Metadata database and the MediaSense API

MediaSense maintains a database containing a great deal of information about recorded sessions. The database is stored redundantly on the primary and secondary servers. The data includes:

- Various track, participant, call and session identifiers.
- Time stamps and durations.
- Real time session state.
- URIs for streaming and downloading recordings in various formats.
- Server address where recorded files are stored.

Tags

Along with the preceding information, MediaSense stores tags for each session.

Tags are brief, arbitrary, text strings that a client can specify and associate to individual sessions using the Web 2.0 APIs, and optionally, to specific time offsets within those sessions. Timed session tags are useful for identifying points in time when something happened, such as when a caller became irate or an agent gave erroneous information. Un-timed session tags may be used to attach notes which are applicable to the entire session, such as a contact center agent ID or to mark or categorize some sessions with respect to other sessions.

MediaSense also uses the tagging facility to mark when certain actions occurred during the session (such as pause and resume) or when the media inactivity state changes as reported by the SIP signaling. These are known as system-defined tags.

While most tags are associated with an entire session, media inactivity state change tags are associated with a specific track in the session.

MediaSense API

The MediaSense API offers a number of methods to search and retrieve information in the metadata database. Authenticated clients perform simple queries such as finding all sessions that have been deleted by the automatic pruning mechanism or finding all sessions within a particular time range that are tagged with a certain string. The API also supports much more complex queries as well as a sorting and paging scheme by which only a selected subset of the result set will be returned.

The API provides access to a number of other MediaSense functions as well. Use the API to subscribe for events, to manage disk storage, to manipulate recording sessions that are in progress, to remove unneeded inactive sessions and recover their resources, and to invoke operations such as conversion to .mp4 or .wav. Lengthy operations are supported through a remote batch job control facility. The API is described in detail in the Cisco MediaSense Developer Guide.

MediaSense API interactions are conducted entirely over HTTPS and require that clients be authenticated. Depending on the type of request, clients will use either POST or GET methods. Response bodies are always delivered in JSON format. HTTP version 1.1 is used, which allows TCP links to remain connected from request to request. For best performance, clients should be written to do the same.

API requests may be addressed to either the primary or the secondary server (the client needs to authenticate to each server separately), and must provide the HTTP session identifier that was previously obtained from the server being addressed.

Events

The MediaSense event mechanism provides server-based clients with immediate notification when actions of interest to them take place. The following types of events are supported:

- Session events - when recording sessions are started, ended, updated, deleted, or pruned.
- Tag events - when tags are attached to or removed from recorded sessions.
- Storage threshold events - when disk space occupancy rises above or falls below certain preconfigured thresholds.

Session events provide critical information about a session given its current state. A client could then, for example, use the URIs provided in these events to offer real time monitoring and control buttons to an auditor or contact center supervisor. A client might also implement a form of selective recording (as opposed to compliance recording) by deleting (after the fact) sessions that it determines do not need to be recorded.

Tag events are used as a form of inter-client communication: when a session is tagged by one client, all other subscribed clients are informed about it.

Storage threshold events allow a server-based client application to manage disk usage. The client would subscribe to these events and selectively delete older recordings (when necessary) according to its own rules. For example, the client might tag selected sessions for retention and then when a threshold event is received, delete all sessions older than a certain date except those tagged for retention.

Events are populated with JSON formatted payloads and delivered to clients using a Symmetric Web Services protocol (SWS), which is essentially a predefined set of HTTP requests sent from MediaSense to the client (note that HTTPS is not currently supported for eventing).

When a client subscribes for event notifications, it provides a URL to which MediaSense will address its events, as well as a list of event types or categories in which it has an interest. Any number of clients may

subscribe and clients may even subscribe on behalf of other recipients (i.e., the subscribing client may specify a host other than itself as the event recipient). The only restriction is that there cannot be more than one subscription to the same URL.

Events are always generated by either the primary or the secondary server. When both are deployed, each event is generated on one server or the other, but not both (which has implications for high availability).Customers must choose one of two modes of event delivery - one which favors reliability or one which favors convenience.

Metadata differences between CUBE and Unified Communications Manager

At a high level, the metadata which is captured by MediaSense is call controller agnostic. Every recording is made up of one or more sessions, every session has a sessionId, and sessions can have tracks, participants, tags and URI's associated with them and so forth. However, the details of the SIP-level interaction with MediaSense diverge somewhat depending on whether the call is controlled by a CUBE or by a Unified Communications Manager. This leads to some differences in the metadata that MediaSense clients observe and in the real time events they receive.

- The first difference is in the topology: with Unified Communications Manager, calls are forked by one of the endpoints. With CUBE, calls are forked by a network element (ISRG2) through which the call passes. With CUBE, the RTP media and the SIP dialog come from the same device; with Unified Communications Manager, they are different devices. Therefore the way participants are identified and in the way they are associated with media tracks is different.
- More substantial differences in the metadata come from mid-call activities. For example, hold and resume trigger a new session for Unified Communications Manager calls, but they insert track level tags for CUBE calls. Transfer of a forking phone terminates a recording session on Communications Manager, but such an action is not possible with CUBE calls.
- Conference detection varies considerably too. With Unified Communications Manager, this action appears as a transfer to a conference bridge with updated participants and a metadata flag identifying it as a conference. With CUBE, the action also appears as a transfer, but the metadata flag is not supported and the participant data is erratic and unreliable.
- Another difference is with respect to call correlation. Unified Communications Manager and CUBE use different mechanisms for identifying calls.

Simple application clients can be agnostic to call controller type, but more sophisticated clients will usually need to know whether a call was managed by a CUBE or by a Unified Communications Manager.

The Cisco MediaSense Developers Guide contains a full description of the differences between CUBE and Unified Communications Manager deployments.