Loggers

The mechanism used by VXML Server to record information about global administrations, errors caused by sub-systems, activities taken by callers to deployed applications or administrators is by using loggers. Loggers collect this information and can do any number of tasks with it, from aggregating it for reporting purposes, to sending that information to external systems for managing, to simply storing the information in log files. A developer can produce a logger to supplement or replace the functionality the loggers included with VXML Server provide. Due to the complexity of integrating with VXML Server, loggers can be built only by using the Java API.

This chapter describes in detail how to create custom loggers and integrate them with VXML Server. For more detail on the loggers included with VXML Server, refer to the User Guide for Cisco Unified CVP VXML Server and Unified Call Studio.

VXML Server Logging Design

Before discussing the design of an individual logger, it is warranted to introduce the design of the logging mechanism within VXML Server. Knowledge of this design will help the logger developer create loggers that work harmoniously with the system.

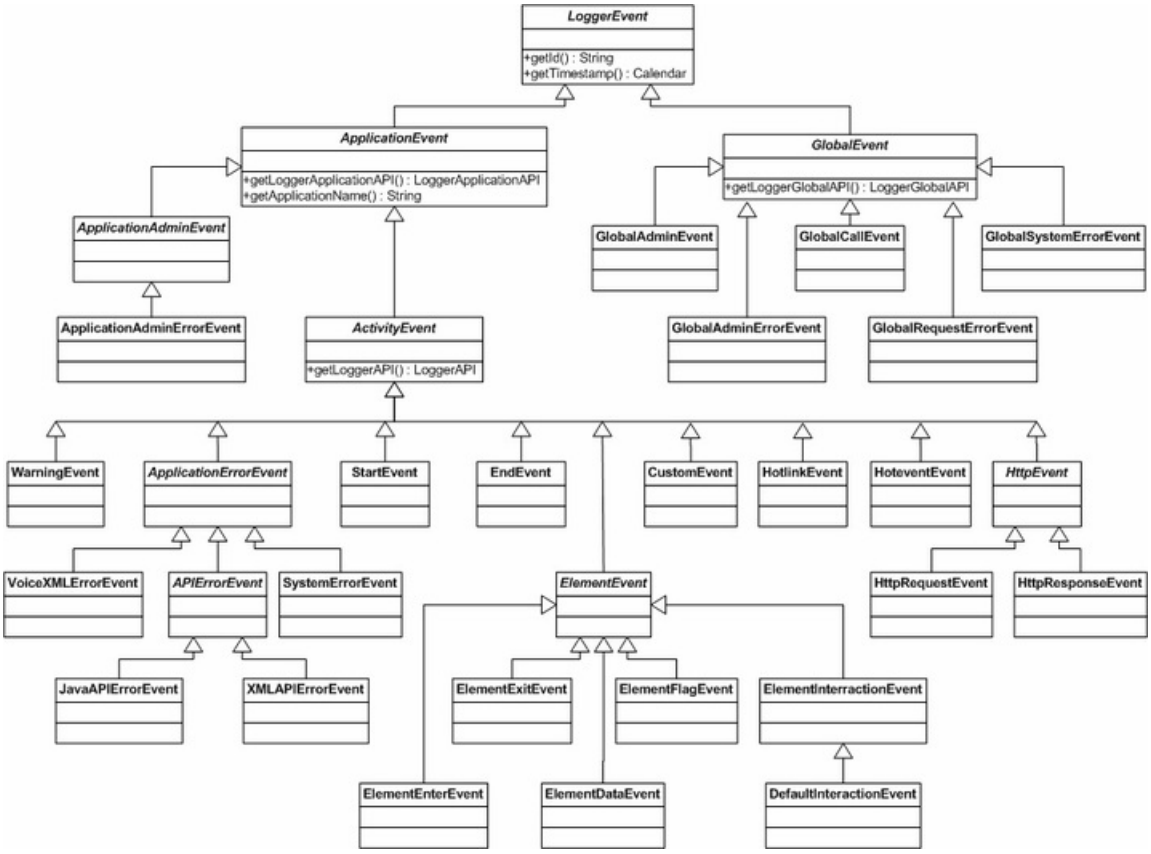
Logger Events

The mechanism by which information is passed to a logger is through an event object. This object will encapsulate information about what just occurred, including a timestamp. Event objects are created by VXML Server in many different situations that belong to three levels: related to global activities, related to an application and related to a call session. The event object will contain all the information accessible to the logger for the particular event as well as information about the environment. For global level events, the environment varies. For some events the environment consists of HTTP information about the request such as parameters and headers. Other events were activated internally and so do not define any environment information. For application-level events such as an administration event, the environment consists of application data and global data (not call data since this event is not affiliated with a call). For call-level events such as a call start event, the environment consists of information about the call such as the ANI, element and

session data, default audio path, etc. Since the purpose of a logger is to report information, loggers are limited to obtaining environment information and cannot change any value. Loggers may still need to store session-related information for its purposes so to accommodate this VXML Server provides loggers *scratch* data that is stored in the session and will be available to the logger only for those events associated with the session.

The following figure shows the class hierarchy for all events both global and application.

Figure 1: Logger Event



Notes on events:

- `GlobalEvent` and `ApplicationEvent` both extend the generic event class `LoggerEvent`. In this structure, new event types in the future can be added without affecting the existing class hierarchy.
- All events have an ID, obtained by calling the `getID` method.

Loggers Use

An application designer can define any number of loggers to use in an application and an administrator can define any number of global loggers to deploy to Unified CVP VXML Server. Logger instances are defined with unique names and loggers can be built to support configurations. Multiple instances of the same logger class can also be defined, with a different configuration for each. Unified CVP VXML Server will then create a separate instance of the logger class for each instance referenced in the application's settings and in the server configuration. Application logger instances are created when the individual applications are initialized

and are maintained for the lifetime of the application. All events for the application, both application-level events as well as call-level events, are handled by this single instance.

Note An instance variable in the logger class will allow it to maintain information that spans calls. Global logger instances are created when VXML Server initializes and are maintained for the lifetime of the system and hence the logger class can maintain information that lasts the lifetime of the system. It will handle any global events that occur.

A logger is expected to *register* the events it wishes to act on. This is done on logger initialization. When VXML Server loads, it initializes all the global loggers and the loggers referenced for an application and records which events each will act on. When a situation occurs that would constitute an event, VXML Server checks to see if any loggers will act on the event and if so, will create the event object and pass it to the logger instances. This registration mechanism allows VXML Server to save the overhead in creating an event if no loggers will act on it. Additionally, should there be multiple loggers acting on an event, only one event object is created and passed to all of them.

In order to ensure that no call be held up due to logging activities, the entire VXML Server logging mechanism is fully multi-threaded. The only logging-related activity that an HTTP request thread (provided by the application server) performs is creating an event object and adding it to a queue. It does not actually handle the logging of that event and once it has added the event to the queue, it continues with the call. A separate, constantly running asynchronous process, called the Logger Manager, will process the events in the queue. This allows the logging process to act independently from the process of handling a call and so will not directly affect the performance of the system.

In order to ensure that no logger be held up due to the activities of another logger (or the same logger) while handling an event, a second layer of threads are used. While the Logger Manager handles the queue, when it is time for a logger to be given the event to handle, this is itself done in a separate thread. The Logger Manager therefore is responsible only for managing the queue of events and spawning threads for loggers to handle them. This ensures that a logger that takes a long time to handle an event does not hold up the logging for the same or other applications, it only holds up the thread in which it is running. A thread pooling mechanism exists to efficiently manage thread use. To avoid creating too many threads when under load, the maximum number of allowable threads in the pool can be configured in the global configuration file named `global_config.xml` found in the conf directory of VXML Server by editing the contents of the `<maximum_thread_pool_size>` tag. For a thread to be reused after it is done with the current task, the `<keep_alive_time>` tag from the same configuration file can be set. When all the allowable threads in the pool are taken, VXML Server will not process the queue until a thread becomes available from the pool.

Note One of the consequences here is that the longer the events remain in the queue, the less *real-time* the logging will occur. Additionally, if the maximum thread pool size is made too low to handle a given call volume, the queue can become very large and could eventually cause issues with memory and spiking CPU. Typically, though, a logger handles an event in a very short period of time, allowing a small number of threads to handle the events created by many times that number of simultaneous callers.

There are times when the true asynchronous nature of the logging design works against the developer. The tasks done by a logger can take a variable amount of time to complete so there is no guarantee when a call event will be handled. This is by design, and for a logger that simply records events that are then sorted by timestamp later, this is not a problem. A logger that requires more context, though, could encounter issues. For example, if a logger needed to note when a call was received so that an event that occurred later on in the

call could be handled correctly, problems could be encountered because there would be no guarantee that the events would be handled in the same order they occurred within the call.

To remedy this situation while keeping the design unfettered, it is possible to specify that VXML Server pass a logger instance events in the same order they occurred in a call. With this option on, the logger developer can be assured that the events for a call would not be handled out of order. In fact, the Activity Logger included with VXML Server has this requirement. The penalty for this requirement, however, is a loss of some of the true asynchronous nature of the system as there will now be situations where events that are ready to be handled must wait for a previous event to be handled by the logger. If a logger hung while handling one event, the queue would forever contain the events that occurred after it in the call, and that call session would not be fully logged. This feature, however, is available to application loggers only, global loggers handle their events as soon as a thread is allocated from the pool to handle it. This is understandable because global log events are more holistic in nature and do not track detailed behavior as application loggers do.

Some of the conclusions that can be deduced from the VXML Server logging design are summarized below:

- A logger developer need not worry about the time taken by the logger to handle an event as it will have no bearing on the performance of the call. With that said, the developer must also be aware of the expected call volume and ensure that the logger not take so long as to use up the event threads faster than they can be handled.
- Loggers work under a multi-threaded environment and the logger developer must understand how to code with this in mind. A single logger class can be handling events for many calls and so it must manage internal and external resources in a synchronized manner to prevent non-deterministic behavior.
- When possible, design an application logger so that it does not rely on events within a call being passed to it in the order in which they occurred in the call. Doing so will maximize performance due to being able to handle events whenever they occur. Should the logger be unable to do so, require that the enforce call event order option be turned on for the logger.

Logger Design

Similar to configurable elements, a logger is constructed by creating a Java class that extends an abstract base class, `GlobalLoggerBase` or `ApplicationLoggerBase`, which in turn extend from the LoggerBase class. The base classes define abstract methods that must be implemented by the logger. Loggers have methods for initialization and destruction as well as an execution method to call when an event is to be handled. These methods may throw an `EventException` to indicate an error in the logger.

An application logger has the additional requirement of implementing a Java marker interface named `LoggerPlugin` to allow Unified CVP Builder for Call Studio to recognize it as a valid logger.

All Java classes related to loggers are found in the `com.audium.server.logger` package while the logger event classes are found in the `com.audium.server.logger.events` package.

Global Logger Methods

The following bullets details the global logger methods:

- `void initialize(File configFile, LoggerGlobalAPI api)`

  This method is called by VXML Server when a new logger instance is created. This occurs in two different situations: the application server starts up or the VXML Server web application is restarted.

The global logger designer has optionally included a reference to a configuration file for the logger which is passed here as a `File` object. If no configuration file reference was specified in the conf directory, this object will be `null`.

The method also receives a `LoggerGlobalAPI` object, which is used to access the GlobalAPI. The `GlobalAPI` provides access to global data (see the User Guide for Cisco Unified CVP VXML Server and Unified Call Studio for details).

Aside from initializing the logger, this method is also responsible for configuring which events the logger instance is to handle. This is done by setting the value of the member variable `eventsToListenFor`. This variable is a `HashSet` that must contain all the event IDs that the logger listens for. VXML Server accesses this `HashSet` to determine if a new event should be sent to the logger. When an event occurs, `eventsToListenFor` is accessed to see if the event ID can be found. If so, the logger will be notified of the event. The IDs for the events are defined in the interface `IEventIDs`. `GlobalLoggerBase` implements this interface so the event IDs are available directly within the logger class.

Note Global loggers can only register for global events (those whose event names begin with GLOBAL).

- `void destroy(LoggerGlobalAPI api)`

  This method is called in two different situations: the application server is shut down, or the VXML Server web application is restarted.

  The method also receives a `LoggerGlobalAPI` object, which is used to access the `GlobalAPI`. The `GlobalAPI` provides access to global data.

  The purpose of this method is to give the logger the opportunity to perform clean up operations. Typically, this would involve closing database connections or files that were opened by the logger in its initialize method or while handling events. A logger that does not have that requirement must still implement the destroy method but can define an empty implementation.

- `void log(GlobalEvent event)`

  This method is the execution method for the logger and is called by VXML Server when a new event has occurred that the logger must handle. This method is called only if VXML Server has found the event's ID in the `eventsToListenFor HashSet`.

  Only one object is passed to this method, a `GlobalEvent`. This object encapsulates all the information about the event and provides access to other environment information depending on the type of event. `GlobalEvent` is a base class for all global level events and the logger will typically check for the event type and cast to the appropriate event to get its information. All event classes are found in the `com.audium.server.logger.events` package.

- `void doPreLogActivity(GlobalEvent event)`
  `void doPostLogActivity(GlobalEvent event)`

  These two methods can be optionally overridden to perform any activity desired before and after the `log` method is called. By default these methods are defined in the `GlobalLoggerBase` class to do nothing.

Application Logger Methods

The following are the application logger methods:

`void initialize(File configFile, LoggerApplicationAPI api)`

This method is called by VXML Server when a new logger instance is created. This occurs in four different situations: the application server starts up, VXML Server web application is restarted, the application the logger instance belongs to is deployed after the application server had started up, and the application is updated. These situations are the same as those for when the application start class is called (see Application Start Classes for more on these situations).

The application designer has optionally included a reference to a configuration file for the logger which is passed here as a `File` object. If no configuration file reference was specified in the application settings, this object will be `null`.

The method also receives a `LoggerApplicationAPI` object, which is used to access the `GlobalAPI`. The `GlobalAPI` provides access to application data and global data (see the User Guide for Cisco Unified CVP VXML Server and Unified Call Studio for details).

Aside from initializing the logger, this method is also responsible for configuring which events the logger instance is to handle. This is done by setting the value of the member variable `eventsToListenFor`. This variable is a `HashSet` that must contain all the event IDs that the logger listens for. VXML Server accesses this `HashSet` to determine if a new event should be sent to the logger. When an event occurs, `eventsToListenFor` is accessed to see if the event ID can be found. If so, the logger will be notified of the event. The IDs for the events are defined in the interface `IEventIDs`. `ApplicationLoggerBase` implements this interface so the event IDs are available directly within the logger class.

`void destroy(LoggerApplicationAPI api)`

This method is called by VXML Server when an application is released. This occurs in four different situations: the application server is shut down, VXML Server web application is restarted, the application the logger instance belongs to is released, and the application is updated. These situations are the same as those for when the application end class is called (see Application End Classes for more on these situations).

The method also receives a `LoggerApplicationAPI` object, which is used to access the GlobalAPI. The GlobalAPI provides access to application data and global data.

The purpose of this method is to give the logger the opportunity to perform clean up operations. Typically, this would involve closing database connections or files that were opened by the logger in its initialize method or while handling calls. A logger that does not have that requirement must still implement the destroy method but can define an empty implementation.

`void log(ApplicationEvent event)`

This method is the execution method for the logger and is called by VXML Server when a new event has occurred that the logger must handle. This method is called only if VXML Server has found the event's ID in the `eventsToListenFor HashSet`.

Only one object is passed to this method, an `ApplicationEvent`. This object encapsulates all the information about the event and provides access to other environment information depending on the type of event. `ApplicationEvent` is a base class for all application-level events and the logger will typically check for the event type and cast to the appropriate event to get its information. All event classes are found in the `com.audium.server.logger.events` package.

The following notes refer to the figure in Logger Events, on page 1:

- All application events have access to the Global API to get application and global data.
- Application events are defined at two levels: call-level, and application-level.

  Call-level events extend from `ActivityEvent` and are associated with a particular call. As such, these events provide information about the call environment through the method `getLoggerAPI()`. The resulting `LoggerAPI` object is from the Session API (see Session API for more) and provides read-only access to session information such as element and session data.

  Application-level events are associated only with an application and not a call session. Currently, the only application-level events are the application administration event (`ApplicationAdminEvent`) that reports on administration activity performed on the application and the application administration error event (`ApplicationAdminErrorEvent`) that reports any errors encountered while performing administration activities.
- The events extending `ActivityEvent` are all the events that can occur in a call: a new call, a call ending, an element being entered, an element exiting, an element storing data, a flag element being triggered, an element interacting with the caller, a hotevent being activated, a hotlink being activated, a custom event caused by calling the `addToLog` method of the Session API or entering data in an element's configuration pane in Builder for Call Studio, a warning event, an error event, and an HTTP event signifying a new HTTP request made to VXML Server.
- Error events are defined in a hierarchy to define different types of errors. This allows a logger developer to act on certain types of errors or to do different tasks based on the error type.
- Currently, when a voice element returns interaction data, VXML Server sends loggers a `DefaultInteractionEvent`, which extends from `ElementInteractionEvent`. This design will allow for different interaction content in the future without affecting the existing class hierarchy.

`void doPreLogActivity(ApplicationEvent event)` **and** **`void doPostLogActivity(ApplicationEvent event)`**

These two methods can be optionally overridden to perform any activity desired before and after the `log` method is called. By default these methods are defined in the ApplicationLoggerBase class to do nothing.

Utility Methods

These utility methods provide important information to the logger.

Common Utility Methods

These methods are defined in the `LoggerBase` class and are therefore applicable to both application and global loggers.

- `int getLoggerType()`

  This method returns the type of the logger, and is for future use. Currently, it returns `APPLICATION_LOGGER` or `GLOBAL_LOGGER`.
- `String getLoggerInstanceName()`

  This method returns the name of the logger instance as defined in either VXML Server's configuration or the settings of the application.

- `HashSet getEventsListeningFor()`

  This method returns the `HashSet` containing the event IDs that the logger is handling. This method is called by VXML Server to determine if to send a new event to the logger. This methods simply returns the protected member variable `eventsToListenFor` defined in `LoggerBase`.

- `String getLogFileDirectory()`

  This method returns the full path of the directory VXML Server has provided this logger to store its logs, if necessary. On startup, VXML Server creates a folder for each global logger instance within the global `logs` folder. It also creates a folder for each application logger instance within the `logs` folder of the particular application. The folders are given the logger instance name. The folder is for exclusive use of the logger instance should it require it. Should the logger create log files, it should use this directory unless the logger's is designed to log somewhere else. Should the logger not use files, such as if it logs directly to a database, this method and the folder it references can be ignored.

Application Logger Utility Methods

The following utility methods apply only to application loggers.

- `String getApplicationName()`

  This method returns the name of the application that the logger instance belongs to.

- `String enforceCallEventOrder()`

  This method returns `true` if the logger instance has been configured to enforce the call event order, `false` otherwise. This is useful if the logger wishes to throw an error if the logger instance was not configured a certain way.