Database

The `database` element provides the ability to run an SQL command on external databases within a voice application call flow. The element requires JNDI to be configured in the Java application server to handle database connections. Only a single SQL statement can be run per element. There are four types of commands that can be made:

- **Single** – This is used to run a SQL query that returns only a single row. Element data will be created with the variable names being the names of the columns returned and the value of that column as the element data value (as a string). If no row is returned, no element data will be set.
- **Multiple** – This is used to run a SQL query that returns multiple rows. A Unified CVP-defined Java data structure, the Java class `ResultSetList`, stores the full result and is placed in session data. If no rows are returned, the `ResultSetList` object in session data will be empty. For detail about the ResultSetList data structure, refer to the javadocs for this class.
- **Inserts** – This is used to run a SQL INSERT command that inserts information into the database.
- **Updates** – This is used to run a SQL UPDATE command that updates information in the database.

The developer can utilize substitution to create dynamic queries. The Database element is ideal for performing simple queries and updates. It may not be sufficient for performing complex database interactions such as multiple dependent queries or stored procedure calls. One would use a custom configurable or generic action element for these tasks. To avoid performance issues while creating database connections, you must implement database pooling on the application server.

Settings

Name (Label)	Type	Req'd	Single Setting Value	Substitution Allowed	Default	Notes
type (Type)	string enum	Yes	`true`	`true`	`single`	The type of query: `single`, `multiple`, `insert` or `update`. **Note** The "xml_resultset" element data is not created when `insert` or `update` is selected.
jndiName (JNDI Name)	string	Yes	`true`	`true`	*None*	JNDI name for the SQL datasource of the database.
key (Session Data Key)	string	Yes	`true`	`true`	*None*	For queries of type multiple, the name of the session variable for which the results of the query will be stored.
query (SQL Query)	string	Yes	`true`	`true`	*None*	The SQL query to run.
enableXmlResultSet (Result-Set XML)	Boolean	Yes	`true`	`false`	`true`	If the Result-Set XML option is set to False, the "xml_resultset" element data is not created when the XML Data conversion functionality is disabled.

Element Data

In the substitution tag, the two element data `num_rows_processed` and `xml_resultset` are available by default when a database element is selected. The {Data.Element.DBElement1.num_rows_processed} and {Data.Element.Database_01.xml_resultset} are the two tags that can be added for these element data respectively. The Database element `num_rows_processed` carries the number of rows fetched when a query is selected from the database, and the number of rows updated when any update, delete or insert operation is made in the database. The `xml_resultset` carries the database result in the XML form for a single query or multiple select query. The `num_rows_processed` can be used for any data *type* settings. The `xml_resultset` can only be used for Insert and Update *type* settings. However, when the *type* setting is set to *single* for an Element data, the names of the return columns are created containing the respective return values.

For example, if a query returned the following information:

```
foo bar
```

```
123 456
```

The following element data will be created: *foo* with the value *123* and *bar* with the value *456*.

Session Data

Session data is created *only* when the `type` setting is set to *multiple*. In all other cases, no session data is created.

Name	Type	Notes
[value of setting “key”]	ResultSetList	The Java data structure that stores the returned values from a multiple type query. The name of the session data variable is specified by the developer in the `key` setting.

Exit States

Name	Notes
done	The database query was successfully completed.

Folder and Class Information

Studio Element Folder Name	Class Name
Integration	com.audium.server.action.database.DatabaseAction

Events

Name (Label)	Notes
Event Type	You can select Java Exception as event handler type.

The output of the Customer_Lookup element can be in JSON format . To know more about parsing the JSON Data refer to "Parsing JSON Data" section in *User Guide for Cisco Unified CVP VXML Server and Cisco Unified Call Studio.*

Create JNDI Database Connection in Tomcat for Use in VXML Applications

Summary

Steps

This section explains how to create a new JNDI database connection in Tomcat. These instructions are useful when you want to use the built-in Studio Database element, or create some custom code that accesses database functionality through JNDI.

1. To enable database access on your application server, a compatible JDBC driver must be installed. These drivers, typically packaged as JAR files, should be placed in a directory accessible to the application server classpath (on Tomcat, for example, place in *%CVP_HOME%\VXMLServer\Tomcat\lib)*.

   Note The database must exist for this connection to work. CVP VXML Server will not create the database for you.

2. Add a Tomcat Context for the database connection so that the CVP VXML Server knows how to communicate with your database. For more information, see https://tomcat.apache.org/tomcat-9.0-doc/jndi-datasource-examples-howto.html
3. In Audium Builder for CVP Studio, edit the configuration of the Database element in question. Enter the string you entered *below* in <LABEL_YOU_CHOOSE> from the Tomcat Context into the JNDI Name property of the Settings tab of your Database element.

   Note Do not include the *jdbc/* portion here.

   Here is an example that uses MySQL (edit *context.xml* from *AUDIUM_HOME\Tomcat\conf* folder):

   -
   ```
   <Context>
   <Resource name="jdbc/<LABEL_YOU_CHOOSE>"
   auth="Container"
   type="javax.sql.DataSource"
   username="USER_NAME"
   password="USER_PW"
   driverClassName="com.mysql.jdbc.Driver"
   url="jdbc:mysql://HOSTNAME_OR_IP:PORT/DB_NAME" />
   </Context>
   ```

The default port number for MySQL is 3306. An example url for the above Context would be *jdbc:mysql://localhost:3306/DB_name*

Note Alternately, the `<Resource>` can be configured in the *server.xml* file under `<GlobalNamingResources>`, and a `<ResourceLink>` created in *context.xml* under `<Context>`

4. Under heavy load conditions, enable Database Connection Pooling.

A database connection pool creates and manages a pool of connections to a database. Recycling and reusing already existing connections to a database is more efficient than opening a new connection. For further information on Tomcat Database Pooling, see https://tomcat.apache.org/tomcat-9.0-doc/jndi-datasource-examples-howto.html.

Note

Tomcat 8.0 has two connection pool libraries: commons-dbcp and tomcat-jdbc-pool. Due to a known issue with tomcat-jdbc-pool connection pool library, if the connection between the CVP VXML server and the remote SQL server goes down, the connections are not re-established automatically. The connections can be re-established only after the VXMLServer tomcat service is restarted.

The commons-dbcp connection pool library does not have this problem. The commons-dbcp library is used by default, and the tomcat-jdbc-pool is only used if the tomcat context.xml file contains the following line:

```
factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
```

Due to this issue, Cisco does not recommend using the tomcat-jdbc-pool library.