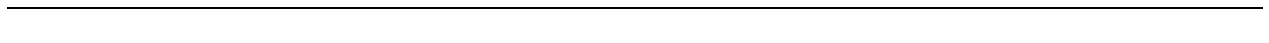




## **SLVoice Application 2.0**

**Initial Release**  
Version 2.0.2961



---

---

By releasing the specification to the SLVoice interfaces, we hope to take another step to help you better utilize the voice and communications services Vivox provides. We look forward to working with and getting feedback from the Second Life Community regarding ways in which Vivox can enrich and expand communications in Second Life in a scalable, high-quality manner.

This manual, including its content and any portion thereof (collectively, the “Content”), is a work protected by copyright and is the sole and exclusive property of Vivox, Inc. (“Vivox”).

The Content is provided “as is” and without any expressed or implied warranties. In no event will Vivox be liable for any direct, indirect, incidental, special, exemplary, or consequential damages, however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of the Content, even if advised of the possibility of such damages.

You can provide feedback regarding issues or suggestions related to this document its use by sending email to [sl-feedback@vivox.com](mailto:sl-feedback@vivox.com),

---

---

## Table Of Contents

<b>1. GETTING STARTED.....</b>	<b>5</b>
1.1 OVERVIEW.....	5
1.2 SETTING UP THE CONNECTION TO THE SLVOICE CLIENT GATEWAY.....	5
<b>2. BACKGROUND.....</b>	<b>7</b>
2.1 OBJECT TYPES.....	7
2.1.1 <i>Accounts</i> .....	8
2.1.2 <i>Channels</i> .....	8
2.1.3 <i>Sessions</i> .....	8
<b>3. XML TEMPLATE MESSAGES.....</b>	<b>9</b>
3.1 REQUEST FORMAT.....	9
3.2 RESPONSE FORMAT.....	9
3.3 EVENT FORMAT.....	9
<b>4. XML MESSAGE CONTENT.....</b>	<b>10</b>
4.1 CONNECTOR RELATED REQUESTS.....	10
4.1.1 <i>Create Connector</i> .....	10
4.1.2 <i>Shutdown Connector</i> .....	11
4.1.3 <i>Mute or unmute the microphone</i> .....	11
4.1.4 <i>Mute or unmute the speaker</i> .....	12
4.1.5 <i>Set microphone volume</i> .....	12
4.1.6 <i>Set local speaker volume</i> .....	13
4.1.7 <i>List Capture Devices</i> .....	14
4.1.8 <i>List Render Devices</i> .....	14
4.1.9 <i>Select Render (Playback) Device</i> .....	15
4.1.10 <i>Select Capture (Input) Device</i> .....	15
4.1.11 <i>Audio Tuning Wizard – Start the Audio Render Process</i> .....	16
4.1.12 <i>Audio Tuning Wizard – Stop the Audio Render Process</i> .....	17
4.1.13 <i>Audio Tuning Wizard – Start the Audio Capture Process</i> .....	17
4.1.14 <i>Audio Tuning Wizard – Stop the Audio Capture Process</i> .....	18
4.1.15 <i>Audio Tuning Wizard – Set Mic Volume</i> .....	18
4.1.16 <i>Audio Tuning Wizard – Set Speaker Volume</i> .....	19
4.2 ACCOUNT RELATED REQUESTS.....	19
4.2.1 <i>Login of user account</i> .....	19
4.2.2 <i>Logout of user account</i> .....	20
4.3 SESSION RELATED REQUESTS.....	21
4.3.1 <i>Create a Session</i> .....	21
4.3.2 <i>Terminate a Session</i> .....	22
4.3.3 <i>Setting positional parameters for local user and other participants</i> .....	22
4.3.4 <i>Set the combined speaking and listening position in 3D space</i> .....	23
<b>5. EVENT MESSAGES.....</b>	<b>25</b>
5.1 ACCOUNT STATE MESSAGES.....	25
5.2 SESSION STATE MESSAGES.....	25
5.2.1 <i>Session new</i> .....	25
5.2.2 <i>Session state change</i> .....	25
5.2.3 <i>Participant state change</i> .....	26
5.2.4 <i>Participant Properties Event</i> .....	26

---

---

5.3	AUDIO TUNING WIZARD EVENTS .....	27
5.3.1	Audio Properties Event.....	27
<b>6.</b>	<b>ERROR CODES.....</b>	<b>28</b>
<b>7.</b>	<b>TYPES AND CODES.....</b>	<b>29</b>
7.1	LOGIN STATE CODES .....	29
7.2	PARTICIPANT TYPE.....	29
7.3	SESSION STATE CODES .....	29
7.4	CONNECTOR STATE CODES .....	29
7.5	AUDIO DEVICE CODES.....	30
7.6	SESSION TYPE CODES .....	30
7.7	ANSWER MODE CODES.....	30
7.8	SESSION TERMINATE REASON CODES .....	31
7.9	PARTICIPANT STATE CODES .....	31

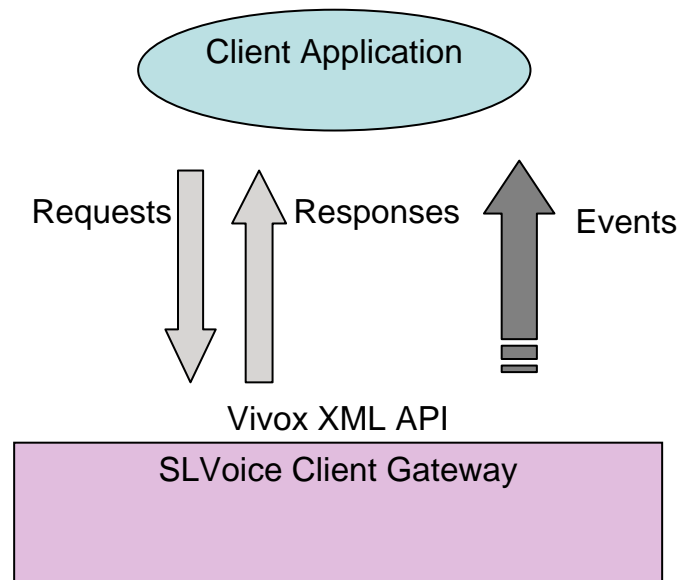
---

---

## 1. Getting Started

### 1.1 Overview

The following diagram illustrates a typical third party “out of process” application integration. In this type of integration, the application sends and receives XML messages to the SLVoice Application process using TCP.



The core object of the SLVoice gateway is a ‘Connector’ object which is the first element that is instantiated and encompasses all other elements. The application sends a **request** and is returned a **response**. Each request-response pair is matched up with a mandatory application generated unique identifier. The response will always include the originating request so that applications may operate with a minimal amount of state – the application can always retrieve the original request from the response.

In addition to sending requests and handling responses, the application must be able to handle SLVoice events. **Events** are generated when there are changes to the local client state. These events may be a direct result of a previous request or they may result from actions occurring elsewhere on the network that effect a change to the local client state.

### 1.2 Setting up the Connection to the SLVoice Client Gateway

In order to access SLVoice functionality, the application must communicate with the SLVoice Client Gateway process using TCP.

Before communicating with the SLVoice Client Gateway process, the application must start that process using the appropriate platform specific method. Before attempting communication with the SLVoice Client Gateway, the start time for this process may vary depending on the activity of the operating system. Typically, it will be available in one or two seconds, but the application should retry the TCP connections to the SLVoice Client Gateway. This will provide time for the SLVoice Client Gateway to initialize and begin listening for incoming requests.

The gateway executable is named as follows for each of the following platforms:

Windows; SLVoice.exe

Mac OS X; SLVoice

---

---

The following set of command line flags can be passed to the SLVoice Client Gateway to control its behavior.

[-h] hide the client gateway window on startup

-c : cleanup - terminates other running instances of the SLVoice client gateway. This is especially useful when debugging.

[-o <address>] process *sends and receives* responses and events to/from this address

For TCP, use the IP address followed by a ':' character and then the port number e.g "127.0.0.1:4000"

When the application communicates with the gateway process via TCP then:

- The process *listens* for incoming connection requests on 127.0.0.1:44124. Subsequent requests are read from the established connection.
- Responses and Events are sent on this same established connection.

---

---

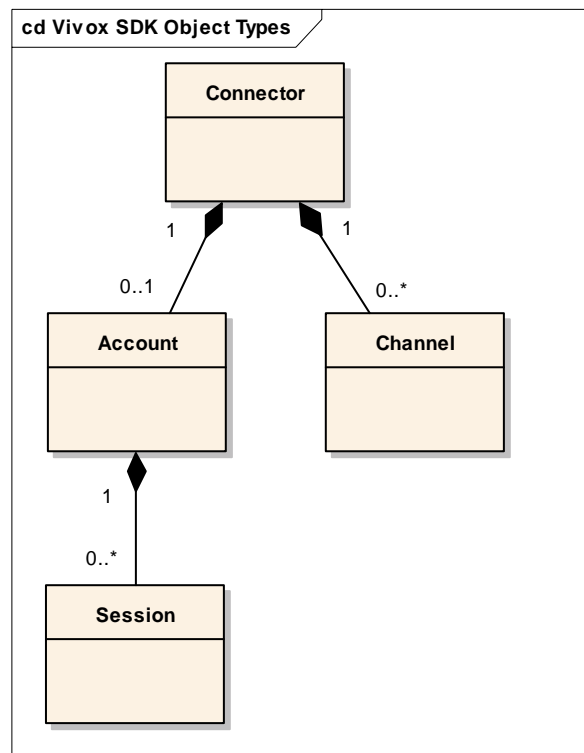
## 2. Background

### 2.1 Object Types

The SLVoice application (through Connector) manages the state of four object types. These are:

1. Connector
2. Account
3. Session
4. Channel

The following UML diagram illustrates the relationship between these objects.



The application is responsible for creating these objects in a particular sequence using the appropriate requests (the specific request and response formats are documented later in this document). Every time an object is created, a handle to that object is returned in the response. This handle can then be used to create other requests which modify the state of these objects. This handle is unique within the application instance, but not across application instances. The application needs to keep and manage the following handles:

1. The current connector handle
2. The current account handle
3. The current session handle

The connector object is the root object, and must be created first. Once a connector object is created successfully, the application will create an account object by logging in. Once logged in, the application can join channels, place calls, and perform many other operations.

---

---

### **2.1.1 Accounts**

In order to interact with any services via the API, 'user' account credentials (e.g. account name and password) are necessary to complete the initial login to network servers. After the Connector is created, an account login is first command call that should be completed. It is within the context of an Account that calls are started, (sessions) and many other user specific actions occur..

### **2.1.2 Channels**

After joining a channel the application may receive notifications (Events) which are conference roster related. Events are generated every time there is some change in the presence state of any participant in the channel. These might include the joining or leaving of a channel, speaking indications and other indications.

### **2.1.3 Sessions**

Sessions are representative of ephemeral connections to zero or more other 'users' in the system. These sessions may or may not have specific media selected when they start. In addition a session may be with a specific user, or with a channel (which provides connectivity to a number of users).

Sessions may be established by the local SLVoice gateway being requested to make a 'call' or by the appearance of a 'call' coming inbound to the SLVoice gateway from the network. These sessions are always within the context of an Account (e.g. the user that is logged in).

---

---

### 3. XML Template messages

All of the xml messages follow a standard format based on one of three types described below.

#### 3.1 Request Format

Issued by the Application to the SLVoice API. The RequestID is set by the calling Application, and its value should be unique within the lifetime of that application instance. Incrementing counters or GUIDs can be used to generate this value. The SLVOICE returns this opaque value in the response associated with this Request.

```
<Request requestId="GUID" action="[context object].[action].[version]">
    [Request Specific XML]
</Request>
```

It is critical that the application provide a unique transaction value in place of the example string "GUID" as shown above. This globally unique ID is utilized by both the SLVoice API and the application to distinguish requests.

Each Request Message **must** be followed by three linefeed characters ("\n\n\n"). This sequence of characters is used by the gateway to determine message boundaries.

#### 3.2 Response Format

Returned to the Application as a result of a specific Request presented by the Application. The requestId, action, and InputXml will be that of the input Request message and thus the Application can be operated with little state required.

```
<Response requestId="GUID" action="[context object].[action].[version]">
    <ReturnCode>0</ReturnCode>
    <Results>
        <StatusCode></StatusCode>
        <StatusString></StatusString>
        [Response Specific XML]
    </Results>
    <InputXml>
        [INPUT request XML message]
    </InputXml>
</Response>
```

#### 3.3 Event Format

Events are presented to the Application to indicate state changes that may affect Application logic or UI elements. These Events may be the delayed result of a request being completed or a change resulting from some remote network element.

```
<Event type="[Event Type]">
    [Event Specific XML]
</Event>
```

---

---

## 4. XML message content

All XML commands within the SLVoice interface context follow straightforward naming syntax to simplify usability and coding by application writers. All of the commands have the following format [context].[action].[version]

### 4.1 Connector related requests

#### 4.1.1 Create Connector

This is used to initialize and stop the Connector as a whole. The Connector Create call must be completed successfully before any other requests are made (typically during application initialization). The shutdown should be called when the application is shutting down to gracefully release resources.

##### Request:

Parameters:

- **ClientName:** A string value indicating the Application name
- **AccountManagementServer:** URL for the management server
- **MinimumPort:** This value must be supplied. If both MinimumPort and MaximumPort are set to '0' the SLVoice Client Gateway will select random open ports. If a range of ports on the client needs to be specified enter the minimum port number and the maximum port number to create a range of ports for the SLVoice to use. The specified range must be at least 8 ports.
- **MaximumPort:** This value must be supplied. If both MinimumPort and MaximumPort are set to '0' the SLVoice Client Gateway will select random open ports. If a range of ports on the client needs to be specified, enter the minimum port number and the maximum port number to create a range of ports for the SLVoice to use.. The specified range must be at least 8 ports.
- **Folder:** The folder where any logs will be created
- **FileNamePrefix:** This will be prepended to beginning of each log file
- **FileNameSuffix:** The suffix or extension to be appended to each log file
- **LogLevel:**
  - 0: NONE - No logging
  - 1: ERROR - Log errors only
  - 2: WARNING - Log errors and warnings
  - 3: INFO - Log errors, warnings and info
  - 4: DEBUG - Log errors, warnings, info and debug

```
<Request requestId="GUID" action="Connector.Create.1">
  <ClientName>V2 SDK</ClientName>
  <AccountManagementServer>http://www.vivox.com/api2</AccountManagementServer>
    <MinimumPort>5000</MinimumPort>
    <MaximumPort>5100</MaximumPort>
  <AttemptStun>
</AttemptStun>
  <Logging>
    <Enabled>>true</Enabled>
    <Folder>~/tmp/logs</Folder>
    <FileNamePrefix>Connector</FileNamePrefix>
    <FileNameSuffix>.log</FileNameSuffix>
    <LogLevel>10</LogLevel>
  </Logging>
</Request>
```

##### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error

- **VersionID:** Version number of the SDK
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status. This is not present
- **ConnectorHandle:** Valid on success, handle value for this initialized Connector instance

```
<Response requestId="GUID" action="Connector.Create.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <VersionID>1</VersionID>
    <StatusCode>0</StatusCode>
    <StatusString>Status OK</StatusString>
    <ConnectorHandle>c1_m1000</ConnectorHandle>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

#### 4.1.2 Shutdown Connector

##### Request:

Parameters:

- **ConnectorHandle:** Handle returned from successful Connector ‘create’ request

```
<Request requestId="GUID" action="Connector.InitiateShutdown.1">
  <ConnectorHandle>c1_m1000</ConnectorHandle>
</Request>
```

##### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Connector.InitiateShutdown.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString></StatusString>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

#### 4.1.3 Mute or unmute the microphone

##### Request:

Parameters:

- **ConnectorHandle:** Handle returned from successful Connector ‘create’ request
- **Value:** A number, either true (mute) and false (unmute).

```
<Request requestId="GUID" action="Connector.MuteLocalMic.1">
  <ConnectorHandle>connector-handle</ConnectorHandle>
  <Value>true</Value>
</Request>
```

---

---

## Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Connector.MuteLocalMic.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>Status OK</StatusString>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

### 4.1.4 Mute or unmute the speaker

This is used to mute or unmute the speaker

#### Request:

Parameters:

- **ConnectorHandle:** Handle returned from successful Connector ‘create’ request
- **Value:** A number, either true (mute) and false (unmute).

```
<Request requestId="GUID" action="Connector.MuteLocalSpeaker.1">
  <ConnectorHandle>connector-handle</ConnectorHandle>
  <Value>true</Value>
</Request>
```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Connector.MuteLocalSpeaker.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>Status OK</StatusString>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

### 4.1.5 Set microphone volume

#### Request:

Parameters:

- **ConnectorHandle:** Handle returned from successful Connector ‘create’ request

- 
- 
- **Level:** The level of the audio, a number between -100 and 100 where 0 represents ‘normal’ speaking volume

```
<Request requestId="GUID" action="Connector.SetLocalMicVolume.1">
  <ConnectorHandle>connector-handle</ConnectorHandle>
  <Value>50</Value>
</Request>
```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Connector.SetLocalMicVolume.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>Status OK</StatusString>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

#### 4.1.6 Set local speaker volume

##### Request:

Parameters:

- **ConnectorHandle:** Handle returned from successful Connector ‘create’ request
- **Level:** The level of the audio, a number between -100 and 100 where 0 represents ‘normal’ speaking volume

```
<Request requestId="GUID" action="Connector.SetLocalSpeakerVolume.1">
  <ConnectorHandle>connector-handle</ConnectorHandle>
  <Value>50</Value>
</Request>
```

##### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Connector.SetLocalSpeakerVolume.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>Status OK</StatusString>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

---

---

### 4.1.7 List Capture Devices

This is used to get a list of audio devices that can be used for capture (input) of voice.

#### Request:

Parameters: None

```
<Request requestId="GUID" action="Aux.GetCaptureDevices.1">
</Request>
```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status
- **Device:** Zero or more elements with the name of the device
- **CurrentCaptureDevice:** The capture device currently selected.

```
<Response requestId="GUID" action="Aux.GetCaptureDevices.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>status string</StatusString>
    <CaptureDevices>
      <CaptureDevice>
        <Device>SigmaTel Audio</Device>
      </CaptureDevice>
      <CaptureDevice>
        <Device>Logitech Headset</Device>
      </CaptureDevice>
    </CaptureDevices>
    <CurrentCaptureDevice>
      <Device>Logitech Headset</Device>
    </CurrentCaptureDevice>
  </Results>
</InputXml>
<Request requestId="GUID" action="Aux.GetCaptureDevices.1">
</Request>
</InputXml>
</Response>
```

### 4.1.8 List Render Devices

This is used to get a list of audio devices that can be used for render (playback) of voice.

#### Request:

Parameters: None

```
<Request requestId="GUID" action="Aux.GetRenderDevices.1">
</Request>
```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status
- **Device:** Zero or more elements with the name of the device
- **CurrentRenderDevice:** The render device currently selected.

---



---

```

<Response requestId="GUID" action="Aux.GetRenderDevices.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode></StatusCode>
    <StatusString>status string</StatusString>
    <RenderDevices>
      <RenderDevice>
        <Device>SigmaTel Audio</Device>
      </RenderDevice>
      <RenderDevice>
        <Device>Logitech Headset</Device>
      </RenderDevice>
    </RenderDevices>
    <CurrentRenderDevice>
      <Device>Logitech Headset</Device>
    </CurrentRenderDevice>
  </Results>
</InputXml>
  <Request requestId="GUID" action="Aux.GetRenderDevices.1">
  </Request>
</InputXml>
</Response>

```

#### 4.1.9 Select Render (Playback) Device

This command is used to select the render device.

##### Request:

Parameters:

- **RenderDeviceSpecifier:** The name of the device as returned by the Aux.GetRenderDevices command.

```

<Request requestId="GUID" action="Aux.SetRenderDevice.1">
  <RenderDeviceSpecifier>Logitech Headset</RenderDeviceSpecifier>
</Request>

```

##### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```

<Response requestId="GUID" action="Aux.SetRenderDevice.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>status string</StatusString>
  </Results>
</InputXml>
  <Request requestId="GUID" action="Aux.SetRenderDevice.1">
    <RenderDeviceSpecifier>Logitech Headset</RenderDeviceSpecifier>
  </Request>
</InputXml>
</Response>

```

#### 4.1.10 Select Capture (Input) Device

This command is used to select the capture device.

##### Request:

Parameters:

- **CaptureDeviceSpecifier:** The name of the device as returned by the Aux.GetCaptureDevices command.

---

---

```
<Request requestId="GUID" action="Aux.SetCaptureDevice.1">
  <CaptureDeviceSpecifier>Logitech Headset</CaptureDeviceSpecifier>
</Request>
```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Aux.SetCaptureDevice.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>100</StatusCode>
    <StatusString>status string</StatusString>
  </Results>
  <InputXml>
    <Request requestId="GUID" action="Aux.SetCaptureDevice.1">
      <CaptureDeviceSpecifier>something</CaptureDeviceSpecifier>
    </Request>
  </InputXml>
</Response>
```

#### 4.1.11 Audio Tuning Wizard – Start the Audio Render Process

This command is used to start the audio render process, which will then play the passed in file through the selected audio render device. This command should not be issued if the user is on a call.

#### Request:

Parameters:

- **SoundFilePath:** The fully qualified path to the sound file.
- **Loop:** 1 if the file is to be played continuously and 0 if it should be played once.

```
<Request requestId="GUID" action="Session.RenderAudioStart.1">
  <SoundFilePath>c:\temp\file.wav</SoundFilePath>
  <Loop>1</Loop>
</Request>
```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Session.RenderAudioStart.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>100</StatusCode>
    <StatusString>status string</StatusString>
  </Results>
  <InputXml>
    <Request requestId="GUID" action="Session.RenderAudioStart.1">
      <SoundFilePath>c:\temp\file.wav</SoundFilePath>
      <Loop>1</Loop>
    </Request>
  </InputXml>
</Response>
```

---

---

#### 4.1.12 Audio Tuning Wizard – Stop the Audio Render Process

This command is used to stop the audio render process.

##### Request:

Parameters:

- **SoundFilePath:** The fully qualified path to the sound file issued in the start render command.

```
<Request requestId="GUID" action="Session.RenderAudioStop.1">
<SoundFilePath>c:\temp\file.wav</SoundFilePath>
</Request>
```

##### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Session.RenderAudioStop.1">
<ReturnCode>0</ReturnCode>
<Results>
  <StatusCode>100</StatusCode>
  <StatusString>status string</StatusString>
</Results>
<InputXml>
  <Request requestId="GUID" action="Session.RenderAudioStop.1">
    <SoundFilePath>c:\temp\file.wav</SoundFilePath>
  </Request>
</InputXml>
</Response>
```

#### 4.1.13 Audio Tuning Wizard – Start the Audio Capture Process

This command is used to start the audio capture process which will cause AuxAudioProperty Events to be raised. These events can be used to display a microphone VU meter for the currently selected capture device. This command should not be issued if the user is on a call.

##### Request:

Parameters:

- **Duration:** (unused but required )

```
<Request requestId="GUID" action="Aux.CaptureAudioStart.1">
<Duration>100</Duration>
</Request>
```

##### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Aux.CaptureAudioStart.1">
<ReturnCode>0</ReturnCode>
<Results>
  <StatusCode>100</StatusCode>
  <StatusString>status string</StatusString>
</Results>
```

---

---

```
</Results>
<InputXml>
  <Request requestId="GUID" action="Aux.CaptureAudioStart.1">
    <Duration>100</Duration>
  </Request>
</InputXml>
</Response>
```

#### 4.1.14 Audio Tuning Wizard – Stop the Audio Capture Process

This command is used to stop the audio capture process.

##### Request:

```
<Request requestId="GUID" action="Aux.CaptureAudioStop.1">
</Request>
```

##### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Aux.CaptureAudioStop.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>100</StatusCode>
    <StatusString>status string</StatusString>
  </Results>
  <InputXml>
    <Request requestId="GUID" action="Aux.CaptureAudioStop.1">
      </Request>
    </InputXml>
  </Response>
```

#### 4.1.15 Audio Tuning Wizard – Set Mic Volume

This command is used to set the mic volume while in the audio tuning process. Once an acceptable mic level is attained, the application must issue a connector set mic volume command to have that level be used while on voice calls.

##### Request:

Parameters:

- **Level** – the microphone volume (-100 to 100 inclusive)

```
<Request requestId="GUID" action="Aux.SetMicLevel.1">
  <Level>2</Level>
</Request>
```

##### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Aux.SetMicLevel.1">
  <ReturnCode>0</ReturnCode>
```

---

---

```
<Results>
  <StatusCode>100</StatusCode>
  <StatusString>status string</StatusString>
</Results>
<InputXml>
  <Request requestId="GUID" action="Aux.SetMicLevel.1">
    <Level>20</Level>
  </Request>
</InputXml>
</Response>
```

#### 4.1.16 Audio Tuning Wizard – Set Speaker Volume

This command is used to set the speaker volume while in the audio tuning process. Once an acceptable speaker level is attained, the application must issue a connector set speaker volume command to have that level be used while on voice calls.

##### Request:

Parameters:

- **Level** – the speaker volume (-100 to 100 inclusive)

```
<Request requestId="GUID" action="Aux.SetSpeakerLevel.1">
  <Level>2</Level>
</Request>
```

##### Results:

Parameters:

- **ReturnCode**: Non-0 value indicates error
- **StatusCode**: Numeric value of completion
- **StatusString**: Optional string values to present readable status

```
<Response requestId="GUID" action="Aux.SetSpeakerLevel.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>100</StatusCode>
    <StatusString>status string</StatusString>
  </Results>
  <InputXml>
    <Request requestId="GUID" action="Aux.SetSpeakerLevel.1">
      <Level>20</Level>
    </Request>
  </InputXml>
</Response>
```

## 4.2 Account related requests

### 4.2.1 Login of user account

This is used to login a specific user account(s). It may only be called after Connector initialization has completed successfully.

##### Request:

Parameters:

- **ConnectorHandle**: Handle returned from successful Connector 'create' request
- **AccountName**: Users account name

- **AccountPassword:** Users account password
- **AudioSessionAnswerMode:** Values may be “AutoAnswer” or “VerifyAnswer”
- **ParticipantPropertyFrequency:** This is an integer that specifies how often the SLVoice will send participant property events while in a channel. If this is not set the default will be “on state change”, which means that the events will be sent when the participant starts talking, stops talking, is muted, is unmuted. The valid values are:
  - **0** – Never
  - **5** – 10 times per second
  - **10** – 5 times per second
  - **50** – 1 time per second
  - **100** – on participant state change (this is the default)

*note: it is recommended that you use the highest participant\_property\_frequency possible for your implementation. If you are showing a speaking energy indicator in your UI you should use either 10 or 50. If you need a more accurate energy meter you may use 5.*

```
<Request requestId="GUID" action="Account.Login.1">
  <ConnectorHandle>c1_m1000</ConnectorHandle>
  <AccountName>bruno</AccountName>
  <AccountPassword>password</AccountPassword>
  <AudioSessionAnswerMode>AutoAnswer</AudioSessionAnswerMode>
  <ParticipantPropertyFrequency>100</ParticipantPropertyFrequency>
</Request>
```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status
- **AccountHandle:** Valid on success, handle value for this initialized account login

```
<Response requestId="GUID" action="Account.Login.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>OK</StatusString>
    <AccountHandle>c1_m1000joe</AccountHandle>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

#### 4.2.2 Logout of user account

This is used to logout a user session. It should only be called with a valid AccountHandle.

#### Request:

Parameters:

- **AccountHandle:** Handle returned from successful Connector ‘login’ request

```
<Request requestId="GUID" action="Account.Logout.1">
  <AccountHandle>account-handle</AccountHandle>
</Request>
```

---

---

## Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```
<Response requestId="GUID" action="Account.Logout.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>OK</StatusString>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

## 4.3 Session related requests

### 4.3.1 Create a Session

Sessions typically represent a connection to a media session with one or more participants. This is used to generate an ‘outbound’ call to another user or channel. The specifics depend on the media types involved. A session handle is required to control the local user functions within the session (or remote users if the current account has rights to do so). Currently creating a session automatically connects to the audio media, there is no need to call Session.Connect at this time, this is reserved for future use.

#### Request:

Parameters:

- **AccountHandle:** Handle returned from successful Connector ‘login’ request
- **URI:** This is the URI of the terminating point of the session (ie who/what is being called)
- **Name:** This is the display name of the entity being called (user or channel)
- **Password:** Only needs to be supplied when the target URI is password protected
- **PasswordHashAlgorithm:** This indicates the format of the password as passed in. This can either be “ClearText” or “SHA1UserName”. If this element does not exist, it is assumed to be “ClearText”. If it is “SHA1UserName”, the password as passed in is the SHA1 hash of the password and username concatenated together, then base64 encoded, with the final “=” character stripped off.

```
<Request requestId="GUID" action="Session.Create.1">
  <AccountHandle>account-handle</AccountHandle>
  <URI>sip:bruno@foo.com</URI>
  <Name>Bruno</Name>
  <Password>mypassword</Password>
  <PasswordHashAlgorithm>SHA1UserName</PasswordHashAlgorithm>
</Request>
```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status
- **SessionHandle:** Valid on success, handle value for this created session

---



---

```

<Response requestId="GUID" action="Session.Create.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString></StatusString>
    <SessionHandle>session-handle</SessionHandle>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>

```

### 4.3.2 Terminate a Session

This is used to ‘end’ an established session (i.e. hang-up or disconnect).

#### Request:

Parameters:

- **SessionHandle:** Handle returned from successful Session ‘create’ request or a SessionNewEvent

```

<Request requestId="GUID" action="Session.Terminate.1">
  <SessionHandle>session-handle</SessionHandle>
</Request>

```

#### Results:

Parameters:

- **ReturnCode:** Non-0 value indicates error
- **StatusCode:** Numeric value of completion
- **StatusString:** Optional string values to present readable status

```

<Response requestId="GUID" action="Session.Terminate.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString> </StatusString>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>

```

### 4.3.3 Setting positional parameters for local user and other participants

These requests are used to effectively set the ‘ears’ and ‘mouth’ positions of the user in the specified session. The reason for allowing these to be different is that a view angle may be changed for an associate avatar without actually changing its location in a 3D virtual world. In other words a view may ‘zoom in’ or ‘zoom out’ with out affecting the perspective of other participants in a session.

#### Request:

Parameters:

- **SessionHandle:** Handle returned from successful Session ‘create’ request or a SessionNewEvent
- **Position:** Positional vector of the users position
- **Velocity:** Velocity vector of the position
- **AtOrientation:** At Orientation (X axis) of the position

- **UpOrientation:** Up Orientation (Y axis) of the position
- **LeftOrientation:** Left Orientation (Z axis) of the position

#### 4.3.4 Set the combined speaking and listening position in 3D space

```

<Request requestId="GUID" action="Session.Set3DPosition.1">
  <SessionHandle>session-handle</SessionHandle>
  <SpeakerPosition>
    <Position>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </Position>
    <Velocity>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </Velocity>
    <AtOrientation>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </AtOrientation>
    <UpOrientation>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </UpOrientation>
    <LeftOrientation>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </LeftOrientation>
  </SpeakerPosition>
  <ListenerPosition>
    <Position>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </Position>
    <Velocity>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </Velocity>
    <AtOrientation>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </AtOrientation>
    <UpOrientation>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </UpOrientation>
    <LeftOrientation>
      <X>1</X>
      <Y>2</Y>
      <Z>3</Z>
    </LeftOrientation>
  </ListenerPosition>
</Request>

```

Set User Volume for a particular user. Does not affect how other users hear that user.

```

<Request requestId="GUID" action="Session.SetParticipantVolumeForMe.1">
  <SessionHandle>account-handle</SessionHandle>
  <ParticipantURI>sip:mikes@foo.com</ParticipantURI>

```

---

---

```
    <Volume>-100</Volume>
</Request>
<Response requestId="GUID" action="Session.SetParticipantVolumeForMe.1">
  <ReturnCode>0</ReturnCode>
  <Results>
    <StatusCode>0</StatusCode>
    <StatusString>Status OK</StatusString>
  </Results>
  <InputXml>
    [INPUT request XML message]
  </InputXml>
</Response>
```

---

---

## 5. Event Messages

Event messages are generated by the SLVoice and indicate a change in internal state for a particular context (e.g. connector, account, session, participant). In some cases this might cause the Application to update its UI (i.e. a roster list of channel participants) or the Application may complete some other action based upon the event status.

Events are passed through the same process as Responses from Request requests.

### 5.1 Account State messages

This event message is sent whenever the login state of the particular Account has transitioned from one value to another.

#### Event:

Parameters:

- **AccountHandle:** Handle returned from successful Account request .
- **StatusCode:** Code used to identify why a state change has been made
- **StatusString:** Text (in English) to describe the Status Code
- **State:** New state of the entity, please refer to the codes at the end of this doc for specific state codes

```
<Event type="LoginStateChangeEvent">
  <AccountHandle>account-handle</AccountHandle>
  <StatusCode>100</StatusCode>
  <StatusString>status-string</StatusString>
  <State>2</State>
</Event>
```

### 5.2 Session State messages

#### 5.2.1 Session new

Sent when an incoming session occurs.

#### Event:

Parameters:

- **AccountHandle:** Handle returned from successful Connector 'login' request
- **SessionHandle:** Handle created for inbound session
- **URI:** Full URI of the session (user/channel)
- **Name:** Name of the channel if IsChannel is true
- **IsChannel:** set to "true" if this session relates to a Channel, "false" if not
- **AudioMedia:** set to type of audio media

```
<Event type="SessionNewEvent">
  <AccountHandle>account-handle</AccountHandle>
  <SessionHandle>session-handle</SessionHandle>
  <URI>sip:confctl-6@foo.com</URI>
  <IsChannel>true</IsChannel>
  <Name>Channel 1</Name>
  <AudioMedia>default</AudioMedia>
</Event>
```

#### 5.2.2 Session state change

Sent for specific Session state changes (connected, disconnected)

#### Event:

---

---

Parameters:

- **SessionHandle:** Handle returned from successful Session ‘create’ request or a SessionNewEvent
- **StatusCode:** Code used to identify why a state change has been made
- **StatusString:** Text (in English) to describe the Status Code
- **State:** New state of the entity, please refer to the codes at the end of this doc for specific state codes
- **URI:** Full URI of the session (user/channel?)
- **IsChannel:** set to “true” if this session relates to a Channel, “false” if not
- **ChannelName:** The name of the channel

```
<Event type="SessionStateChangeEvent">
  <SessionHandle>session-handle</SessionHandle>
  <StatusCode>100</StatusCode>
  <StatusString>status-string</StatusString>
  <State>2</State>
  <URI>sip:confctl-6@foo.com</URI>
  <IsChannel>true</IsChannel>
  <ChannelName>channel-name</ChannelName>
</Event>
```

### 5.2.3 Participant state change

Sent for specific Participant state changes (new participants, dropped participants)

**Event:**

Parameters:

- **SessionHandle:** Handle returned from successful Session ‘create’ request or a SessionNewEvent
- **StatusCode:** Code used to identify why a state change has been made
- **StatusString:** Text (in English) to describe the Status Code
- **State:** New state of the entity, please refer to the codes at the end of this doc for specific state codes
- **ParticipantURI:** The URI of the participant whose state has changed
- **AccountName:** The Account name of the Participant
- **DisplayName:** The display name of the participant (may or may not be available)
- **ParticipantType:** The type of the participant (user, moderator, etc...), please refer to the codes at the end of this doc for specific type codes

```
<Event type="ParticipantStateChangeEvent">
  <SessionHandle>session-handle</SessionHandle>
  <StatusCode>100</StatusCode>
  <StatusString>status string</StatusString>
  <State>6</State>
  <ParticipantURI>sip:bruno@foo.com</ParticipantURI>
  <AccountName>Bruno</AccountName>
  <DisplayName>Cool Guy</DisplayName>
  <ParticipantType>0</ParticipantType>
</Event>
```

### 5.2.4 Participant Properties Event

Sent for specific Participant Property changes (IsSpeaking, Volume, Energy, etc...)

**Event:**

Parameters:

- **SessionHandle:** Handle returned from successful Session ‘create’ request or a SessionNewEvent
- **ParticipantURI:** The URI of the participant whose properties are being updated
- **IsLocallyMuted:** Used to determine if the user has locally muted their mic (0 – not muted, 1 – muted)
- **IsModeratorMuted:** Used to determine if the user has been muted by the moderator (0 – not muted, 1 – muted)
- **Volume:** This is the volume level that has been set by the user, this should not change often and is a value between 0 and 100 where 50 represents ‘normal’ speaking volume

- 
- 
- **Energy:** This is the energy, or intensity of the participants audio. This is used to determine how loud the user is speaking. This is a value between 0 and 1.

```
<Event type="ParticipantPropertiesEvent">
  <SessionHandle>session-handle</SessionHandle>
  <ParticipantURI>sip:bruno@foo.com</ParticipantURI>
  <IsLocallyMuted>>false</IsLocallyMuted>
  <IsModeratorMuted>>false</IsModeratorMuted>
  <IsSpeaking>>false</IsSpeaking>
  <Volume>50</Volume>
  <Energy>0.6</Energy>
</Event>
```

## 5.3 Audio Tuning Wizard Events

### 5.3.1 Audio Properties Event

Audio Properties Events are sent after audio capture is started. These events are used to display a microphone VU meter.

#### Event:

Parameters:

- **MicIsActive:** 1 is voice is detected on the microphone
- **MicEnergy:** audio energy, from 0 to 1 (floating point)
- **MicVolume:** current mic volume
- **SpeakerVolume:** currently unimplemented, and always 0

```
<Event type="AuxAudioPropertiesEvent">
  <MicIsActive>>true</MicIsActive>
  <MicEnergy>0.1</MicEnergy>
  <MicVolume>1</MicVolume>
  <SpeakerVolume>0</SpeakerVolume>
</Event>
```

---

---

## 6. Error Codes

Status Code	Status String
1	'Unknown Error'
200	'Wrong user credentials'
201	'Expired user credentials'
202	'Missing user credentials'
203	'User not logged in'
210	'Not logged in as an administrator'
211	'Not logged in as channel moderator'
220	'Failed retrieving directory'
300	'Required parameter(s) missing or invalid'
301	'Unknown mode'
302	'Unable to parse argument'
403	'Account does not exist'
500	'Invalid Command'
711	'User is not on channel'
1000	'Invalid XML'
1001	'Required Object does not exist'
1002	'Maximum number of connectors exceeded'
1003	'Maximum number of sessions exceeded'
1004	'Operation Failed'
1005	'User already logged in'
1006	'User already logged out'
1007	'User must be logged in to perform this operation'
1008	'Invalid argument'
1009	'Invalid username or password'
20712	Unable to complete channel lookup – channel closed.
20713	Channel Participant Limit Reached.
20202	Missing User Credentials
20203	Account not logged in
20210-20212	Account does not have permissions

---

---

## 7. Types and Codes

### 7.1 Login State Codes

(LoginStateChangeEvent : State)

State Code	State
0	Logged Out
1	Logged In
4	Error

### 7.2 Participant Type

(SessionStateChangeEvent or ParticipantChangeEvent)

State Code	State
0	User
1	Moderator
2	Focus

### 7.3 Session State Codes

(SessionStateChangeEvent : State)

State Code	State
1	Idle
2	Answering
3	In Progress
4	Connected
5	Disconnected
6	Hold
7	Refer
8	Ringling

### 7.4 Connector State Codes

---

---

(ConnectorStateChangeEvent : State)

State Code	State
0	Shutdown
1	Initializing
2	Initialized
3	Shutting down

### 7.5 Audio Device Codes

State Code	State
0	DefaultSpeaker
1	DefaultMicrophone

### 7.6 Session Type Codes

(SessionStateChangeEvent : Type)

State Code	State
0	None
1	AddText
2	AddAudio
3	AddVideo
4	DropText
5	DropAudio
6	DropVideo

### 7.7 Answer Mode Codes

State Code	State
1	Automatically Accept Audio
2	Offer Session Event
3	Automatically Accept AudioVideo
4	Automatically Accept Text

5	Automatically Reject
6	Not Supported

## 7.8 Session Terminate Reason Codes

(SessionStateChangeEvent : State)

State Code	State
0	Normal
1	Do not Disturb (DND)
2	Busy
3	Reject
4	Timeout
5	Shutdown
6	Insufficient Security Level
7	Not Supported

## 7.9 Participant State Codes

(ParticipantStateChangeEvent : State)

State Code	State
1	Idle
2	Pending
3	Incoming
4	Answering
5	In progress
6	Ringling
7	Connected
8	Disconnecting
9	Disconnected