Automation Library (Ant, Maven) for Dynatrace

Overview

Name	Automation Library for Dynatrace to integrate Dynatrace in an automated environment such as a build or automated test environment
Description	Automation is key in different phases of the Software Development Lifecycle. Builds are automated by using build-servers that also execute all unit and integration tests to ensure the basic quality criteria for a build before it gets deployed to the next stage. Functional as well as load-tests can be automated to test applications that come out of the build-environment to ensure full functionality, performance and scalability.DynaTrace provides this Automation Library to easily integrate DynaTrace in your **Continuous Integration** or Test Automation Environment such as **Ant**, **Maven**, **Jenkins**, ...The automation library provides contextual information such as the version/revision/build number to be used by the Dynatrace Automation Test Center and visible in the Test Automation Dashlet. The library also includes reporting capabilities.
Version	1.0
Dynatrace Versions	5.x, 6.x
Author	dynaTrace software
License	dynaTrace BSD
Support	Supported
Download for Java	**Dynatrace AppMon 6.5:** • Basic Automation Library (Server REST SDK) • Automation Library for Ant • Automation Library Maven Plugin **Dynatrace 6.3:** • Automation Library with Ant task definitions and examples • Automation Library ONLY • Ant Integration ONLY • Maven Plugin **Dynatrace 6.2:** • Automation Library with Ant task definitions and examples • Automation Library ONLY • Ant Integration ONLY • Maven Plugin **Dynatrace 6.1:** • Automation Library with Ant task definitions and examples • Automation Library ONLY • Ant Integration ONLY • Maven Plugin
Download for .NET	**dynaTrace 5.x, 6.x** • NANT Task Library • MSBuild Tasks Library
Sample Downloads	• dynaTrace Automation Walkthrough including Samples for JUnit and Selenium • dynaTrace Automation Samples for Ant and Maven (6.1.0 release)
Report an Issue	https://github.com/Dynatrace/Dynatrace-Server-REST-Java-SDK/issues https://github.com/Dynatrace/Dynatrace-Ant-Plugin/issues https://github.com/Dynatrace/Dynatrace-Maven-Plugin/issues

For more information on using the Ant and Maven tasks, please see Integrate Dynatrace in Continuous Integration Builds

Description

The automation library enables FULL Automation of Dynatrace by leveraging the REST interfaces of the Dynatrace Server. The automation library includes Ant Tasks, Maven Goals and a console application to execute the following actions on the Dynatrace Server:

- **Start/Stop Session Recording**: Returns the actual recorded session name
- **Set Test Information**: Sets information about the tested built to be used by Test Automation Center
- **Clear Session**: Clears the live session
- **Reanalyze Stored Sessions**: triggers business transaction analysis of a stored session
- **Enable/Disable Profile**
- **Activate Configuration**: Activates a configuration within a system profile
- **Get Agent Information**: Either returns the number of connected agents or specific information about a single agent
- **Create Memory/Thread Dumps**: Triggers memory or thread dumps for a specific connected agent
- **Restart Server/Collector**

If you're looking to automate Dynatrace from the command line, please see the documentation of the Command Line Tool.

Releases from Dynatrace AppMon 6.5 onward

Navigate to appropriate project pages for detailed installation and usage descriptions. Each project also contains examples.

Please report issues in each GitHub project's Issue Tracker.

Server REST SDK: https://github.com/Dynatrace/Dynatrace-Server-REST-Java-SDK (needed only when you want to work with thinly-wrapped Server REST endpoints from your Java code, other plugins are based on that library)

Ant plugin: https://github.com/Dynatrace/Dynatrace-Ant-Plugin

Maven plugin: https://github.com/Dynatrace/Dynatrace-Maven-Plugin

Releases prior to Dynatrace AppMon 6.5

The Download package includes:

dtAutomation_{VERSION}.zip	Automation Library for Dynatrace {VERSION} including implemenation for Ant and Maven. Also includes a JUnit Runner for Selenium and a separate dtTaskDefs.xml that defines and documents all available Ant-Tasks.
dtAutomationForMaven_{VERSION}.zip	Automation Library packaged to be used with the Maven Plugin Repository (includes description files and correct file structure).
com.dynatrace.diagnostics.automation_{VERSION}.zip	Automation Library for specific Dynatrace version, including implementation for Ant and Maven.
dynaTraceAutomationSample.zip	The package includes a sample build.xml for Ant, pom.xml for Maven and Dashboards and configuration files to be used for the Reporting feature.

Installation

To be used with Ant

For 5.5 and 6.x

- Extract Automation Library Package to your local file system
- Under `lib/dynatrace` you find `dtTaskDefs.xml` (defines all Ant Task) and `com.dynatrace.diagnostics.automation.jar` (th e actual automation library)
- Have a look at `build.xml` in the root directory as a sample on how to call the ant tasks

To be used with Maven

- Extract `dynaTraceForMaven_{VERSION}.zip` to your local file system.
- To install the Dynatrace Maven plugin, execute the following command (for 6.2 release):

```
mvn install:install-file -DgroupId=dynaTrace -DartifactId=dtAutomation
-Dversion=6.2 -Dpackaging=maven-plugin -Dfile=<your download
location>\dtAutomation\6.2\dtAutomation.jar
```

- Define properties for the Dynatrace goals as shown in pom.xml from the sample package
- Invoke your maven goals, e.g.: `mvn dynaTrace:dtAutomation:6.2:startRecording`

The Dynatrace maven plugin has the following identification (pluginGroupId:pluginArtifactId:pluginVersion): dynaTrace:dtAutomation:6.2

Usage

With Ant

A full example that includes calling JUnit and Selenium tests can be seen in the Demo Application Package. You can also download the `build.xml` as shown later in this paragraph which is as part of dtAutomation_6.2.0.1238.zip or the dynaTraceAutomationSample_6.2.0.1238.zip sample package.
Use and import `dtTaskDefs.xml` in your existing Ant file. This file defines all TaskDefs and global properties that allow you to specify username, password, serverURL, ... Here is the full `build.xml` file that is part of the download sample package. It includes a sample call to all TaskDefs including documentation of the properties:

```
<project name="Sample DynaTrace Automation Ant file">
 <description>
  Shows how to use the Dynatrace Automation Ant Tasks
 </description>

 <!-- Import the Dynatrace Automation Tasks -->
 <import file="lib/dynaTrace/dtTaskDefs.xml"/>

 <!-- Setting defaults for Test Automation -->
 <property name="dtVersionMajor" value="1" />
 <property name="dtVersionMinor" value="0" />
 <property name="dtVersionRevision" value="0" />
 <property name="dtVersionMilestone" value="Milestone 1" />
 <property name="dtVersionBuild" value="123" />
 <property name="dtAgentGroup" value="GoSpaceFrontend" />

 <!-- Enables GoSpace and actives GoSpace Light configuration
  configuration and profilenames need are case sensitive
 -->
 <target name="EnableProfileAndActivateConfiguration">
  <DtEnableProfile profileName="GoSpace" enable="true" />
  <DtActivateConfiguration profileName="GoSpace" configuration="GoSpace light" />
 </target>

 <!-- Clears the live session of GoSpace -->
 <target name="ClearSession">
  <DtClearSession profileName="GoSpace" />
 </target>

 <!-- Sets Test Meta Data Information for Test Automation -->
 <target name="SetTestInformationForTestAutomation">
  <DtStartTest versionMajor="${dtVersionMajor}" versionMinor="${dtVersionMinor}"
versionRevision="${dtVersionRevision}"
    versionMilestone="${dtVersionMilestone}" versionBuild="${dtVersionBuild}"/>
        <echo message="Regisered test run  with id: ${dtTestrunID}"/>
  <!--
```

```
            after registering a test run, you need to pass the dtTestrunID as an
additional
            parameter to the agent injected into the JVM that's executing the tests.
            The following example assumes that dtagent property already holds basic
agent
            definition and parameters, dtTestrunID is added as an extra parameter
        -->
        <!--
        <junit fork="yes">
          <jvmarg value="${dtagent},optionTestRunIdJava=${dtTestrunID}" />
               ...
        </junit>
        -->
    </target>

 <!-- Retrieves the number of Connected Agents
  Then queries the agent information for the first connected agent and the first
GoSpaceFrontend agent
  The requested information is stored in Ant Properties that can later be used by
other tasks, e.g.: DtMemoryDump to identify the agent
  -->
 <target name="GetAgentInfo">
  <DtGetAgentInfo agentCountProperty="AgentCount" />
  <echo message="Connected Agents: ${AgentCount}"></echo>

  <DtGetAgentInfo agentNameProperty="AgentName" agentHostNameProperty="AgentHost"
agentProcessIdProperty="AgentProcessId" infoForAgentByIndex="0" />
  <echo message="First Agent: ${AgentName} - ${AgentHost} - ${AgentProcessId}"></echo>

  <DtGetAgentInfo agentNameProperty="AgentName" agentHostNameProperty="AgentHost"
agentProcessIdProperty="AgentProcessId" infoForAgentByName="GoSpaceFrontend" />
  <echo message="First GoSpaceFrontend: ${AgentName} - ${AgentHost} -
${AgentProcessId}"></echo>
 </target>

 <!-- Takes a Memory and Thread Dump from a specific agent
  We call GetAgentInfo first that retrieves the Agent Information for which the dump
gets created
 -->
 <target name="TakeMemoryAndThreadDumps" depends="GetAgentInfo" >
  <DtMemoryDump memoryDumpNameProperty="MemoryDumpName" profileName="GoSpace"
agentName="${AgentName}" hostName="${AgentHost}" processId="${AgentProcessId}" />
  <echo message="Created Memory Dump: ${MemoryDumpName}" />

  <DtThreadDump threadDumpNameProperty="ThreadDumpName" agentName="${AgentName}"
hostName="${AgentHost}" processId="${AgentProcessId}" />
  <echo message="Created Thread Dump: ${ThreadDumpName}" />
 </target>

 <!-- Start Recording a new Dynatrace Session
  The actual session name will be stored in the Ant Property "SessionName" which can
later be used as input for other tasks, e.g.: ReanalyzeSession
 -->
 <target name="StartRecording">
  <DtStartRecording profileName="GoSpace" sessionNameProperty="SessionName"
sessionName="AntSession" sessionDescription="This Session is triggered by an Ant Task"
/>
        <echo message="Start Recording SessionName: ${SessionName}" />
 </target>
```

```
 <!-- Stops current recording
  The actual session name will be stored in the Ant Property "SessionName" which can
later be used as input for other tasks, e.g.: ReanalyzeSession
 -->
 <target name="StopRecording">
  <!-- The sleep is in there so that you can manually create some purepaths in the
meantime that end up in the session -->
  <sleep seconds="10"/>

  <DtStopRecording profileName="GoSpace" sessionNameProperty="SessionName" />
  <echo message="Stopped Recording SessionName: ${SessionName}" />
    </target>

 <!-- Stops current recording and also reanalizes the session
  In this case the System Profile name comes from the global dtProfile Property
 -->
 <target name="StopRecordingWithReanalyze">
  <!-- The sleep is in there so that you can manually create some purepaths in the
meantime that end up in the session -->
  <sleep seconds="10"/>

  <DtStopRecording doReanalyzeSession="true" reanalyzeStatusProperty="ReanalyzeStatus"
/>
  <echo message="Stopped Recording SessionName and reanalized: ${SessionName} -
${ReanalyzeStatus}" />
    </target>

 <!-- Reanalyzes a specific Dynatrace Session
  We use the value stored in the SessionName property which falls set by StopRecording
 -->
 <target name="ReanalyzeSession">
  <DtReanalyzeSession sessionName="${SessionName}"
reanalyzeStatusProperty="ReanalyzeStatus" reanalyzeSessionTimeout="60000"
reanalyzeSessionPollingInterval="5000" />

  <echo message="Reanalyze finished? ${ReanalyzeStatus}" />
 </target>

 <!-- Combines the calls to Start/Stop and Reanalyze -->
 <target name="StartStopRecordingAndReanalyze"
depends="StartRecording,StopRecording,ReanalyzeSession" >
 </target>

 <!-- Combines the calls to Start and StopWithReanalyze -->
 <target name="StartStopInclReanalyze"
depends="StartRecording,StopRecordingWithReanalyze" >
 </target>

 <target name="DashboardReporting">
  <!-- Queries a single dashboard and puts the result out to an XML File using XSLT to
transform it to HTML -->
  <DtReport dashboardname="Test" source="live:GoSpace" createHtml="true"
xmlToFile="./results/report.xml"/>

  <!-- Creates a dashboard report for every transaction or webrequest that is on the
iterator dashborad
   outputs all files in the report directory
   an overview page is created to navigate through all result files
```

```
   -->
  <DtReport dashboardname="RulesDashboard" iteratorDashboard="TransactionDashboard"
source="live:GoSpace" createHtml="true" reportDir="./results"/>

  <!-- Make sure that - when using Business Transactions on the Iterator or Data
Dashboard to reanalyze stored sessions before running the report
      Either us DtReanalyzeSession or specify the reanalyzeSession="true" property for
DtReport
  -->
 </target>

 <target name="ArchitecturValidation">
  <DtArchiVal source="live:GoSpace" archiValFile="archivalrules.xml"
reportFile="./results/archivalreport.xml" errorProperty="ArchiValFailed"/>
```

```
 </target>
</project>
```

With Maven

The Dynatrace version in the maven package may still say 3.5 even though you downloaded the package for Dynatrace 5.5 or 6.0. This problem is fixed with 6.1 release.

Download the Step-by-Step Guide on How to use Maven and WebDriver with Dynatrace: MavenWebDriverIntegration.docx

A full example can be seen in the pom.xml as part of the dynaTraceAutomationSample_6.2.0.1238.zip sample package.
Also look at description of the previous Maven Plugin Page that explains how to use the reporting and other maven goals that this plugin provides.

```
<properties>
  <!-- Setting default values for Dynatrace Maven goals that operate on a system
profile -->
  <dynaTrace.username>admin</dynaTrace.username>
  <dynaTrace.password>admin</dynaTrace.password>
  <dynaTrace.serverUrl>http://localhost:8020</dynaTrace.serverUrl>
  <dynaTrace.systemProfile>GoSpace</dynaTrace.systemProfile>

  <!-- This property will be used to store the actual Session Name for e.g.:
Start/Stop Recording -->
  <dynaTrace.sessionNameProperty>dynaTrace.sessionName</dynaTrace.sessionNameProperty>

  <!-- Following is a list of properties for goal: startRecording -->
  <dynaTrace.sessionName>My Stored Session</dynaTrace.sessionName>
  <dynaTrace.sessionDescription>My stored Session
Description</dynaTrace.sessionDescription>
  <dynaTrace.recordingOption>all</dynaTrace.recordingOption> <!-- other options:
violations|timeseries -->
  <dynaTrace.sessionLocked>false</dynaTrace.sessionLocked>
  <dynaTrace.appendTimestamp>false</dynaTrace.appendTimestamp>
</properties>
```

Now we can call the startRecording goal in the following way:

```
mvn dynaTrace:dtAutomation:6.2:startRecording
```

You can inject the Dynatrace agent as part of surefire unit testing in Maven pom.xml with settings similar to this:

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.5</version>
<configuration>
<argLine>-agentpath:"C:\Program Files\dynaTrace\dynaTrace
6.2.0\agent\lib\dtagent.dll"=name=Maven,server=localhost:9998</argLine>
</configuration>
</plugin>
```

For Java Developers

The Automation Library includes the Dynatrace REST Automation SDK (com.dynatrace.diagnostics.automation.rest.sdk) which provides the helper class `RESTEndpoint` which exposes all important REST Interfaces of the Dynatrace Server.
Here is an example on how to use this helper class:

```
// Start and Stop Session Recording
RESTEndpoint endPoint = new RESTEndpoint("admin", "admin", "localhost");
String sessionName = endPoint.startRecording("easyTravel", "Session1", "My Load Test
Session", "all", true, false);
System.out.println("Started recording Session: "+ sessionName);
endPoint.stopRecording("easyTravel");

// list all connected agents
for(Agent agent : endPoint.getAgents()) {
  System.out.println("Agent Name: " + agent.getName());
}
```

Javadocs for the `RESTEndpoint` are available in the following archive: javadoc_RESTEndpoint_6.2.0.1238.zip.

DynatraceReporting

For a detailed description about the usage and samples see the documentation on Ant Task for dynaTrace 3.1 Reporting