Session API

As described in the previous chapter, Unified CVP provides a mechanism for the developer to access and change information having to do with the phone call or the session. Through this API one can get environment information such as the phone number of the caller (ANI), the time the call began, and application settings such as the default audio path. This API is also the conduit for the developer to set element or session data, send custom logging events, or access the user management system. A subset of the Session API, called the Global API provides access to data that exists beyond individual sessions.

Any custom component built by the developer will be sent this API to interface with the session. This section of the document describes the API and what it can be used for. Both the Java and XML versions of this API are described. Subsequent chapters will detail the APIs used to actually construct components.

- Java API, on page 1
- XML API, on page 2

Java API

As described previously, every Unified CVP component is constructed by implementing a Java interface or extending a Java class and overriding a single method. One argument to this method is a Unified CVP-specified Java class that acts as the API to the session. Methods in this class are used to get or change information stored in the session, such as element or session data.

A different API class is used depending on the component. All API classes are derived from the base class `APIBase`, though all non-logger API classes directly extend `ComponentAPI` (both are found in the `com.audium.server.session` package). `APIBase` defines information retrieval functions any component accessed within a call session can use, such as:

- Obtaining telephony information such as the ANI and DNIS.
- Obtaining application setting data such as the gateway adapter name, default audio path, maintainer, etc.
- Getting element or session data created by components run prior to the current component.
- Retrieving a list of elements and the exit states encountered by the caller prior to the current component.
- Obtaining information on where the current application resides in order to aid in the loading of custom content found there.

`ComponentAPI` adds to this the ability to alter some environment settings:

- Getting access to the User Management system, allowing the component to create, modify, or query information on users.
- Creating session data. This class does not allow the creation of element data because only elements can do so (the start of call class cannot, for example).
- Adding custom content to the activity log and warnings to the error log.
- Triggering custom logging events and warning events that are picked up by loggers.
- Setting the maintainer, default audio path, application language and encoding, as well as the call's session timeout. At any point in the application, these settings can be changed.
- Accessing the Global API to get and set application and global data (See User Guide for Cisco Unified CVP VXML Server and Unified Call Studio for more information on application and global data).

The following bullets list the API classes that are used for various components (also found in the `com.audium.server.session` package). A detailed description of what each class provides is given in the individual section for that component.

- **CallStartAPI** – This class is sent as an argument to the start of call class.
- **CallEndAPI** – This class is sent as an argument to the end of call class.
- **ElementAPI** – This class is used by all standard and configurable element classes as well as dynamic configuration classes. The following classes extend `ElementAPI` to provide additional functionality required for different kinds of elements.
  - **ActionAPI / ActionElementData** – The `ActionAPI` class is used by generic action element classes and is extended by `ActionElementData` which is used by configurable action element classes.
  - **DecisionElementData** – This class is used by configurable decision element classes.
  - **VoiceElementData** – This class is used by configurable voice element classes.
- **LoggerAPI** – This class is sent as an argument to a logger's method for handling a session-specific logging event.

XML API

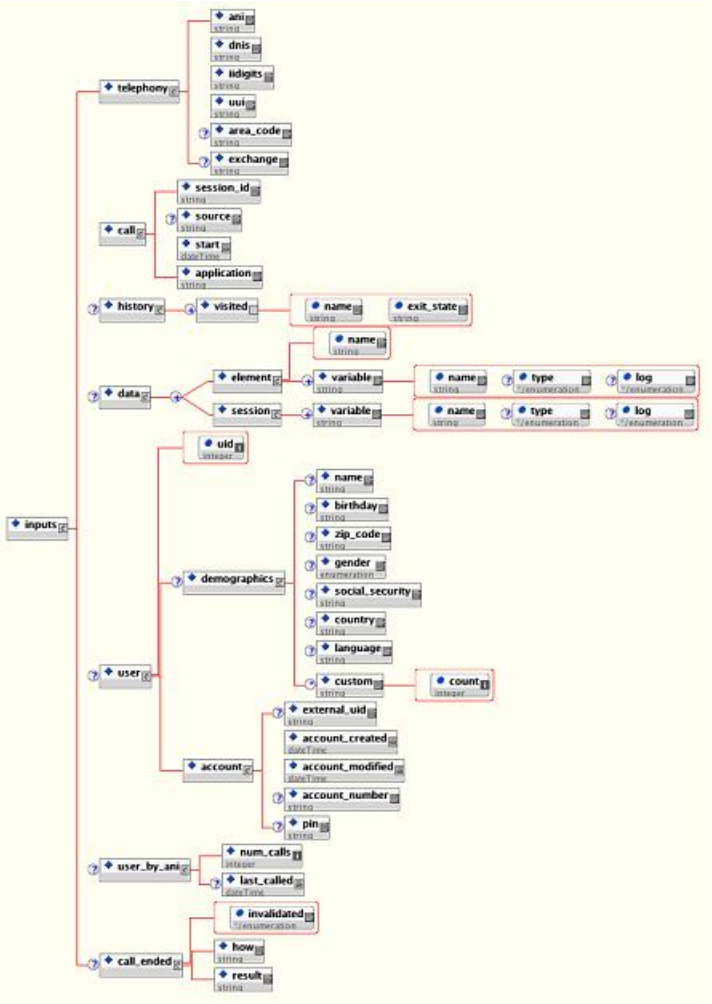
When a component uses the Java API, the Session API is accessed via an object passed to the method. A similar setup exists when a component uses the XML API. The entire contents of the Session API are made available via a set of XML documents passed to the component in the HTTP request. Each component will receive this information whether they need it or not, since VXML Server does not know in advance what information the component could require. The component can choose to ignore these documents if the information contained within are not required, or use a fast, event-based parser to extract only the desired information from the documents.

Each component receives two POST arguments containing complete XML documents representing the Session API. The first, named *inputs*, lists the session information representing the state of the application up to the point when the component was reached. The second argument, named *settings*, lists the current value for the application settings.

XML Document Sent Inputs

The following example is the DTD diagram for the XML document sent to all components in the *inputs* argument. Its DTD is defined in the file `ElementRequest.dtd` in the VXML Server `dtds` folder.

Figure 1: XML Example Diagram for Document Sent to All Components in "inputs"



The tags in this XML document are:

- **telephony** – This tag holds information about the call itself such as the ANI. It also contains the area code and exchange of the ANI. All values are *NA* if not sent by the voice browser (area code and exchange won't appear at all in the case that the ANI is not sent).
- **call** – This tag holds call information. The session ID is used by VXML Server to identify the call. The `<source>` tag, if applicable, contains the name of the application that transferred to this one (the tag does not appear if application visit is a new call). The `<start>` tag contains the time when the call started. The `<application>` tag contains the name of the application.
- **history** – This tag holds the history of elements visited so far in the call. The name and exit state of the element is included as attributes to a `<visited>` tag. Multiple tags are listed in the order in which the

elements were visited in the call. The `<history>` tag will not appear if no elements were visited before this one (that is, the start of the call).

- **data** – This tag holds all the element and session data created so far in the call. The `<element>` tag's `name` attribute holds the name of the element. All the variables created by this element appear in this tag. The `log` attribute indicates whether this variable's value will appear in the activity log file (no session variables appear in the log). The `<data>` tag will not appear if no element or session data exist. If the session data variable holds a Java class, the tag will contain the results of the `toString()` method called on that object.
- **user** – This tag appears only if the application is configured to use the user management system and the call has been associated with a particular UID. The `<demographics>` tag holds the user demographic information. The `<account>` tag contains information about the account such as when it was created and modified, the account number and pin (if applicable), etc. This data appears exactly as in the user database. See User Guide for Cisco Unified CVP VXML Server and Unified Call Studio for more information on user management.
- **user_by_ani** – This tag appears only if the application is configured to use the user management system, though unlike `<user>`, the tag appears even if the call is not associated with a UID. The tag holds information about the number of calls made to this application by the current phone number and the last time a call was received to the application by that number.
- **call_ended** – This tag appears *only* when being sent as a request to an end of call event. It defines how the call ended and the result of the call. The possible content of `<how>` are: *hangup* (the caller hung up), *disconnect* (the application hung up on the caller), *application_transfer* (the application visit ended by transferring to another application), *call_transfer* (a blind transfer took place) and *app_session_complete* (the application visit ended even if the call itself continued - such as via a CTI event).

  The possible content of `<result>` can be *normal*, *max_ports* (the caller hung up while on hold waiting to enter the application), *suspended* (the caller called into a suspended application), *error* (an error occurred during the call), *timeout* (the session timed out) and *invalidated* (the session was invalidated by an element).

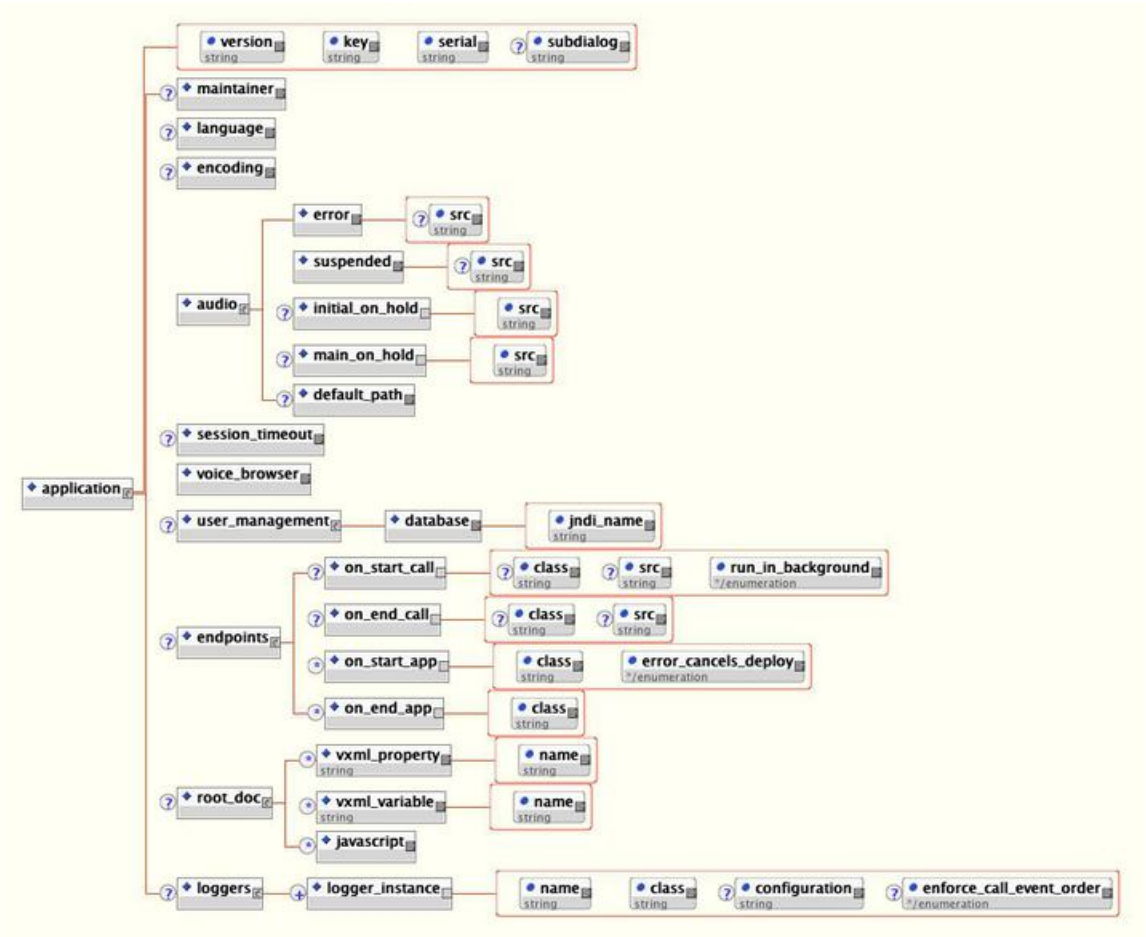
Note The `invalidated` attribute of `<call_ended>` also indicates if the session was invalidated by an element.

XML Document Sent Settings

The following example shows the DTD diagram for the XML document sent to all components in the *settings* argument. Its DTD is defined in the file `Settings.dtd` in the VXML Server `dtds` folder.

Note This document shares the same DTD as the static application settings file `settings.xml` created when an application is deployed from Call Studio to VXML Server. This document is simply an XML representation of the application settings in Call Studio's project preferences for the application.

Figure 2: DTD Diagram for XML Document Sent to All Components in Settings Argument



The tags in this XML document are:

- **application** – The root tag. The `key` and `serial` attributes are used by Call Studio and VXML Server and can be safely ignored here. The `version` attribute holds the version of the file. The above diagram represents version 1.2 of the DTD. The `subdialog` attribute holds a `true` value if the application is accessed as a subdialog, a `false` value if it is not.
- **maintainer** – This tag holds the e-mail address of the maintainer.
- **language** – This tag holds the language for the application. This value shows up in the VoiceXML pages produced by VXML Server. The contents of this tag are formatted according to the specification for using languages in VoiceXML (for example, *en-US*).
- **encoding** – This tag holds the encoding format for the application. This value determines how the VoiceXML pages produced by VXML Server are encoded. The contents of this tag are formatted according to the specification for encoding XML pages (for example, *UTF-8*).
- **audio** – This tag holds all audio files and/or TTS phrases to use for various situations.
  - **error** – This tag encapsulates the message to play when an error occurs and the application does not contain an error element.

- **suspended** – This tag encapsulates the message to play when a caller calls an application that is suspended.
- **initial_on_hold** – This tag encapsulates the first message a caller hears when all the VXML Server ports are in use.
- **main_on_hold** – This tag encapsulates the message repeatedly played after the initial on hold audio is played until a VXML Server port is available.
- **default_path** – This tag lists the URI path in which all pre-recorded audio for this application is located.

- **session_timeout** – This tag lists the length of time in minutes of inactivity VXML Server will time out the call session.
- **voice_browser** – This tag lists the voice browser selected for this application.

Note The real name of the voice browser is used here, *not* the display name. Gateway Adapter real names can be seen by reading the folder name for that adapter in the `gateways` folder of VXML Server.

- **user_management** – This tag encapsulates information concerning the user management database. The `<database>` tag's `jndi_name` attribute contains the JNDI name for the database and the tag itself contains the database to use, which is either *MySQL* or *SQLServer*.
- **endpoints** – This tag encapsulates actions to perform at the endpoints of an application and a call.
  - **on_start_call** – The Java class or URI to call when a call to the application starts. The `class` attribute lists the full class name of the Java class. The `src` attribute contains the URI to call. The two attributes are mutually exclusive. The attribute `run_in_background` is *true* if the Java class or URI is to be accessed in a separate thread by VXML Server; it is *false* if the call is to wait for the action to complete.
  - **on_end_call** – The Java class or URI to call when a call to the application ends. The `class` attribute lists the full class name of the Java class. The `src` attribute contains the URI to call. The two attributes are mutually exclusive.
  - **on_start_app** – The Java class(es) to call when the application is loaded or updated. The `class` attribute lists the full class name of the Java class. The tag can appear multiple times, denoting multiple classes to run at the start of an application. The classes will be run in the order in which they appear in the document. The `error_cancels_deploy` attribute is set to *true* when an error in the class should cause the application loading to fail.
  - **on_end_app** – The Java class(es) to call when the application is taken down (either as a result of the application server shutting down or due to an update). The `class` attribute lists the full class name of the Java class. The tag can appear multiple times, denoting multiple classes to run at the end of an application. The classes will be run in the order in which they appear in the document.
- **root_doc** – This tag contains tags representing the additions made to the application's root document. This tag will not appear if no language, encoding, properties or variables are added to the root document. The possible additions are:

- **vxml_property** – This tag can appear any number of times listing the VoiceXML properties to add to the root document. Typically most voice browsers will take a property set here as an indication that it apply to the entire application. The `name` attribute lists the name of the property and the tag itself encapsulates the value.
- **vxml_variable** – This tag can appear any number of times listing the VoiceXML variables to add to the root document. A variable set in the root document is available to all VoiceXML pages in the application. The `name` attribute lists the name of the variable and the tag itself encapsulates the value.
- **javascript** – This tag encapsulates a Javascript function to place in the application's root document. Any number of these tags can appear to add multiple Javascript functions to the root document.

- **loggers** – This tag contains one or more `<logger_instance>` tags defining the loggers that are to listen to the events for calls for this application. The `name` attribute defines the logger instance name (all logger instances must have unique names). The `class` attribute defines the full Java class name of the logger to use. The optional `configuration` tag points to a configuration file for the logger instance. When the optional attribute `enforce_call_event_order` is *true*, VXML Server will ensure that the logger receives the logger events for a call in the order in which they occurred in the call.

Again, these two documents are sent as HTTP POST arguments to any URL using the XML API. The documents mirror all the functionality provided in the Java API to obtain session information. Changing session information, such as setting session data, is done in the response XML document. Since each component has a separate response document, they are described in each component's individual chapter.