

# Accessing the Database from a Plugin

---

You can add tables needed by your plugin to the application database. You do this by including a schema.xml file that describes the schema of your additions and by defining a Spring bean that represents access to the data from your Java code.

A schema.xml file captures the database schema you're adding. That XML is in the shape needed by SQLGen, a tool inside the application that reads your XML and converts it to the SQL commands needed to create the tables.

The following example from the feedblog sample defines a table with seven columns, of which the feedblogID column is the primary key:

```
<schema name="feedblog">
  <table name="jiveFeedBlog" description="table for feed blogs">
    <column name="feedblogID" type="bigint" nullable="false" description="Primary key"/>
    <column name="blogID" type="bigint" nullable="false" description="ID of the blog"/>
    <column name="authorID" type="bigint" nullable="false" description="ID of the author" />
    <column name="feedURL" type="varchar" size="255" nullable="false" description="URL of the feed" />
    <column name="includeTags" type="varchar" size="50" nullable="true" description="" />
    <column name="excludeTags" type="varchar" size="50" nullable="true" description="" />
    <column name="includePostedByLink" type="int" nullable="false" description="" />
    <index type="primary" name="jiveFeedBlog_pk" column="feedblogID" />
  </table>
</schema>
```

The table will be created when the plugin is deployed. In the plugin.xml file you define the version of your schema (which can be changed when providing upgrades for plugins) as well as the schema name to look for in the schema.xml. The following example tells the application to assign version 1 to the feedblog table schema (in the jiveVersions table) and to look for a schema named "feedblog" in schema.xml.

```
<!-- Database sequence and schema version information -->
<databaseKey>feedblog</databaseKey>
<databaseVersion>1</databaseVersion>
```

Note: As of version 2.5 the plugin framework does not support automatic schema upgrades from a plugin. In order to change your database schema you will either need to manually modify the database tables or drop the tables, remove the plugin entry from `jiveVersion` and re-install the plugin.

Your code can access the database through its own data access object (DAO) Spring bean. You write a class to handle database requests and configure your plugin to recognize this class as a Spring bean. (To qualify as a Spring bean, a class need only have a default constructor or, if that's overridden, specify in configuration which constructor to use.) Look below for a Spring configuration example.

The following abbreviated DAO bean example executes an SQL insert to add new data to the `jiveFeedBlog` table defined in the `schema.xml` file above.

```
public class FeedBlogDAOImpl extends JiveJdbcDaoSupport implements FeedBlogDAO {
    public void addFeed(FeedBlog feedblog) {
        FeedBlogImpl impl = getImpl(feedblog);
        String sql = "INSERT INTO jiveFeedBlog (feedblogID, blogID, authorID, feedURL,
            includeTags, excludeTags, includePostedByLink) "
            + "VALUES (?, ?, ?, ?, ?, ?)";
        long id = nextID(impl);
        getSimpleJdbcTemplate().update(sql, id, impl.getBlogID(), impl.getAuthorID(),
            impl.getFeedURL(),
            impl.getIncludeTags(), impl.getExcludeTags(), impl.isIncludePostedByLink());
        impl.setID(id);
    }
}
```

You make the application aware of your Spring bean by including it in a `spring.xml` configuration file that you include in your plugin. Beans you specify in this way are added to the set of Spring beans already available to application code as well as to your plugin's.

The following snippet from a `spring.xml` file creates references to two beans: the DAO bean described above and a manager bean that has a higher-level role:

```
<beans>
  <!-- Defines the DAO class as a Spring bean. -->
  <bean id="feedBlogDAO"
    class="com.jivesoftware.clearspace.plugin.feedblog.impl.FeedBlogDAOImpl">
```

```
<!--  
    Includes a reference to an object used by a class higher  
    up the inheritance chain.  
-->  
<property name="dataSource">  
    <util:property-path path="dataSourceFactory.dataSource" />  
</property>  
</bean>  
  
<!-- Defines the a manager class as a Spring bean. -->  
<bean id="feedBlogManager"  
    class="com.jivesoftware.clearspace.plugin.feedblog.impl.FeedBlogManagerImpl"  
    parent="jiveManager">  
    <!--  
        Properties specify Spring-managed object references that can be injected  
        (assigned to variables via setter methods) into the manager at run time.  
        The feedBlogDAO property here refers to the FeedBlogDAOImpl bean defined above.  
    -->  
    <property name="feedBlogDAO" ref="feedBlogDAO" />  
    <property name="blogManager" ref="blogManagerImpl" />  
</bean>  
  
<!-- Other beans omitted for brevity. -->  
</beans>
```

When you've used the spring.xml file to add your DAO bean to the Spring context, you can take advantage of Spring's dependency injection feature. With dependency injection, you add a class-level variable and public setter method, then allow Spring to "inject" an object reference. This replaces code that would create an instance of the class.

In the following example, the FeedBlogDAOImpl set by Spring corresponds to the feedBlogDAO property defined in configuration for this manager bean. The code uses the instance to call its addFeed method (the code in the bean above).

```
// Imports omitted for brevity.  
  
public class FeedBlogManagerImpl implements FeedBlogManager {  
  
    // Variable to hold injected object reference.  
    FeedBlogDAOImpl dao;  
  
    // An accessor through which Spring can inject a reference  
    // to the FeedBlogDAOImpl class that handles database  
    // requests.  
    public void setFeedBlogDAO(FeedBlogDAOImpl feedblogDAO) {  
        this.dao = feedblogDAO;  
    }  
}
```

## Accessing the Database from a Plugin

```
}  
  
// Code omitted for brevity.  
  
// A method that delegates to a DAO method.  
  
public void addFeed(FeedBlog fb) {  
    dao.addFeed(fb);  
}  
  
// Code omitted for brevity.  
}
```