Red Hat Virtualization 4.0

Java SDK Guide

Using the Red Hat Virtualization Java SDK

Last Updated: 2019-02-21

Red Hat Virtualization 4.0 Java SDK Guide

Using the Red Hat Virtualization Java SDK

Red Hat Virtualization Documentation Team
Red Hat Customer Content Services
rhev-docs@redhat.com

Legal Notice

Abstract

This guide describes how to install and work with version 3 and version 4 of the Red Hat Virtualization Java software development kit.

Table of Contents

CHAPTER 1. OVERVIEW **3**
1.1. PREREQUISITES 3
1.2. INSTALLING THE JAVA SOFTWARE DEVELOPMENT KIT 3
1.3. DEPENDENCIES 4
1.4. CONFIGURING SSL 4

CHAPTER 2. USING THE SOFTWARE DEVELOPMENT KIT **6**
2.1. CONNECTING TO THE RED HAT ENTERPRISE VIRTUALIZATION MANAGER IN VERSION 3 6
2.2. CONNECTING TO THE RED HAT VIRTUALIZATION MANAGER IN VERSION 4 7
2.3. LISTING ENTITIES 8
2.4. MODIFYING THE ATTRIBUTES OF RESOURCES 8
2.5. GETTING A RESOURCE 9
2.6. ADDING RESOURCES 9
2.7. PERFORMING ACTIONS ON RESOURCES 10
2.8. LISTING SUB-RESOURCES 10
2.9. GETTING SUB-RESOURCES 11
2.10. ADDING SUB-RESOURCES TO A RESOURCE 11
2.11. MODIFYING SUB-RESOURCES 11
2.12. PERFORMING ACTIONS ON SUB-RESOURCES 12

APPENDIX A. APIBUILDER METHODS **13**

APPENDIX B. CONNECTIONBUILDER METHODS **15**

CHAPTER 1. OVERVIEW

The Java software development kit is a collection of classes that allows you to interact with the Red Hat Virtualization Manager in Java-based projects. By downloading these classes and adding them to your project, you can access a range of functionality for high-level automation of administrative tasks.

Red Hat Virtualization provides two versions of the Java software development kit:

`Version 3`

The V3 Java software development kit provides backwards compatibility with the class and method structure provided in the Java software development kit as of the latest release of Red Hat Enterprise Virtualization 3.6. Applications written using the Java software development kit from Red Hat Enterprise Virtualization 3.6 can be used with this version without modification.

`Version 4`

The V4 Java software development kit provides an updated set of class and method names and signatures. Applications written using the Java software development kit from Red Hat Enterprise Virtualization 3.6 must be updated before they can be used with this version.

Either version of the Java software development kit can be used in a Red Hat Virtualization environment as required by installing the corresponding package and adding the required libraries to your Java project.

1.1. PREREQUISITES

To install the Java software development kit, you must have:

- A system where Red Hat Enterprise Linux 7 is installed. Both the Server and Workstation variants are supported.
- A subscription to Red Hat Virtualization entitlements.

IMPORTANT

The software development kit is an interface for the Red Hat Virtualization REST API. As such, you must use the version of the software development kit that corresponds to the version of your Red Hat Virtualization environment. For example, if you are using Red Hat Virtualization 4.0, you must use the version of the software development kit designed for 4.0.

1.2. INSTALLING THE JAVA SOFTWARE DEVELOPMENT KIT

Install the Java software development kit and accompanying documentation.

Procedure 1.1. Installing the Java Software Development Kit

1. Enable the required channels:

```
# subscription-manager repos --enable=rhel-7-server-rpms
# subscription-manager repos --enable=rhel-7-server-rhv-4.0-rpms
# subscription-manager repos --enable=jb-eap-7.0-for-rhel-7-server-
rpms
```

2. Install the required packages:

   - For V3:

     ```
     # yum install ovirt-engine-sdk-java ovirt-engine-sdk-javadoc
     ```

   - For V4:

     ```
     # yum install java-ovirt-engine-sdk4
     ```

The Java software development kit and accompanying documentation are downloaded to the **`/usr/share/java/rhevm-sdk-java`** directory, and can now be added to Java projects.

1.3. DEPENDENCIES

To use the Java software development kit in Java applications, you must add the following JAR files to the class path of those applications:

- **`commons-beanutils.jar`**
- **`commons-codec.jar`**
- **`httpclient.jar`**
- **`httpcore.jar`**
- **`jakarta-commons-logging.jar`**
- **`log4j.jar`**

The packages that provide these JAR files are installed as dependencies to the ovirt-engine-sdk-java package. By default, they are available in the **`/usr/share/java`** directory on Red Hat Enterprise Linux 6 and Red Hat Enterprise Linux 7 systems.

1.4. CONFIGURING SSL

The Red Hat Virtualization Manager Java SDK provides full support for HTTP over Secure Socket Layer (SSL) and the IETF Transport Layer Security (TLS) protocol using the Java Secure Socket Extension (JSSE). JSSE has been integrated into the Java 2 platform as of version 1.4 and works with the Java SDK out of the box. On older Java 2 versions, JSSE must be manually installed and configured.

1.4.1. Configuring SSL

The following procedure outlines how to configure SSL using the Java SDK.

Procedure 1.2. Configuring SSL

1. Download the certificate used by the Red Hat Virtualization Manager.

   NOTE

   By default, the location of the certificate used by the Red Hat Virtualization Manager is in **`/etc/pki/ovirt-engine/ca.pem`**.

2. Create a truststore:

```
$ keytool -import -alias "server.crt truststore" -file ca.crt -
keystore server.truststore
```

3. Specify the **`trustStoreFile`** and **`trustStorePassword`** arguments when constructing an instance of the **`Api`** or **`Connection`** object:

```
myBuilder.trustStoreFile("/home/username/server.truststore");
myBuilder.trustStorePassword("p@ssw0rd");
```

NOTE

If you do not specify the **`trustStoreFile`** option when creating a connection, the Java SDK attempts to use the default truststore specified by the system variable **`javax.net.ssl.trustStore`**. If this system variable does not specify a truststore, the Java SDK attempts to use a truststore specified in **`$JAVA_HOME/lib/security/jssecacerts`** or **`$JAVA_HOME/lib/security/cacerts`**.

1.4.2. Host Verification

By default, the identity of the host name in the certificate is verified when attempting to open a connection to the Red Hat Virtualization Manager. You can disable verification by passing the following argument when constructing an instance of the **`Connection`** class:

```
myBuilder.insecure(true);
```

IMPORTANT

This method should not be used for production systems due to security reasons, unless it is a conscious decision and you are aware of the security implications of not verifying host identity.

CHAPTER 2. USING THE SOFTWARE DEVELOPMENT KIT

This chapter outlines several examples of how to use the Java Software Development Kit. All examples in this chapter use version 3 of the software development kit unless otherwise noted.

2.1. CONNECTING TO THE RED HAT ENTERPRISE VIRTUALIZATION MANAGER IN VERSION 3

In V3 of the Java software development kit, the **`Api`** class is the main class you use to connect to and manipulate objects in a Red Hat Enterprise Virtualization environment. To declare an instance of this class, you must declare an instance of the **`ApiBuilder`** class, pass the required arguments to this instance using builder methods, then call the **`build`** method on the instance. The **`build`** method returns an instance of the **`Api`** class that you can then assign to a variable and use to perform subsequent actions.

The following is an example of a simple Java SE program that creates a connection with a Red Hat Enterprise Virtualization environment, then gracefully shuts down and closes the connection:

Example 2.1. Connecting to the Red Hat Enterprise Virtualization Manager

```
package rhevm;

import org.ovirt.engine.sdk.Api;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.ovirt.engine.sdk.ApiBuilder;
import org.ovirt.engine.sdk.exceptions.ServerException;
import org.ovirt.engine.sdk.exceptions.UnsecuredConnectionAttemptError;

public class rhevm {

    public static void main(String[] args) {

        Api api = null;

        try {

            ApiBuilder myBuilder = new ApiBuilder()

            .url("https://rhevm.example.com/api")
            .user("admin@internal")
            .password("p@ssw0rd")
            .keyStorePath("/home/username/server.truststore")
            .keyStorePassword("p@ssw0rd");

            api = myBuilder.build();

            api.shutdown();

        } catch (ServerException | UnsecuredConnectionAttemptError |
IOException ex) {
            Logger.getLogger(Ovirt.class.getName()).log(Level.SEVERE,
null, ex);
```

```
        } finally {
            if (api != null) {
                try {
                    api.close();
                } catch (Exception ex) {

Logger.getLogger(Ovirt.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    }
}
```

This example creates a connection using basic authentication, but other methods are also available. For a list of the key arguments that can be passed to instances of the **`ApiBuilder`** class, see Appendix A, *ApiBuilder Methods*.

NOTE

Note that the **`Api`** class does not implement the **`Autocloseable`** interface. As such, it is recommended that you shut down instances of the **`Api`** class in a **`finally`** block as per the above example to ensure the connection with the Red Hat Enterprise Virtualization Manager is closed gracefully.

2.2. CONNECTING TO THE RED HAT VIRTUALIZATION MANAGER IN VERSION 4

In V4 of the Java software development kit, the **`Connection`** class is the main class you use to connect to and manipulate objects in a Red Hat Virtualization environment. To declare an instance of this class, you must declare an instance of the **`ConnectionBuilder`** class, pass the required arguments to this instance using builder methods, then call the **`build`** method on the instance. The **`build`** method returns an instance of the **`Connection`** class that you can then assign to a variable and use to perform subsequent actions.

The following is an example of a simple Java SE program that creates a connection with a Red Hat Virtualization environment using version 4 of the software development kit:

Example 2.2. Connecting to the Red Hat Virtualization Manager

```
package rhevm;

import org.ovirt.engine.sdk4.Connection;
import org.ovirt.engine.sdk4.ConnectionBuilder;

public class rhevm {

    public static void main(String[] args) {

            ConnectionBuilder myBuilder =
ConnectionBuilder.connection()

            .url("https://rhevm.example.com/ovirt-engine/api")
```

```
                    .user("admin@internal")
                    .password("p@ssw0rd")
                    .trustStoreFile("/home/username/server.truststore")
                    .trustStorePassword("p@ssw0rd");

            try (Connection conn = myBuilder.build()) {

                // Requests

            } catch (Exception e) {

                // Error handling

            }
        }
}
```

This example creates a connection using basic authentication, but other methods are also available. For a list of the key arguments that can be passed to instances of the **ConnectionBuilder** class, see Appendix B, *ConnectionBuilder Methods*.

2.3. LISTING ENTITIES

The following example outlines how to list entities in the Red Hat Virtualization Manager. In this example, the entities to be listed are virtual machines, which are listed using the **getVMs()** method of the **Api** class.

Procedure 2.1. Listing Entities

- Declare a **List** of the type of entity to be listed and use the corresponding method to get the list of entities:

  ```
  List<VM> vms = api.getVMs().list();
  ```

2.4. MODIFYING THE ATTRIBUTES OF RESOURCES

The following example outlines how to modify the attributes of a resource. In this example, the attribute to be modified is the description of the virtual machine with the name 'test', which is changed to 'java_sdk'.

Procedure 2.2. Modifying the Attributes of a Resource

1. Declare an instance of the resource whose attributes are to be modified:

   ```
   VM vm = api.getVMs().get("test");
   ```

2. Set the new value of the attribute:

   ```
   vm.setDescription("java_sdk");
   ```

3. Update the virtual machine to apply the change:

```
VM newVM = vm.update();
```

2.5. GETTING A RESOURCE

In the Java Software Development Kit, resources can be referred to via two attributes: **name**, and **UUID**. Both return an object with the specified attribute if that object exists.

To get a resource using the value of the **name** attribute:

```
VM vm = api.getVMs().get("test");
```

To get a resource using the value of the **UUID** attribute:

```
VM vm = api.getVMs().get(UUID.fromString("5a89a1d2-32be-33f7-a0d1-
f8b5bc974ff6"));
```

2.6. ADDING RESOURCES

The following examples outline two ways to add resources to the Red Hat Virtualization Manager. In these examples, the resource to be added is a virtual machine.

Example 1

In this example, an instance of the **VM** class is declared to represent the new virtual machine to be added. Next, the attributes of that virtual machine set to the preferred values. Finally, the new virtual machine is added to the Manager.

```
org.ovirt.engine.sdk.entities.VM vmParams = new
org.ovirt.engine.sdk.entities.VM();

vmParams.setName("myVm");
vmParams.setCluster(api.getClusters().get("myCluster"));
vmParams.setTemplate(api.getTemplates().get("myTemplate"));
...
```

```
VM vm = api.getVMs().add(vmParams);
```

Example 2

In this example, an instance of the **VM** class is declared in the same way as Example 1. However, rather than using the **get** method to reference existing objects in the Manager, each attribute is referenced by declaring an instance of that attribute. Finally, the new virtual machine is added to the Manager.

```
org.ovirt.engine.sdk.entities.VM vmParams = new
org.ovirt.engine.sdk.entities.VM();

vmParams.setName("myVm");
org.ovirt.engine.sdk.entities.Cluster clusterParam = new Cluster();
clusterParam.setName("myCluster");
vmParams.setCluster(clusterParam);
org.ovirt.engine.sdk.entities.Template templateParam = new Template();
```

```
templateParam.setName("myTemplate");
vmParams.setTemplate(templateParam);
...
```

```
VM vm = api.getVMs().add(vmParams);
```

2.7. PERFORMING ACTIONS ON RESOURCES

The following example outlines how to perform actions on a resource. In this example, a virtual machine with the name 'test' is started.

Procedure 2.3. Performing an Action on a Resource

1. Declare an instance of the resource:

   ```
    VM vm = api.getVMs().get("test");
   ```

2. Declare action parameters to send to the resource:

   ```
   Action actionParam = new Action();
   org.ovirt.engine.sdk.entities.VM vmParam = new
   org.ovirt.engine.sdk.entities.VM();
   actionParam.setVm(vmParam);
   ```

3. Perform the action:

   ```
   Action res = vm.start(actionParam);
   ```

Alternatively, you can perform the action as an inner method:

```
Action res = vm.start(new Action()
{
    {
        setVm(new org.ovirt.engine.sdk.entities.VM());
    }
});
```

2.8. LISTING SUB-RESOURCES

The following example outlines how to list the sub-resources of a resource. In this example, the sub-resources of a virtual machine with the name 'test' are listed.

Procedure 2.4. Listing Sub-Resources

1. Declare an instance of the resource whose sub-resources are to be listed:

   ```
   VM vm = api.getVMs().get("test");
   ```

2. List the sub-resources:

   ```
   List<VMDisk> disks = vm.getDisks().list();
   ```

2.9. GETTING SUB-RESOURCES

The following example outlines how to reference the sub-resources of a resource. In this example, a disk with the name 'my disk' that belongs to a virtual machine with the name 'test' is referenced.

Procedure 2.5. Getting the Sub-Resources of a Resource

1. Declare an instance of the resource whose sub-resources are to be referenced:

   ```
   VM vm = api.getVMs().get("test");
   ```

2. Declare an instance of the sub-resource to be referenced:

   ```
   VMDisk disk = vm.getDisks().get("my disk");
   ```

2.10. ADDING SUB-RESOURCES TO A RESOURCE

The following example outlines how to add sub-resources to a resource. In this example, a new disk with a size of '1073741824L', interface 'virtio' and format 'cow' are added to a virtual machine with the name 'test'.

Procedure 2.6. Adding a Sub-Resource to a Resource

1. Declare an instance of the resource to which sub-resources are to be added:

   ```
   VM vm = api.getVMs().get("test");
   ```

2. Create parameters to define the attributes of the resource:

   ```
   Disk diskParam = new Disk();
   diskParam.setProvisionedSize(1073741824L);
   diskParam.setInterface("virtio");
   diskParam.setFormat("cow");
   ```

3. Add the sub-resource:

   ```
   Disk disk = vm.getDisks().add(diskParam);
   ```

2.11. MODIFYING SUB-RESOURCES

The following example outlines how to modify sub-resources. In this example, the name of a disk with the name 'test_Disk1' belonging to a virtual machine with the name 'test' is changed to 'test_Disk1_updated'.

Procedure 2.7. Updating a Sub-Resource

1. Declare an instance of the resource whose sub-resource is to be modified:

   ```
   VM vm = api.getVMs().get("test");
   ```

2. Declare an instance of the sub-resource to be modified:

```
VMDisk disk = vm.getDisks().get("test_Disk1");
```

3. Set the new value of the attribute:

```
disk.setAlias("test_Disk1_updated");
```

4. Update the sub-resource:

```
VMDisk updateDisk = disk.update();
```

2.12. PERFORMING ACTIONS ON SUB-RESOURCES

The following example outlines how to perform actions on sub-resources. In this example, a disk with the name 'test_Disk1' belonging to a virtual machine with the name 'test' is activated.

Procedure 2.8. Performing an Action on a Sub-Resource

1. Declare an instance of the resource containing the sub-resource on which the action is to be performed:

```
VM vm = api.getVMs().get("test");
```

2. Declare an instance of the sub-resource:

```
VMDisk disk = vm.getDisks().get("test_Disk1");
```

3. Declare action parameters to send to the sub-resource:

```
Action actionParam = new Action();
```

4. Perform the action:

```
Action result = disk.activate(actionParam);
```

APPENDIX A. APIBUILDER METHODS

The following table outlines the key methods available to the **`ApiBuilder`** class used in V3 of the Java software development kit.

Table A.1. ApiBuilder Methods

Method	Argument Type	Description
`user`	**`String`**	The name of the user with which to connect to the Manager. You must specify both the user name and domain, such as **`admin@internal`**. This method must be used together with the **`password`** method.
`password`	**`String`**	The password of the user with which to connect to the Manager.
`sessionID`	**`String`**	The identifier of a session with which to connect to the Manager. If you have already authenticated with the Manager and a session is available, you can specify this argument instead of specifying a user name and password.
`requestTimeout`	**`Integer`**	The timeout, in seconds, to wait for responses to requests. If a request takes longer than this value to respond, the request is cancelled, and an exception is thrown. This argument is optional.
`sessionTimeout`	**`Integer`**	The timeout, in minutes, after which an active session is destroyed if no requests are made to the Manager. This argument is optional.
`persistentAuth`	**`Boolean`**	Enables or disables persistent authentication using cookies. This option is enabled by default, so this method is only required to disable this option.
`noHostVerification`	**`Boolean`**	Enables or disables verification of the host name in the SSL certificate presented by the server where the Manager is hosted. By default, the identity of host names is verified, and the connection is rejected if the host name is not correct, so this method is only required to disable this option.
`keyStorePath`	**`String`**	Specifies the location of a file containing the CA certificate used to verify the certificate presented by the server where the Manager is hosted. This method must be used together with the **`keyStorePassword`** method.
`keyStorePassword`	**`String`**	The password used to access the keystore file specified in the **`keyStorePath`** method.

Method	Argument Type	Description
`filter`	`Boolean`	Enables or disables filtering of objects based on the permissions of the user making the request. By default, this option is disabled, which allows any user to see all objects in the environment. This method is only required to restrict the objects in the environment to those visible to the user making the request.
`debug`	`Boolean`	Enables or disables debug output. By default, this option is disabled.

APPENDIX B. CONNECTIONBUILDER METHODS

The following table outlines the key methods available to the **`ConnectionBuilder`** class used in V4 of the Java software development kit.

Table B.1. ConnectionBuilder Methods

Method	Argument Type	Description
`user`	**`String`**	The name of the user with which to connect to the Manager. You must specify both the user name and domain, such as **`admin@internal`**. This method must be used together with the **`password`** method.
`password`	**`String`**	The password of the user with which to connect to the Manager.
`compress`	**`Boolean`**	Specifies whether responses from the server where the Manager is hosted should be compressed. This option is disabled by default, so this method is only required to enable this option.
`timeout`	**`Integer`**	The timeout, in seconds, to wait for responses to requests. If a request takes longer than this value to respond, the request is cancelled, and an exception is thrown. This argument is optional.
`ssoUrl`	**`String`**	The base URL of the server where the Manager is hosted. For example, **`https://server.example.com/ovirt-engine/sso/oauth/token?\grant_type=password&scope=ovirt-app-api`** for password authentication.
`ssoRevokeUrl`	**`String`**	The base URL of the SSO revoke service. This option only needs to be specified when you use an external authentication service. By default, this URL is automatically calculated from the value of the **`url`** option so that SSO token revoke is performed using the SSO service that is part of the engine.
`ssoTokenName`	**`String`**	The token name in the JSON SSO response returned from the SSO server. By default, this value is **`access_token`**.
`insecure`	**`Boolean`**	Enables or disables verification of the host name in the SSL certificate presented by the server where the Manager is hosted. By default, the identity of host names is verified, and the connection is rejected if the host name is not correct, so this method is only required to disable this option.

Method	Argument Type	Description
`trustStoreFile`	`String`	Specifies the location of a file containing the CA certificate used to verify the certificate presented by the server where the Manager is hosted. This method must be used together with the `trustStorePassword` method.
`trustStorePassword`	`String`	The password used to access the keystore file specified in the `trustStorePath` method.