



@task

Java EJB Example

March 27, 2007

- The following document discusses the layout and reasoning of the Java EJB integration example.
- It includes some helpful hints and suggestions for users desiring to utilize the @task EJB API.

@TASK JAVA EJB EXAMPLE

PROCESS

1. **Outline a Proposed Solution.** To use the SDK you should have a concrete idea of what you would like to accomplish with your integration utility. Some familiarity with @task is recommended, because you will spend less time acquainting yourself with the desired functionality. If you haven't yet read the API Overview document, you should. Once a plan has been developed for your project, you can access the EJB and the API documentation, which is included in the SDK, to get the desired information on the necessary calls to the API. The list of available calls can be accessed in the `com.attask.api.API` class found in the Javadocs in the `client/docs/api`.

In this example we will login, get a session ID, find a group, create a project within that group, attach a task and an issue to the project, create a category for the issue, edit the task and issue, and cleanup. Although these are simple tasks, they demonstrate how the most frequently used calls in the API can be implemented.

2. **Updating the Source Code.** In order to run this sample code on your machine you will need to access the `EJBExample.java` file contained in the `sdk_4_0\client\java-ejb\src` directory, and update the following line of code to refer to the IP address or hostname of @task:

```
props.setProperty("java.naming.provider.url", "jnp://localhost:1099");
```

3. **Setting Up Variables.** In most cases you will have a similar set up of variables as seen below.

```
// Constants
private static final Logger LOG = LogManager.getLogger(EJBExample.class);

// Fields
/** AtTask uses its own session IDs */
protected static String _sessionID = null;

/** Provides access to the information AtTask stores in its sessions */
protected static SessionAttributesBean _sessionAttributes;

/** Implements the EJB calls using the Apache Axis libraries */
protected static API _api;

/** AtTask uses a special int value to represent "null" */
protected static int _intNull;

// Constructors
private EJBExample() {
```

The Logger is set up because of a dependency on java libraries in the SDK. If you are using the Java and/or the Axis libraries you will need to recreate a similar object. In the Fields section we create a container to store the current sessionID. All API calls are done in the context of a session so that @task can keep track of which user is doing each action. Also your code will function faster if it does not have to perform a login every time it accesses the server. The rest of the objects seen above are included for the reason stated.

4. **Connecting to and Logging Into the Server.** @task requires credentials to be submitted for any access of the server and the @task database. The example demonstrates how to locate the server, connect to it and perform a login into the @task database. Please note that the `APISupport` object is a helper file used to connect to an EJB server. This is set to the variable `_api` which implements the EJB calls.

```
// Connect to server.
Properties props = System.getProperties();
props.setProperty("java.naming.provider.url", "jnp://localhost:1099");
props.setProperty("java.naming.provider.port", "8080");
props.setProperty("java.naming.factory.initial",
    "org.jnp.interfaces.NamingContextFactory");
props.setProperty("java.naming.factory.url.pkgs",
    "org.jboss.naming.client");
props.setProperty("jnp.socketFactory",
    "org.jnp.interfaces.TimedSocketFactory");
props.setProperty("jnp.timeout", "0");
```

```

props.setProperty("jnp.sotimeout", "0");
System.getProperties(props);
APISupport.init();
_api = APISupport.getAPI();

```

In the present example, the username and password are set by calling the `getProperty` method on the `props` object. This allows the environment to pass in a different set of credentials than those listed below. The `sessionID` is stored as indicated before, and the `sessionAttributes` are also captured.

```

// Login.
String username = props.getProperty("test.username", "admin");
String password = props.getProperty("test.password", "user");
LOG.info("Logging in to web service");
_sessionID = _api.login(username, password);
_sessionAttributes = _api.getSessionAttributes(_sessionID);
_intNull = _sessionAttributes.getIntNull();

```

5. **Creating a Projects.** As described above, this example demonstrates how to properly create projects, tasks, and issues. Each of these data objects is housed inside a Java Bean, which is just a structure containing information about the object. Thus, whenever a bean is created it has a set of fields, i.e., a task has a name, description, planned start date, etc., some of which require values. For example, a project requires a name, `CategoryID`, `ScheduleID`, and `TemplateID`. Please note that `NULL` values and empty values are acceptable for many of the fields. The following code represents how creating a project is accomplished. First, you must find a group under which the project will be created. In this case we will search the database for a suitable group. If you do not know of a specific group you can always create the project in the default group.

```

// Find a group for the new project.
LOG.info("Searching for groups");
GroupBean[] groups = _api.getGroups(_sessionID);
GroupBean group = groups[0];
// There is a default group that includes everybody.
groupID = group.getID();

```

Once the group has been set you can now give the project a name and fill in values for the required fields

```

// Create a project.
ProjectBean project = new ProjectBean();
project.setName("Project Name");
project.setGroupID(groupID);

// All of these fields must be set to some value, but NULL or empty
// values are OK.
project.setCategoryID(_intNull);
project.setScheduleID(_intNull);
project.setTemplateID(_intNull);
// The planned start date is required.
project.setPlannedStartDate(new GregorianCalendar());

```

Additionally, there are many fields in `@task` that require a `String` value, these include fields that are drop-downs. In this example the value of `status` is set directly by passing in "CUR", but you may also get a `String` value by asking the `Enum` for it.

```

// This field is also required and must use one of the values listed
// in the provided Javadoc documentation.
project.setStatus("CUR");
project.setCompletionType("MAN");
project.setUpdateType("AUTO");

```

Now that all the values have been set you can create the project. This is functionally the same as clicking **Submit** in the application.

```

// Create the project and store its ID.
LOG.info("Creating a project");
projectID = _api.addProject(_sessionID, project);

```

- 6. Adding a Task to a Project.** Now that we have a functioning project we want to add a task to it. This process has many of the same steps as creating a project. In this case we create a TaskBean. Tasks require a value for duration. As described below, the duration is stored in 8 hour days.

```
// Add a task to the project.
TaskBean task1 = new TaskBean();
task1.setName("Task Name");
task1.setProjectID(projectID);

// By default, the duration is stored in 8 hour days.
task1.setDuration(3.0);
task1.setPercentComplete(0.0);

// This field is also required and must use one of the values listed in
// the provided Javadoc documentation.
task1.setPriority(1);
```

The following values are required fields and in most cases will be set to some values, but for our example NULL values are acceptable. After these values have been given, we create the task in a similar fashion to a project.

```
task1.setRoleID(_intNull);
task1.setCategoryID(_intNull);
task1.setMilestoneID(_intNull);
task1.setAssignedToID(_intNull);
task1.setParentID(_intNull);

// Create the task and store its ID.
LOG.info("Creating a task");
taskIDs = _api.addTasks(_sessionID, new TaskBean[] { task1 });
```

- 7. Editing Tasks and Issues.** Editing tasks and issues is a fairly straightforward process. As you can see from the example code below, it requires retrieving the desired task via the taskID and modifying a value(s) as desired. Please note that @task does not support single field editing. You must first retrieve the bean by calling a “get” method, which will preserve all of the saved and unedited fields, and edit the desired fields and then call “edit”. If you do not do this you run the risk of overwriting information with NULL or empty values. After the task has been edited we confirmed that the change had been modified, but you do not need to replicate this practice. An example of this process is found below.

```
// Edit the task.
TaskBean task2 = _api.getTaskByTaskID(_sessionID, taskIDs[0]);
task2.setName("New Task Name");
LOG.info("Editing a task");
_api.editTask(_sessionID, task2);

TaskBean task3 = _api.getTaskByTaskID(_sessionID, taskIDs[0]);
if (!"New Task Name".equals(task3.getName())) {
    LOG.error("Task not modified");
}
```

- 8. Creating a Category.** In @task you have the option to append custom data to your projects, tasks and issues. In general a category is made up of parameters. For example, a category could represent a box of fruit, and a parameter could be the different types of fruit. In @task this parameter could be represented by a dropdown field that is populated with types of fruit that are found in the category. For a more in depth explanation of categories and parameters please refer to the user guide and search for “categories.” The user guide is found at the following URL: <http://www.attask.com/app-support-i18n/>.

The example code creates a category with a NULL parameter for simplicity sake. Also, note that an Issue is referred to as an OpTask in the source code.

```
// Create an issue category.
CategoryBean category = new CategoryBean();
category.setName("Category Name");
category.setGroupID(groupID);

// These are required fields, but NULL or empty values are OK.
category.setDescription("This is a category description");
category.setParameterIDs(null);
```

```

// This field is also required and must use one of the values listed in
// the provided Javadoc documentation.
category.setCatObjCode("OPTASK");

// Create the category and store its ID.
LOG.info("Creating an issue category");
categoryID = _api.addCategory(_sessionID, category);

```

9. **Creating and Editing a Categorized Issue.** The procedure for creating a data type should be fairly routine to you at this point, as it is very similar to creating a project or task. This portion of code demonstrates how you can append a category that you have created to an issue (in bold).

```

// Create a categorized issue.
OpTaskBean optask1 = new OpTaskBean();
optask1.setName("Issue Name");
optask1.setProjectID(projectID);
optask1.setCategoryID(categoryID);

// These are required fields, but NULL or empty values are OK.
optask1.setQueueTopicID(_intNull);
optask1.setRoleID(_intNull);
optask1.setOwnerID(_intNull);
optask1.setAssignedToID(_intNull);

// This field is also required and must use one of the values listed in
// the provided Javadoc documentation.
optask1.setOpTaskType("ISU");

// This field is not required, but must use one of the values listed in
// the provided Javadoc documentation.
optask1.setPriority(1);

LOG.info("Creating a categorized issue");
issueID = _api.addOpTask(_sessionID, optask1);

```

10. **Cleaning Up Your Code.** @task does not provide structure for self-cleaning code. Thus, you are responsible for making sure any objects that you no longer want are desired. The following is an example of how this could be accomplished.

```

if (_sessionID != null) {
    try {
        // Clean up.
        if (issueID != _intNull && issueID != 0) {
            int deletedIssue = _api.deleteOpTask(_sessionID, issueID);
        }
        if (categoryID != _intNull && categoryID != 0) {
            int deletedCategory = _api.deleteCategory(_sessionID,
                categoryID);
        }
        if (taskIDs != null && taskIDs.length > 0) {
            int[] deletedTasks = _api.deleteTasks(_sessionID,
                taskIDs);
        }
        if (projectID != _intNull && projectID != 0) {
            int deletedProject = _api.deleteProject(_sessionID,
                projectID);
        }

        // Logout
        _api.logout(_sessionID);
    }
}

```

You should now have a good framework with which to create your own plug-ins and components. For any other information and tips, please check the documentation and visit the @task developer's home page, www.attask.com/developer. A developer's forum is also available from this home page.