

October 2014

Automic JAVA API

Advanced

Brendan Sapience – bsp@automic.com

Our Goal

- Understanding how the AE Java API works
- How to contribute to the common Code and enhance it

Prerequisites

- You need some Java exposure
- A Java IDE must be installed
- An AE instance must be available for testing
- Atomic AE Java API Extended source code must be downloaded*
- You need a minimal knowledge of Git

* If you don't have it: see next slide

*Getting the AE Java API Extended source code

It is publicly stored on Github here (Git):

<https://github.com/brendanSapience/UC4-Automic---Java-API-Framework-Simplified>

Our Process

Step 1
Understanding the structure of the Code

Step 2
Understanding the main classes

Step 3
Contributing

Code Structure

Automatic AE Java API Extended

src

com.automic

com.automic.objects

com.automic.tests

com.automic.utils

Referenced Libraries

uc4.jar - C:\Users\bsa\Go

JRE System Library [J2SE-1.5]

README.md

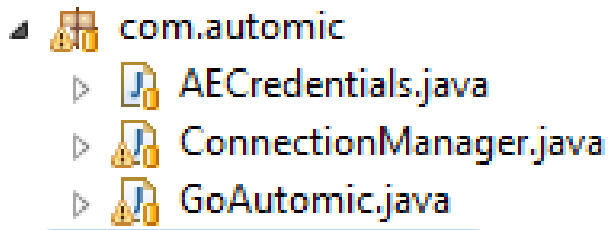
Connection Objects (credentials,
connection method etc.)

Object Specific classes (JOBS,
JOBP, HOSTGROUP, etc.)

Common utilitarian methods

AE Central API (imported from the
thick client)

Code Structure



- Contains all connection parameters (Login, Password, Port, IP Address etc..)
- Should be modified to your environment settings before anything else is done
- Contains the actual authentication method to AE (it uses the information from the AEDCredentials class)
- It contains the “Main” method (the starting point for the execution of this java code). The main itself simply calls tests (from the test package)




Code Structure

com.automic.objects

- ActivityWindow.java
- AutomationEngine.java
- Calendars.java
- Common.java
- Folders.java
- Hostgroups.java
- Jobplans.java
- Jobs.java
- Logins.java
- ObjectBroker.java
- ObjectTemplate.java
- Promptsets.java
- Queues.java
- Scripts.java
- Usergroups.java
- Variables.java

- The **ObjectBroker** is the most important class here. It is the one that will give you access to any Object-specific method. It should be instantiated in the “main” method (see previous slide)
- All Object classes inherit from the **ObjectTemplate** class.
- The **Common** class contains methods that are generic to all objects (open, save, delete, close, create. Execute etc.)
- The **AutomationEngine** class contains methods specific to the AE (get the list of agents, server details etc.)

Code Structure

▲  com.automic.utils
 ▶  AgentTypeEnum.java
 ▶  ObjectTypeEnum.java

- **ObjectTypeEnum** is used as an object type for the methods related to retrieving lists of specific objects in the **Common** class
- **AgentTypeEnum** is used as an agent type for the methods related to retrieving lists of specific agents in the **AutomationEngine** class

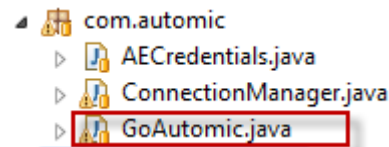
If you want to add a new AE object to work with:

- Add the object class under **com.automic.objects**
- Declare your object class in the **ObjectBroker** class

If you want to add methods to an existing object:

- Add your methods in the corresponding object class

Main Classes



```
public static void main(String argv[]) throws IOException {
```

```
// 1- First, use the static connection object to initiate the connection  
//    (see ConnectionManager class for details)
```

```
Connection conn = new ConnectionManager().authenticate(  
    AECredentials.AEHostnameOrIp, AECredentials.AECPPort,  
    AECredentials.AEClientToConnect, AECredentials.AEUserLogin,  
    AECredentials.AEDepartment, AECredentials.AEUserPassword,  
    AECredentials.AEMessageLanguage);
```

```
// 2- initialize an Object broker object,  
//    it gives you access to all object methods
```

```
ObjectBroker broker = new ObjectBroker(conn, false);
```

```
// 3- use the Object Broker!
```

```
ArrayList<IFolder> mylist = broker.folders.getAllFolders(true);
```

1- Always authenticate first

2- instantiate the Object Broker
second

3- use the Object Broker to access
object-specific methods (we invoke
a method on Folders in this
example)

Main Classes

- com.automic.objects
 - ActivityWindow.java
 - AutomationEngine.java
 - Calendars.java
 - Common.java
 - Folders.java
 - Hostgroups.java
 - Jobplans.java
 - Jobs.java
 - Logins.java
 - ObjectBroker.java

```
public class Folders extends ObjectTemplate{  
  
    public Folders(Connection conn, boolean verbose) {  
        super(conn, verbose);  
    }  
    private ObjectBroker getBrokerInstance(){  
        return new ObjectBroker(this.connection,true);  
    }  
  
    public IFolder getRootFolder() throws IOException{  
        FolderTree tree = new FolderTree();  
        this.connection.sendRequestAndWait(tree);  
        return tree.root();  
    }  
}
```

1- IF adding a new type of object (an object class), always extend the Object Template class

2- IF adding a new type of object (an object class), its constructor should use a Connection object and boolean

3- IF adding a new type of object (an object class), add a getBrokerInstance method

Main Classes

- com.automic.objects
 - ActivityWindow.java
 - AutomationEngine.java
 - Calendars.java
 - Common.java
 - Folders.java
 - Hostgroups.java
 - Jobplans.java
 - Jobs.java
 - Logins.java
 - ObjectBroker.java

```
public class ObjectBroker {  
  
    private Connection connection;  
  
    public Folders folders;  
    public Jobs jobs;  
    public Common common;  
  
    public ObjectBroker(Connection conn, boolean verbose){  
        this.connection = conn;  
  
        folders = new Folders(this.connection, verbose);  
        jobs = new Jobs(this.connection, verbose);  
        common = new Common(this.connection, verbose);  
    }  
  
    public void setConnection(Connection conn){  
        this.connection = conn;  
    }  
}
```

1- IF adding a new type of object (an object class), declare it in the Object Broker class

2- IF adding a new type of object (an object class), instantiate it in the Object Broker constructor

Remember that:

- Before you modify the code, make sure you **FETCH** the code from the public repository
- The code is collaborative
- If you add something, make it as **generic and reusable** as possible
- If you want to contribute to the project, simply fork the repo and send a pull request