Application Management API

VXML Server comes with an application management facility to provide visibility into the runtime environment and provides control over the platform together with its components and services. This comprehensive feature set allows an administrator to operate, administrate and manage (OA&M) the health of the platform as well as obtain statistics and performance measurements.

This chapter describes in detail how to the management server system has been designed and how to create custom management support based on the API provided with the VXML Server platform.

Design

All the OA&M features on VXML Server are built with JMX management standards. Applications and components on VXML Server are instrumented using Managed Beans (MBeans). MBeans expose their management interfaces, composed of attributes, operations and event notifications, through a JMX agent for remote management and monitoring.

Managed resources are categorized at levels of application, global configuration and command, and platform. Each level may facilitate three typical JMX managements: lookup and modify configuration, collect and avail application statistics, notify of state changes and erroneous conditions.

At the application level, administrators can operate the following:

- Get/set default audio path
- Get/set suspended audio
- Get/set session timeout
- Get gateway adapter
- Application release
- Application resume
- Runtime status
- Application update

- Suspend/resume application
- Retrieve application administration history
- Get application data
- Set and remove application data
- Remove all application data
- Get application data names

The global level manages VXML Server as a web application. It allows:

- Read logger event queue size
- Read/write maximum logger thread pool size
- Read/write minimum logger thread pool size
- Read/write logger thread keep alive time
- Read/write session invalidation delay
- Get global status
- Deploy all new applications. This command assumes all the deployable applications have been updated to the AudiumHome. All this command does is to deploy these applications.
- List all new applications. This command lists the names of all new applications (that is, those which have yet to be deployed).
- Deploy new application. This command deploys the specified application. To retrieve a list of new application names, use *List all new applications* (above) first.
- Flush all old applications
- Update all applications
- Update common classes
- Suspend/resume VXML Server
- Retrieve global administration history
- Get/set global data
- Get all global data names
- Remove all global data

At the same level, the management system also provides information regarding metrics collection:

- The total number of calls since VXML Server starts up.
- Maximum and average number of concurrent calls, the timestamp when it reaches the maximum calls.
- Maximum and average response time.
- Number of calls that time out.
- Number of calls that encounter errors.

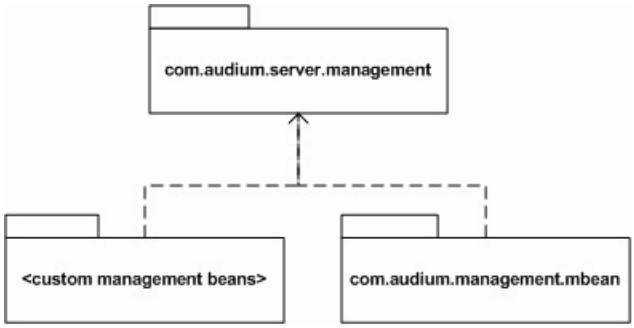
- Transfer/zero-out rate. The number of calls transferring to an operator or live agent.
- Abandon rate. The number of calls that end as hang up.
- Call completion rate. The number of calls that are completed as expected through the callflow.
- Maximum logger event queue size.
- Maximum loggers thread count.

The platform level consists of information for VXML Server as a product:

- Product Name
- Product Version
- Installation Key
- Licensed Ports
- License Expiration Date
- Licensed GW Adapters

VXML Server MBeans implement a Java API set that is defined for management interfaces. Developers can create custom management beans that use the API. The following figure depicts the relationship among the management API packages: `com.audium.server.management` and the built-in beans (which use the package `com.audium.server.management.mbean`) and custom beans.

Figure 1: Management Relationship Example: MBeans



When VXML Server starts, all the beans that are deployed to `management` directory would be loaded and registered to the JMX server. Depending on the base class it extends, a bean is grouped in VoiceApplication, Global or Info under the domain *Cisco VXML Server Application Management API*. For an application-scoped bean that extends `AbstractApplicationCommand`, `AbstractApplicationConfig` or `AbstractApplicationData`, a bean instance will be created for each application. For instance, if bean A extends `AbstractApplicationCommand` and there are currently two applications deployed: appA and appB, then two instances of bean A class will be created, one for appA, the other for appB.

For a logger that has been deployed with an application, a management bean will be dynamically generated and registered for it. Currently there is no extra requirement for a logger to become manageable. However, the following operations are excluded from logger beans due to their irrelevance with respect to manageability: `log()`, `init()`, `equals()`, `destroy()`, `initialize()`, `hasCode()`, `getClass()`, `wait()`, `notify()`, `notifyAll()`, and `toString()`.

Custom beans that directly implement `AudiumManagementBeanInterface` but do not extend any of the following abstract classes will be loaded but not registered when VXML Server starts up: `AbstractApplicationCommand`, `AbstractApplicationConfig`, `AbstractApplicationData`, `AbstractGlobalCommand`, `AbstractGlobalConfig`, `AbstractGlobalData`, and `AbstractCallServicesInfo`.

This type of bean should register itself to the management server and will not have access to the information provided by VXML Server. VXML Server only loads these beans.

Note The standard VXML Server logging mechanism does not pick up errors or exceptions that happen in a custom bean and the developer is responsible for handling the errors themselves or let them propagate to the JMX console.

Management Bean Samples

This section describes how typical bean can be created based on the VXML Server management system API.

1. Create the bean interface where it defines the attributes and operations it will manage. For example:

```
public interface ApplicationConfigMBean {
    public String getDefaultAudioPath();
    public void setDefaultAudioPath(String path);
    public String getSuspendedAudioFile();
    public void setSuspendedAudioFile(String file);
 }
```

2. Create the bean class that implements the interface and extend the predefined abstract class.

```
public class ApplicationConfig extends AbstractApplicationConfig implements
ApplicationConfigMBean{
    public String getDefaultAudioPath() {
           return super.getDefaultAudioPath();
 }
    public void setDefaultAudioPath(String path) {
           super.setDefaultAudioPath(path);
 }
    public String getSuspendedAudioFile() {
           return super.getSuspendedAudioFile();
 }
    public void setSuspendedAudioFile(String file) {
           super.setSuspendedAudioFile(file);
 }
}
```

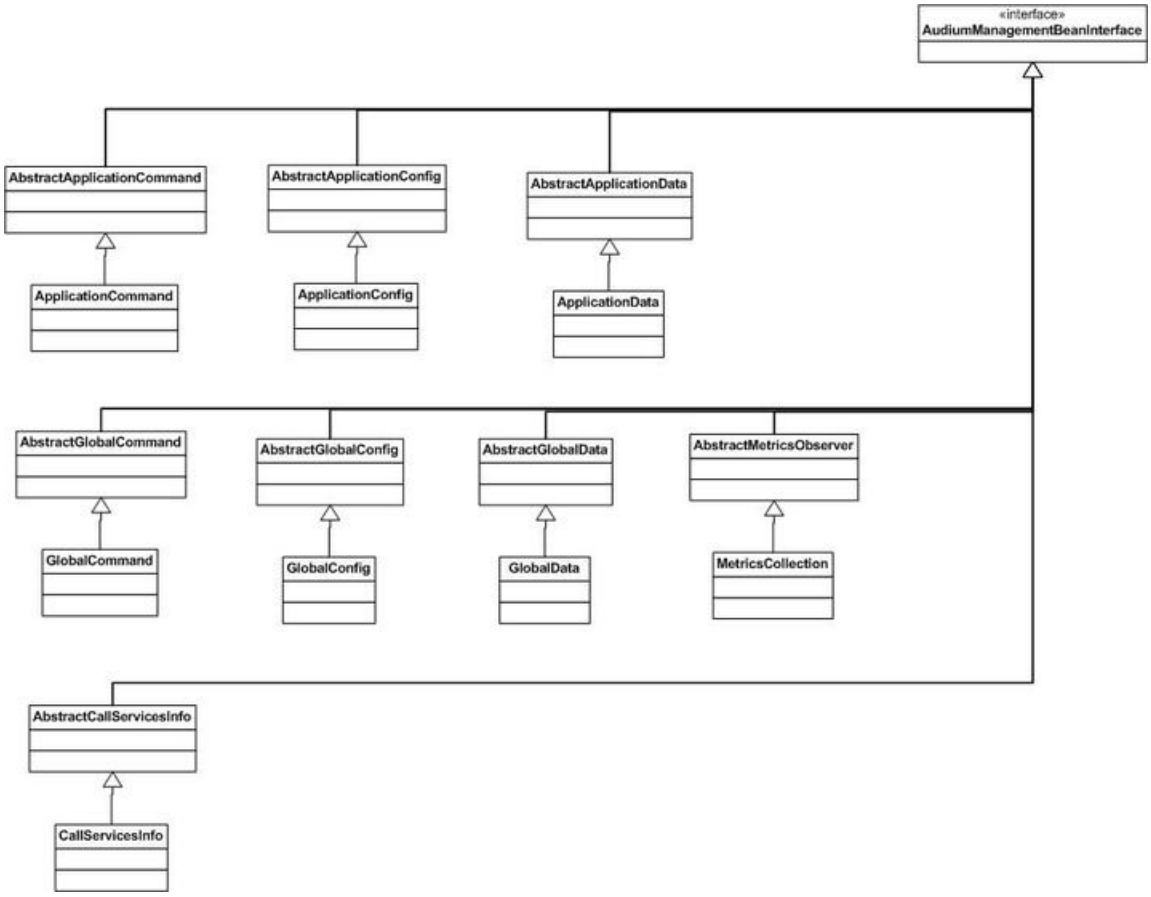
3. By extending the predefined abstract class, a custom bean gets access to the information provided by VXML Server. For example, a bean class that extends `AbstractGlobalData` can call `getAllDataNames()`, `removeAllData()`, `getData()`, and so on. These methods are made accessible when the bean gets loaded.

Application Management Interfaces

When writing custom management beans, the public APIs are used to gain access to the runtime information that will be provided by VXML Server. We now discuss each of these interfaces and abstract classes in details. The following figure displays the static class structure which corresponds to the three levels of the

aforementioned information structure. The built-in management beans are also included in the diagram to show the relationship between the APIs and the beans.

Figure 2: Application Management Static Class Structure



The following list describes the elements in the figure:

- **AudiumManagementBeanInterface** – This is a marker interface for any JMX MBeans to load at the server startup. All the management beans must implement this interface or its subinterfaces.
- **AbstractApplicationCommand** – This abstract class encapsulates all the application commands. The JMX beans that extend this class will have access to all these operations.
- **AbstractApplicationConfig** – This class can be used to set and get application configuration attributes.
- **AbstractApplicationData** – MBeans that extend this class will have access to application data and can manipulate this data by adding or removing them. Application data is typically used to store application-specific information that is available to all calls.
- **AbstractGlobalCommand** – This abstract class encapsulates all the global commands. The JMX beans that extend this class will have access to all these operations.
- **AbstractGlobalConfig** – This class can be used to set and get global configuration attributes.

- **AbstractGlobalData** – Subclasses of this class will have access to global data and can manipulate these data by adding or removing them. Global data is typically used to store static information that needs to be available to all components, no matter which application they reside in.
- **AbstractMetricsCollection** – This class can be used to access all the metrics information at the global level.
- **AbstractMetricsObserver** – This is a marker class. When VXML Server starts up, subclasses of this class will be added as listeners to application activity events, logging management events and global events. Although a subclass will be notified when any of these events occurs, it only needs to react to the interested events. Events are supposed to be handled by overwriting the `update()` method.