IBM Rational Build Forge

Getting started with the API

Jan 2011

Version 1.0

Author:
Robert W. Harris Jr.

Note
Before using this information and the product it supports, read the information in “Notices,” on page 23.

Table of Contents

Overview

With the IBM® Rational® Build Forge® application programming interface (API), you can interact with Rational Build Forge through the Java and Perl programming languages. Rational Build Forge provides client libraries containing a set of functions that you can use to control the management console as if you were interacting with the web interface. However, the real power of the API is its ability to automate tasks such as updating environment groups and running builds.

Assumptions

This guide assumes the proper development kits and libraries, such as a JDK, are in place to develop applications for the desired environment. Knowledge of the desired programming language and development environment is necessary as well.

Installation

Download the API clients from the computer where you run the Rational Build Forge management console web interface. To access the client files, load the http://<*host*>:<*port*>/clients/ address in your browser and use the links in the "Services Layer" section.

Java client

To install the Java API client:

1. Go to the address listed above. Download the jar file available through the "JAR file" link.
2. Add the downloaded "rbf-services-client.jar" file to the build path of the Java project.

Note: The Java API requires JDK 1.5 or later.

Perl client

To install the Perl API client:

1. Go to the address listed above.
2. Download the zip file available through the **Zip file** link.
3. Extract the contents of the .zip file.
4. In a shell prompt, change the working directory to the extracted "rbf-services" directory and run these commands. Note: Depending on operating system and setup, administrator or super user privileges might be required.

```
perl Makefile.PL
make
make install
```

Note: You can use the Perl API client by modifying the include path in the Perl script to include the extracted “rbf-services/lib/BuildForge” directory or by manually copying the “rbf-services/lib/BuildForge” directory to an existing Perl include path.
Note: The Perl API requires Perl version 5.8 or higher.
Note: The Perl API requires the following modules and libraries:

Encode
Exporter
Data::Dumper
IO::Handle
IO::Socket
IO::Socket::INET
Socket

Client documentation

The JavaDoc and PerlDoc references are available in the “Java Client” and “Perl Client” sections of the download page, respectively. For offline use, you can download the JavaDoc reference in .zip format and the PerlDoc reference in a .tar.gz format. You can also access both references directly from the Rational Build Forge management console web interface.

Create a user for the API

For optimal behavior, create a user that is used only for API connections. Because an API connection acts as a user session, using the same user account for the web interface and the API at the same time invalidates one of the sessions. For example, if a user authenticates with the web interface while an API script is running as that user, the API script throws an authentication exception because its session is invalidated. Conversely, if a user is logged into the web interface and an API script starts as that user, the web interface session is invalidated and the user is logged out.

Usage

Connecting to the Rational Build Forge services layer

First import the client packages into the development environment. Complete this task with the "import" lines (Java) and the "use" line (Perl), as shown in the samples below. Next, the API user needs to authenticate with the services layer. The following code demonstrates how to create a connection to the services layer and authenticate a user.

Java:

```java
import java.io.IOException;

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.common.ServiceException;

public class BuildForgeAPIDemo {
	public static void main(String[] args) throws IOException {
		try {
			APIClientConnection conn = new APIClientConnection("localhost", 3966);
			conn.authUser("apiuser", "password");
		}
		catch (ServiceException e) {
			System.out.println("Caught Exception: " + e.getMessage());
			e.printStackTrace();
		}
	}
}
```

Perl:

```perl
#!/usr/bin/perl
use strict;

use BuildForge::Services;

my $conn = new BuildForge::Services::Connection('localhost', 3966);
$conn->authUser('apiuser', 'password');
```

For this example, the services layer host is “localhost,” the services layer port is “3966,” and the API user is “apiuser” with password “password.” Modify those parameters to match the host and port of the services layer and the user account created for the API as recommended above. If the services layer is running on the standard port, you can safely omit the port parameter.

To connect as an LDAP/domain user, simply use the domain string as a third parameter in the authentication statement. For other advanced connection options, such as changing the data transfer protocol or connecting over SSL, consult the documentation in the Rational Build Forge Information Center.

Versions

The Rational Build Forge API client has versions that correspond to the management console version. Older API clients work with the same and any newer versions of the management console. However, the client cannot communicate with management console versions older than the client version. For optimal performance, always use the most recent client package. This practice ensures you have access to current implementations of methods and currently available features.

Programming with the API

Once authenticated with the services layer, you can use the API to interact with the Rational Build Forge management console. Note that all access group privileges are enforced for an API user the same way they are for a web interface user. Only objects the user normally has access to can be interacted with through the API. For example, assume there are three projects in the console: A, B, and C. Suppose a user logs into the web interface and has access to view projects A and B. In the web interface, only those two projects are displayed. Now suppose the same user searches for all projects in the console through the API. Only projects A and B are returned in the search, as they are the only ones the user has access to view. The same idea applies to modifying metadata and starting builds. All permissions are enforced for user actions regardless of the interface.

The Java and Perl client libraries provide almost all the features available in the web interface. However, there are a few actions that the API cannot do in a reasonable way. One notable limitation is step selection when starting builds. Starting builds with all steps selected is the default behavior; however, there is no easy way through the API to choose only certain steps to be run due to the nature of the selection process. In many cases, there are workarounds for the limitations, but they require various levels of depth that can often be avoided by using the web interface.

The Java and Perl client libraries have almost a one-to-one feature parity with each other. The Perl client libraries lack a few simple functions that the Java client libraries have, but there are usually simple workarounds for such situations. As demonstrated in the “Gathering information” example on page 10, there is no direct way in the Perl API to tell whether a project is a library. However, by definition, a Build Forge library is just a project without a selector. Thus, a simple null check on the selector ID produces the desired result.

Naming conventions in the API methods are not completely standardized. An example is the use of the terms “name” and “description”. These two terms almost always refer to what an object is called as it is presented to the user. Some objects such as projects have functions that use the term “name” while other objects like crons use the term “description.” The API documentation is a useful reference for determining correct methods for each object type.

Fetching and saving data with the API

The process of fetching data through the API is simple; however, there is an important concept to remember. After data is pulled from the services layer through the API (and through the web interface), it is stale until refreshed. The API does not automatically refresh data objects after they are initially retrieved; data objects are only updated when they are retrieved again. In essence, when an object is pulled from the services layer, a local copy is created to operate on. This behavior can lead to race conditions and problems, such as modifications getting overwritten, depending on how and when the data is accessed. This is the nature of a concurrent user application, and should be taken into consideration with any API program that modifies data.

When saving an object that has been fetched from the services layer through the API, you must use its update method. Because retrieving an object creates a local copy, the "set" methods of the object only modify the local copy of the data. To commit modifications to the services layer, you must call the "update()" method. Every data object that can be directly updated has an update method. After this method is called, any new data is validated and saved if it is acceptable. If there is an issue, such as lack of permissions or invalid data, an exception is thrown and the data will not be saved. Note that a successful update returns the updated object from the services layer.

One method of correctly retrieving and modifying data is demonstrated in the "Updating an environment" example on page 12.

Examples

Some of the most common uses of the API are gathering information, updating environments, and starting builds. The following examples show one way to implement these use cases. Examples are written in both Java and Perl.

Note that these examples are written for Build Forge 7.1.2.x. Some of the code might not work in previous versions.

Gathering information – listing projects and libraries a user can access

Java:

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

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.client.dbo.Project;
import com.buildforge.services.common.ServiceException;

public class BuildForgeAPIDemo {
      public static void main(String[] args) throws IOException {
            try {
                  APIClientConnection conn = new APIClientConnection("localhost", 3966);
                  conn.authUser("apiuser", "password");

                  /*
                   * Finds all project objects in the system to which the user has access.
                   * Only the immediate project data will be returned, so no steps will
                   * be available in the returned list.
                   */
                  List<Project> projects = Project.findAll(conn);
                  System.out.println("Number of projects/libraries: "+projects.size());

                  for (Project project : projects) {
                        if (project.isLibrary()) {
                              System.out.print("Library: ");
                        }
                        else {
                              System.out.print("Project: ");
                        }
                        System.out.println(project.getName());
                  }
            }
            catch (ServiceException e) {
                  System.out.println("Caught Exception: " + e.getMessage());
                  e.printStackTrace();
            }
      }
}
```

Perl:

```
#!/usr/bin/perl
use strict;

use BuildForge::Services;

my $conn = new BuildForge::Services::Connection('localhost', 3966);
$conn->authUser('apiuser', 'password');

#Returns a reference to an array of BuildForge::Services::DBO::Project
#objects corresponding to all projects in the database that the user
#has access to. These objects will not contain their Step subobjects.
my $projects = BuildForge::Services::DBO::Project->findAll($conn);
print "Number of projects/libraries: ".@$projects."\n";

foreach my $project (@$projects) {
      if (!$project->getSelectorUuid()) {
            print "Library: ";
      }
      else  {
            print "Project: ";
      }
      print $project->getName()."\n";
}
```

Updating an environment – changing the value of an existing environment variable

Java:

```
import java.io.IOException;

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.client.dbo.Environment;
import com.buildforge.services.client.dbo.EnvironmentEntry;
import com.buildforge.services.common.ServiceException;

public class BuildForgeAPIDemo {
	public static void main(String[] args) throws IOException {
		try {
			APIClientConnection conn = new APIClientConnection("localhost", 3966);
			conn.authUser("apiuser", "password");

			String envName = "Environment 1";
			Environment env = Environment.findByName(conn, envName);
			if (env != null) {
				String entryName = "Entry 1";
				EnvironmentEntry entry = env.getEntry(entryName);
				if (entry != null) {
					System.out.println("Old value: "+entry.getParameterValue());
					entry.setParameterValue("new value");
					entry = entry.update();
					System.out.println("New value: "+entry.getParameterValue());
				}
				else {
					System.out.println("Could not find environment entry: "+entryName);
				}
			}
			else {
				System.out.println("Could not find environment: "+envName);
			}
		}
		catch (ServiceException e) {
			System.out.println("Caught Exception: " + e.getMessage());
			e.printStackTrace();
		}
	}
}
```

Perl:

```perl
#!/usr/bin/perl
use strict;

use BuildForge::Services;

my $conn = new BuildForge::Services::Connection('localhost', 3966);
$conn->authUser('apiuser', 'password');

my $envName = 'Environment 1';
my $env = BuildForge::Services::DBO::Environment->findByName($conn, $envName);

if(defined($env)) {
      my $entryName = "Entry 1";
      my $entry = $env->getEntry($entryName);
      if (defined($entry)) {
            print "Old value: ".$entry->getValue()."\n";
            $entry->setValue("new value");
            $entry = $entry->update();
            print "New value: ".$entry->getValue()."\n";
      }
      else {
            print "Could not find environment entry: ".$entryName."\n";
      }
} else {
      print "Could not find environment: ".$envName."\n";
}
```

Starting a build – kicking off a build of a project with all default settings

Note that this method of starting a build does not block; that is, control is immediately returned to the client. You can monitor the build monitored by explicitly reloading it with one of the find methods.

Java:

```
import java.io.IOException;

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.client.dbo.Build;
import com.buildforge.services.client.dbo.Project;
import com.buildforge.services.common.ServiceException;

public class BuildForgeAPIDemo {
      public static void main(String[] args) throws IOException {
            try {
                  APIClientConnection conn = new APIClientConnection("localhost", 3966);
                  conn.authUser("apiuser", "password");

                  String projectName = "Project 1";
                  Project project = Project.findByName(conn, projectName);
                  if (project != null) {
                        Build newBuild = Build.fire(conn, project.getUuid());
                        System.out.println("<"+newBuild.getTag()+"> of project <"+projectName+"> fired.");
                  }
                  else {
                        System.out.println("Could not find project: "+projectName);
                  }
            }
            catch (ServiceException e) {
                  System.out.println("Caught Exception: " + e.getMessage());
                  e.printStackTrace();
            }
      }
}
```

Perl:

```
#!/usr/bin/perl
use strict;

use BuildForge::Services;

my $conn = new BuildForge::Services::Connection('localhost', 3966);
$conn->authUser('apiuser', 'password');

my $projectName = 'Project 1';
my $project = BuildForge::Services::DBO::Project->findByName($conn, $projectName);

my $newBuild;
if (defined($project)) {
      $newBuild = BuildForge::Services::DBO::Build->fire($conn, $project->getUuid());
      print "<".$newBuild->getTag()."> of project <".$projectName."> fired.\n";
} else {
      print "Could not find project: ".$projectName."\n";
}
```

Advanced examples

More advanced uses of the Rational Build Forge API include modifying a build before starting it. Often the environment for a single build needs to be updated without making modifications to the project environment. The first advanced example demonstrates one way this can be done through the API.

Note that this example is written for Rational Build Forge 7.1.2.x. Some of the code might not work in previous versions.

Creating a build, modifying the build environment, and starting the build

Java:

```
import java.io.IOException;

import com.buildforge.services.client.api.APIClientConnection;
import com.buildforge.services.client.dbo.Build;
import com.buildforge.services.client.dbo.BuildClass;
import com.buildforge.services.client.dbo.BuildEnvironment;
import com.buildforge.services.client.dbo.BuildEnvironmentEntry;
import com.buildforge.services.client.dbo.Project;
import com.buildforge.services.common.ServiceException;

public class BuildForgeAPIDemo {
      public static void main(String[] args) throws IOException {
            try {
                  APIClientConnection conn = new APIClientConnection("localhost", 3966);
                  conn.authUser("apiuser", "password");

                  String projectName = "Project 1";
                  Project project = Project.findByName(conn, projectName);
                  if (project == null) {
                        System.out.println("Project <"+projectName+"> could not be found.");
                        return;
                  }

                  //Create a build of the project.
                  Build build = Build.create(conn, project.getUuid());
                  /*
                   * At this point a build has been created (not fired) with the settings from
                   * the project, along with the appropriate build artifacts such as the build
                   * environment. These artifacts can now be edited.
                   */
```

```
                    //Get the build environment uuid.
                    String buildEnvUuid = build.getBuildEnvironmentUuid();
                    //Note that this is NOT the original environment. It is an environment
                    //specific to this build. Updates to this build environment only affect
                    //this build, not the original environment.

                    System.out.println("Original information for new build:");
                    System.out.println("\tParent project: "+projectName);
                    System.out.println("\tClass: "+BuildClass.findByUuid(conn,
build.getBuildClassUuid()).getName());
                    System.out.println("\tBuild environment:");
                    //Note that some build environment entry values are null until the build is fired.
                    for (BuildEnvironmentEntry buildEnvEntry : BuildEnvironment.findByUuid(conn,
buildEnvUuid).getEntries()) {
                          System.out.println("\t\t"+buildEnvEntry.getParameterName()+" =
"+buildEnvEntry.getParameterValue());
                    }

                    //Update a build environment entry.
                    //Note that updates to the build environment are only allowed before the build
                    //is fired.
                    String buildEnvEntryName = "Entry 1";
                    BuildEnvironmentEntry buildEnvEntryToUpdate = BuildEnvironment.findByUuid(conn,
buildEnvUuid).getEntry(buildEnvEntryName);
                    if (buildEnvEntryToUpdate == null) {
                          System.out.println("BuildEnvironmentEntry <"+buildEnvEntryName+"> could not be found.");
                          return;
                    }
                    build.updateBuildEnvEntryValue(buildEnvEntryToUpdate.getUuid(), "api modified value");

                    //Fire the build and wait for it to finish
                    System.out.println("----------------------------------------");
                    System.out.println("Firing build and waiting for completion...");
                    build = Build.fireBuild(conn, build.getUuid(), 100000);
                    System.out.println("Build finished...");
```

```
                  //Check results
                  if (build.getResult().toString().equals("PASSED")) {
                        System.out.println("Build passed!");
                        System.out.println("--------------------------------------");
                        System.out.println("Information for: "+build.getTag());
                        System.out.println("\tParent project: "+projectName);
                        System.out.println("\tClass: "+BuildClass.findByUuid(conn,
build.getBuildClassUuid()).getName());
                        System.out.println("\tBuild environment:");
                        for (BuildEnvironmentEntry buildEnvEntry : BuildEnvironment.findByUuid(conn,
buildEnvUuid).getEntries()) {
                              System.out.println("\t\t"+buildEnvEntry.getParameterName()+" =
"+buildEnvEntry.getParameterValue());
                        }
                        System.out.println("\tDuration: "+build.getDuration()+" seconds");
                  }
                  else {
                        System.out.println("Build failed!");
                  }
            }
            catch (ServiceException e) {
                  System.out.println("Caught Exception: " + e.getMessage());
                  e.printStackTrace();
            }
      }
}
```

Perl:

```
#!/usr/bin/perl
use strict;

use BuildForge::Services;

my $conn = new BuildForge::Services::Connection('localhost', 3966);
$conn->authUser('apiuser', 'password');

my $projectName = "Project 1";
my $project = BuildForge::Services::DBO::Project->findByName($conn, $projectName);
if(!defined($project)) {
      print "Project <".$projectName."> could not be found.\n";
      exit;
}

#Create a build of the project.
my $build = BuildForge::Services::DBO::Build->create($conn, $project->getUuid());
#At this point a build has been created (not fired) with the settings from
#the project, along with the appropriate build artifacts such as the build
#environment. These artifacts can now be edited.

#Get the build environment uuid.
my $buildEnvUuid = $build->getBuildEnvUuid();
#Note that this is NOT the original environment. It is an environment
#specific to this build. Updates to this build environment only affect
#this build, not the original environment.

print "Original information for new build:\n";
print "\tParent project: ".$projectName."\n";
print "\tClass: ".BuildForge::Services::DBO::BuildClass->findByUuid($conn, $build->getBuildClassUuid())-
>getName()."\n";
print "\tBuild environment:"."\n";
#Note that some build environment entry values are null until the build is fired.
my $buildEnvEntries = BuildForge::Services::DBO::BuildEnvironment->findByUuid($conn, $buildEnvUuid)-
>getEntries();
foreach my $buildEnvEntry (@$buildEnvEntries) {
      print "\t\t".$buildEnvEntry->getName()." = ".$buildEnvEntry->getValue()."\n";
}
```

```
#Update a build environment entry.
#Note that updates to the build environment are only allowed before the build
#is fired.
my $buildEnvEntryName = "Entry 1";
my $buildEnvEntryToUpdate = BuildForge::Services::DBO::BuildEnvironment->findByUuid($conn, $buildEnvUuid)-
>getEntry($buildEnvEntryName);
if(!defined($buildEnvEntryToUpdate)) {
      print "BuildEnvironmentEntry <".$buildEnvEntryName."> could not be found.\n";
      exit;
}
$build->updateBuildEnvEntryValue($buildEnvEntryToUpdate->getUuid(), "api modified value");

#Fire the build and wait for it to finish
print "-------------------------------------\n";
print "Firing build and waiting for completion...\n";
$build = $build->fireBuild();
while($build->getState() ne "COMPLETED"){
      sleep 5;
      $build = BuildForge::Services::DBO::Build->findByUuid($conn, $build->getUuid());
}
print "Build finished...\n";

#Check results
if($build->getResult() eq "PASSED") {
      print "Build passed!\n";
      print "-------------------------------------\n";
      print "Information for: ".$build->getTag()."\n";
      print "\tParent project: ".$projectName."\n";
      print "\tClass: ".BuildForge::Services::DBO::BuildClass->findByUuid($conn, $build->getBuildClassUuid())-
>getName()."\n";
      print "\tBuild environment:\n";
      $buildEnvEntries = BuildForge::Services::DBO::BuildEnvironment->findByUuid($conn, $buildEnvUuid)-
>getEntries();
      foreach my $buildEnvEntry (@$buildEnvEntries) {
            print "\t\t".$buildEnvEntry->getName()." = ".$buildEnvEntry->getValue()."\n";
      }
      print "\tDuration: ".$build->getDuration()." seconds\n";
}
```

```
else {
      print "Build failed!\n";
}
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the

exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.html.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.