Remoting Documentation

Remote scripting provides full access to the AnthillPro object model via a remote interface. This provides access to all of the administration features of AnthillPro, as well as the information that is available via the Dashboard. Of course, security (authentication and authorization) are fully enforced when using the remote scripting API.

Running Remote Scripts

The easiest way to access AnthillPro remote scripting is through the built-in BeanShell integration. This allows you to open a BeanShell interactive session and interact with an AnthillPro server from the command line. Use the command below to start a BeanShell session in interactive mode:

- On Linux or Unix: `./ah3client.sh`
- On Windows: `ah3client.cmd`

You can also execute an existing BeanShell script using the commands below:

- On Linux or Unix: `./ah3client.sh filename [args]`
- On Windows: `ah3client.cmd filename [args]`

Remoting and SSL

If the AnthillPro server is configured to use SSL, you will need to modify `ah3client.*` file located in the `..\anthill3-dev-kit\remoting\bin` directory. In addition, you may also need to get a copy of the `ah3.keystore` from an administrator if you can't access the server's `..\conf` directory (however, if SSL was properly configured, you should have the keystore already installed on your local machine).

For **Windows**:

1. Go to the `..\anthill3-dev-kit\remoting\bin` directory.
2. Open `ah3client.cmd` in a text editor.
3. Find `"%JAVA_HOME%\bin\java" -cp "%CLASSPATH%"` (usually near the bottom of the file).
4. Insert the following commands:

   ```
   -Danthill3.client.ssl.keystore="C:\path\to\keystore\ah3.keystore" -Danthill3.client.ssl.keystore.pwd="mypassword"
   ```

   Note that `bsh.Interpreter %*` must directly follow the new commands.
5. Save your changes.

For **UNIX-type systems**:

1. Go to the `./anthill3-dev-kit/remoting/bin` directory.
2. Open `ah3client.sh` in a text editor.
3. Find `$JAVA_HOME/bin/java -cp $CLASSPATH` (usually near the bottom of the file).
4. Insert the following commands:

   ```
   -Danthill3.client.ssl.keystore="/path/to/keystore/ah3.keystore"          -
   Danthill3.client.ssl.keystore.pwd="mypassword"
   ```

   Note that `bsh.Interpreter "$@"` must directly follow the new commands.
5. **Save** your changes.

My First Script

The script below can be used to obtain a list of projects from the AnthillPro server and then print the name of each project. The first few lines of the script import the required packages. You then set some local variables that are used later in the script. Next, open a connection with the AnthillPro server using the `anthill client` class. You then pass in the location of the AnthillPro server (its host and port) as well as the user's authentication information to the `AnthillClient.connect()` method. The user name and password will define the security context within which all activity performed by the remote script will run. Only data that the specified user can access will be accessible via the script. And only the actions (such as initiating builds on projects) to which the user has permission will succeed.

```
import com.urbancode.anthill3.main.client.AnthillClient;
import com.urbancode.anthill3.persistence.UnitOfWork;
import com.urbancode.anthill3.domain.project.*;

String serverHost = "localhost";
int serverPort = 4567;
String userName = "user";
String password = "password";

// obtain connection to the Anthill server
AnthillClient anthill = AnthillClient.connect(serverHost, serverPort,
                                              userName, password);

// create a Unit of Work
UnitOfWork uow = anthill.createUnitOfWork();

// Project
Project[] projects = ProjectFactory.getInstance().restoreAll();

for (int i = 0; i < projects.length; i++) {
  print(projects[i].getName());
}

uow.commitAndClose();
```

Configuring a New Project

The New Project Script creates a new project on the AnthillPro server. The project that is created in this example is the CVS-Anthill-Example project that ships with AnthillPro. After the script is executed, the new project will become visible via the browser-based interface.

The first part of the script imports all the required packages. The second part of the script defines some variables that are used later on. Next, the connection to the AnthillPro server is made using the `anthill client` class. Next, the script creates a `UnitOfWork` object. All interaction with the persistent storage mechanism uses `UnitOfWork` objects. `UnitOfWork` is the equivalent of a business transaction or what is sometime called a long-running transaction. It provides a scope for all changes that should be executed together. Once a `UnitOfWork` object has been created, you begin making changes (such as creating a new object).

Notice that the last line in the script calls the `UnitOfWork.commitAndClose()` method. This method instructs the current `UnitOfWork` to commit all the changes performed within its scope to the persistent storage mechanism. It does this by first sending all the changes to the AnthillPro server and instructing it to store those changes in the database.

The next block of code in the script creates a CDs-repository object. The configuration of the repository performed here, via the script, provides exactly the same data as the configuration that would be performed via a web browser.

After the repository is created, you create the project object and start configuring it. Assign a status group to the project, assign a stamp style group, an environment group, and then configure the quiet period. Create a source configuration object that uses the CVS repository and which accesses the Anthill-Example module in that repository. Next, configure a build job with a single Ant builder.

After that, create a simple activity to execute the build job. The simple activity uses an agent filter that will select any agent in the server group. A workflow is then created and the simple activity is linked to the workflow via a workflow activity object. And since you want the workflow to be an originating workflows, create a build-profile object and associate it with the workflow. Underneath, the difference between originating and non-originating workflows is that originating workflows have corresponding build-profile objects.

After configuring all that, call the `commitAndClose()` method on the unit of work. This actually causes the changes to be written to the database and become available to the AnthillPro server. Once this statement is executed, you can pull up the AnthillPro web interface to see the project.

Conclusion

The remote scripting capability is very powerful and can be used to perform additional automation on AnthillPro. When faced with management of a large number of projects, the remote scripting facility can be used to make mass updates. Notice that the remote scripting facility uses BeanShell as a wrapper for a Java API. It is possible to use the Java API directly without BeanShell and thus write Java client applications for AnthillPro.