the API's XML schema	all	400 (BadRequest)	InfalidXML	<parser error description>
Media cannot be started because no DSP resources is available to handle the media	startMedia	400 (BadRequest)	Unavailable Resource	No Media Resource Available
Media cannot be stopped because the specified stream ID does not exist	stopMedia	400 (BadRequest)	InvalidResourceID	Unknown Media Stream ID: <streamID>