IBM Rational Build Forge

Java API guide

IBM Corporation

Level: Intermediate

May 2008

Contents

Architecture 3

Getting started 4

Connection class 4

Main transactions 4

- Searching 4
- Creating 5
- Updating 5
- Deleting 6
- Managing jobs 6

Sample code 6

- Listing projects 6
- Creating and updating a server 7
- Starting a project 7
- Creating an Access Group 8

Resources 9

The Java™ API was made available in the IBM® Rational® Build Forge® Version 7.0.1, which enables programmatic interaction with the engine for the execution of projects and management of Build Forge data. Javadoc HTML documentation can be accessed at the http:///[installedhost]/clients/java/docs. It provides detailed information on packages, classes and methods. This documentation serves its purpose as a comprehensive reference, but it does not provide guidance on use patterns.

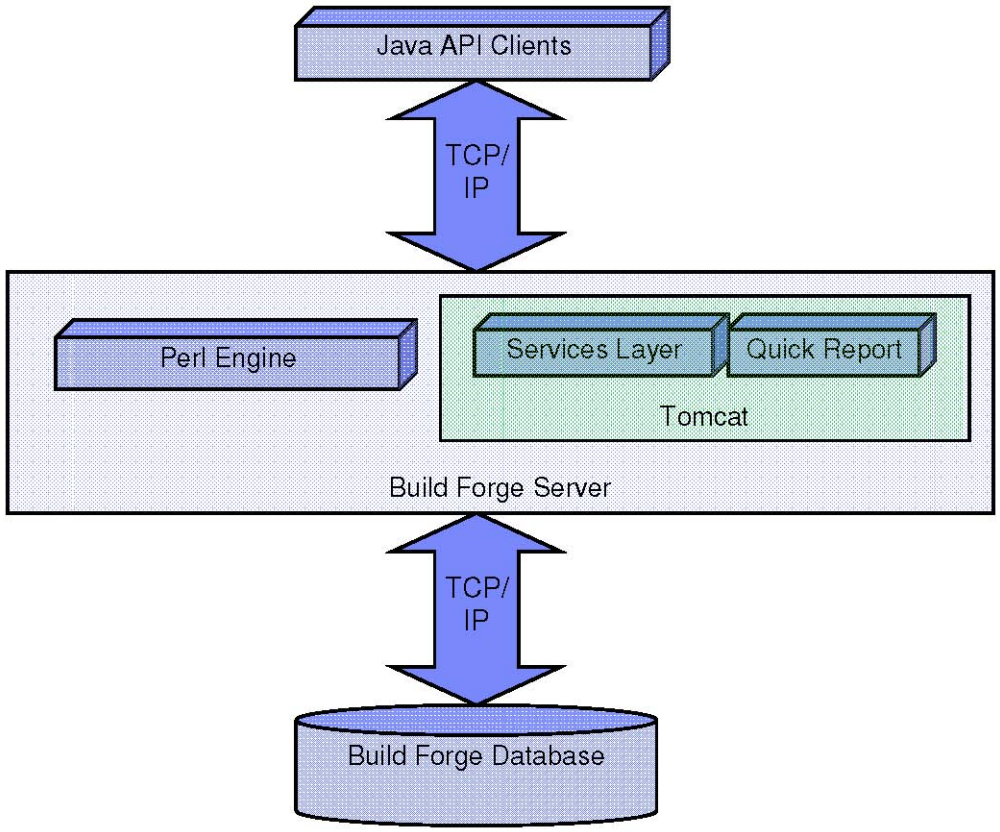
This guide is to complement the reference material, thus constituting a guide to the API use by describing its architecture, installation procedure, and main transactions, plus providing sample code to exemplify the main concepts.

Architecture

Build Forge v7.0.1 release introduces a Services layer that runs in the context of the Apache Tomcat Servlet Engine. Both the Java API and the new Perl API (as opposed to the 7.0 Perl API, which is still included with Build Forge) are built on top of this new Services layer, as illustrated in the diagram that follows.

In Figure 1, notice that this layer can be accessed only through the APIs, because it does not have an interface that can be invoked through the SOAP protocol.

Figure 1. Architecture showing Services layer



Getting started

The Build Forge Java API requires Java Development Kit 1.5. After download, run the file and follow the wizard instructions to complete the installation. The default installation directory for the IBM SDK is C:\Program Files\IBM\Java50).

1. Edit the *PATH* environment variable to include this directory: **`C:\[SDK-Installation-dir]\bin`**
2. Run this command to test the SDK: `java -version`

The next step is to install the Java API (rbf-services-client-java.jar) from the URL http://[installedhost]/clients:

1. Download the API by clicking on the Java Archive, or JAR file link, under the **Services Layer->Java Client list** bullet in the Web page. The recommended directory for storing the jar file is C:\[SDKInstallation-dir]\jre\lib\ext.
2. After downloading the JAR file, edit the *CLASSPATH* environment variable to include the pathname where it is stored.

Authentication is done through this method:
`java.lang.String authUser(java.lang.String username, java.lang.String password)`

The APIClientConnection class has three constructors:

- APIClientConnection () – localhost, default port (port 3966).
- APIClientConnection (java.lang.String hostname) – hostname specified, default port.
- APIClientConnection (java.lang.String hostname, int port) – hostname and port specified.

Connection class

The `com.buildforge.services.client.api.APIClientConnection` class is the starting point of the programmatic interface. An object of this class must be instantiated to establish a connection and authenticate to the services layer. Furthermore, a valid connection object is required for the constructor of all of the model classes (classes modeling the Build Forge concepts, such as Collectors, Selector, and Projects).

The following code excerpt provides an example for connecting to the localhost:3966 and authenticating as user `root`, password `root`:

```
APIClientConnection conn = new APIClientConnection ();
conn.authUser ("root", "root");
```

The conn object can then be passed as arguments to constructors or `search` class methods, as discussed in the topics that follow.

Main transactions

This section covers the main types of transactions offered by the API, namely: searching, creating, updating, and deleting an object.

Searching

The model classes offer class methods to search instances of that class in the Build Forge database, based on different criteria. These search methods will either return a list of

instances that satisfy the search condition or a single object when the search method requires a specific identification as an argument (for example, by ID or name).

This sample code returns the list of all objects of the `Project` class in the database to which the logged-in user has access:

```
List<Project> projList = Project.findAll (conn);
```

where **conn** is an instance of the class APIClientConnection

To search for a specific project based on its name, use this method:

```
Project proj = Project.findByName (conn, “project10”);
```

The objects returned in a list, as shown previously, do not have all of the attributes loaded for performance reasons. To completely instantiate an object, the search method based on a unique identifier returning one object, not a list, must be used, as exemplified here:

```
List<Project> projList = Project.findAll (conn); String projName =
projList.get (0).getName (); Project proj = Project.findByName (conn,
projName);
```

Creating

Follow these steps to create an object:

1. Call the `constructor` class, referencing the **APIClientConnection** object.
2. Make a sequence of calls to the various `set` methods to assign values to the object’s attributes.
3. Call the `create` method.

This sample code creates a new project in the Build Forge database:

```
Project proj = new Project (conn);
proj.setName (“project1”);
proj.setActive (true);
proj.setSelectorId (“JavaSelector”);
proj.create ();
```

Notice that the project is not actually committed to the database until the call to the `create` method.

Updating

An object must be instantiated through a call to the `create` method or a `search` method before it can be updated. When the object has been instantiated, calls to the various `set` methods to update its attributes can be made, followed by a call to the `update` method to persist those modifications in the database, as shown here:

```
Project proj = Project.findByName (conn, “project1”);
proj.setTag (“BUID_$B”);
proj.setSticky (false);
proj.update ();
```

Deleting

As the update transaction, deleting usually requires an instantiated object through the `create` method or a `search` method. When the object has been instantiated, it can be removed from the Build Forge database though a call to the `delete` method:

```
Collector collect = Collector.findById (“JavaServerCollector”);
collect.delete ();
```

Some model objects allow static calls when multiple deletion is allowed, such as: `Register.deleteAllRegistersByBuild (APIClientConnection conn, int buildId).`

Managing jobs

Builds can be started using the Project class and then cancelled using the Build class. The method “fire” of the Project class starts the build and returns on instance of the Build class representing the recently started build. If the build needs to be cancelled, the “cancel” method can be invoked against the build instance, as shown here:

```
Project proj = Project.findByName (conn, “project1”);
Build newJob = proj.fire ();
…
newJob.cancel ();
```

Notice that the `cancel` method won’t work immediately after calling `fire`, because the engine would not have picked up the project for execution at that point. It is also possible to cancel running builds by using the `findRunning` method of the `Build` class, iterating though the list and calling the `cancel` method for the desired builds:

```
List<Build> BuildList = Build.findRunning (conn);
for (Build b: BuildList) {
      b.cancel ();
}
```

Sample code

This section illustrates the use of the API through complete Java programs to perform useful, common tasks, such as listing projects in the database and creating a new server. The programs that follow compile and work correctly but are not at the required quality level for deployment, because they do not catch exceptions and do not accept input for parameters, such as user login.

Listing projects

```
import java.io.IOException;
import java.util.List;\

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.client.dbo.*;
import com.buildforge.services.common.api.APIException;

public class list_project {
      public static void main (String[] args) throws Exception {
            APIClientConnection conn = new APIClientConnection ();
            conn.authUser ("root", "root");
```

```
            List<Project> ProjectList = Project.findAll (conn);
            for (Project p : ProjectList){

                  System.out.println (p.getName());
            }
      }
}
```

Creating and updating a server

```
import java.io.IOException;
import java.util.List;

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.client.dbo.*;
import com.buildforge.services.common.api.APIException;

public class create_server {
      public static void main (String[] args) throws Exception {
            APIClientConnection conn = new APIClientConnection ();
            conn.authUser ("root", "root");
            Server testServer = new Server (conn);
            testServer.setActive (true);
            testServer.setHost ("localhost");
            testServer.setLevel (1);
            testServer.setName ("labserver");
            testServer.setPath ("/builds");
            testServer.create ();
            testServer.setAuthId ("Administrator");
            testServer.update ();
      }
}
```

Starting a project

```
import java.io.IOException;
import java.util.List;

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.client.dbo.*;
import com.buildforge.services.common.api.APIException;

public class start_project {
      public static void main (String[] args) throws Exception {
      String projName = args[0];
      APIClientConnection conn = new APIClientConnection ();
      conn.authUser ("root", "root");
      Project proj = Project.findByName (conn, projName);
      proj.fire ();

      }
}
```

Creating an Access Group

```
import java.io.IOException;
import java.util.List;

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.client.dbo.*;
import com.buildforge.services.common.api.APIException;
import com.buildforge.services.common.api.Permission;

public class group {

      public static void main (String[] args) throws Exception {

            String newGroupName = args[0];
            String inheritGroupName = args[1];

            APIClientConnection conn = new APIClientConnection ();
            conn.authUser ("root", "root");

            AccessGroup newGroup = new AccessGroup (conn);
            newGroup.setName (newGroupName);
            newGroup.create ();

            /* Grant the new access group permission to delete project */
            newGroup.grantPermission (Permission.DeleteProject);

            /* Inherit permissions from existing access group */
            AccessGroup inheritGroup = AccessGroup.findByName (conn,
            9 © Copyright IBM Corp. 2008
            inheritGroupName);

            newGroup.addInheritedGroup (inheritGroup.getLevel ());

            /* make the root user a member */
            User userRoot = User.findByLogin (conn, "root");
            newGroup.addMemberUser (userRoot.getUserId ());

            newGroup.update ();
      }
}
```

Trademarks

Resources

Learn

- Visit the Rational software area on developerWorks for technical resources and best practices for Rational Software Delivery Platform products.
- Subscribe to the developerWorks Rational zone newsletter. Keep up with developerWorks Rational content. Every other week, you'll receive updates on the latest technical resources and best practices for the Rational Software Delivery Platform.
- Subscribe to the Rational Edge newsletter for articles on the concepts behind effective software development.
- Subscribe to the IBM developerWorks newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.
- Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

- Download trial versions of IBM Rational software.
- Download IBM product evaluation versions and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.