



Dialogic® PowerMedia™ XMS JSR 309 Connector Software Release 3.2

**Installation and Configuration Guide
with TeleStax Apache-Tomcat Application Server**

Copyright and Legal Notice

Copyright © 2016-2017 Dialogic Corporation. All Rights Reserved. You may not reproduce this document in whole or in part without permission in writing from Dialogic Corporation at the address provided below.

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation and its affiliates or subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in certain safety-affecting situations. Please see <http://www.dialogic.com/company/terms-of-use.aspx> for more details.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 3300 Boulevard de la Côte-Vertu, Suite 112, Montreal, Quebec, Canada H4R 1P8.

Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Dialogic, Dialogic Pro, Veraz, Brooktrout, Diva, BorderNet, PowerMedia, PowerVille, PowerNova, MSaaS, ControlSwitch, I-Gate, Cantata, TruFax, SwitchKit, Eiconcard, NMS Communications, SIPcontrol, Exnet, EXS, Vision, inCloud9, and NaturalAccess, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation and its affiliates or subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 3300 Boulevard de la Côte-Vertu, Suite 112, Montreal, Quebec, Canada H4R 1P8. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

The names of actual companies and products mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Table of Contents

1. Dialogic JSR 309 Connector Requirements.....	5
2. Contents of the Distribution	6
For Developers.....	7
3. Installation and Configuration	8
Preparing the J2EE Converged Application Server	8
Installing the Dialogic JSR 309 Connector	8
Configure the Application Server Platform	8
Deploying the Verification Application Using the Dialogic JSR 309 Connector	11
Configure the dlgc_sample_demo.properties File	11
Copy All JAVA Library (JAR) Files	12
Deploy the Dialogic JSR 309 Verification Application	13
Configuring the PowerMedia XMS Media File.....	16
Running the Dialogic JSR 309 Verification Application.....	18
Configure Platform SIP Routing Using SIP Servlets Management.....	18
Dial in to the Dialogic JSR 309 Verification Demo	20
4. Dialogic JSR 309 Verification Application	21
About.....	21
Details	21
Application WAR File Content	21
Application Initialization Steps.....	22
Application Steps to Initialize the Dialogic JSR 309 Connector	23
5. Troubleshooting	26
Logging.....	26
Dialogic JSR 309 Connector and Verification Application Troubleshooting	27
SIP Errors.....	28
6. Building and Debugging Sample Demos in Eclipse IDE.....	29
Prerequisites.....	29
Creating the Build Environment.....	29
Prepare the Eclipse Workspace	29
Configure the Application.....	33
Building the Project	49
Configuring Eclipse Project and TeleStax Application Server Deployed Application for Remote Debugging	50
Configuring the Application Server Platform for Remote Debugging.....	50
Eclipse Project Configuration for Remote Debugging	51
7. Appendix A: Dialogic JSR 309 Connector Environment Setup	54
Installing and Configuring the TeleStax Apache-Tomcat Application Server	54
Preinstallation Setup	54
Firewall Configuration.....	55
TeleStax Installation	56
TeleStax Configuration	56
TeleStax Startup	59
TeleStax Verification	59
8. Appendix B: Updating the Dialogic JSR 309 Connector.....	61

Revision History

Revision	Release Date	Notes
3.1	February 2017	Deploying the Verification Application Using the Dialogic JSR 309 Connector : Updated the Copy All JAVA Library (JAR) Files section.
3.0	December 2016	Updates for JSR 309 Release 3.2. Contents of the Distribution : Updated the section. Building and Debugging Sample Demos in Eclipse IDE : Updated the Prepare the Eclipse Workspace section.
2.0	July 2016	Updates for JSR 309 5.2 Service Update 1. Dialogic JSR 309 Connector Requirements : Updated the required platforms. Contents of the Distribution : Added the For Developers section. Configuring the Application Server Platform for Remote Debugging : Updated the script. Appendix A: Dialogic JSR 309 Connector Environment Setup : Updated the remote debugging port number.
1.0	March 2016	Initial release of this document.
Last modified: February 2017		

1. Dialogic JSR 309 Connector Requirements

The following requirements are needed before installing the Dialogic JSR 309 Connector:

- A functional TeleStax Apache-Tomcat Application Server platform for development and testing.

The Dialogic JSR 309 Connector has been tested with the following Apache-Tomcat versions of TeleStax Application Server.

- TeleStax Mobicents Apache-Tomcat AS:

Java 1.7 based:

mss-3.1.633-apache-tomcat-7.0.64

Java 1.8 based:

restcomm-sip-servlets-3.1.699-apache-tomcat-8.0.26

restcomm-sip-servlets-4.0.76-apache-tomcat-8.0.26

- TeleStax TelScale Apache-Tomcat AS:

Java 1.7 based:

TelScale-SIP-Servlets-7.0.3.329-apache-tomcat-7.0.64

Java 1.8 based:

TelScale-SIP-Servlets-7.0.3.329-apache-tomcat-8.0.26

Note: Refer to www.telestax.com for additional information about TeleStax Application Server and their licensing.

- A functional PowerMedia XMS Release 3.2 system.

Note: Refer to [Proper Configuration of PowerMedia XMS](#) for additional information.

- SIP phones or soft clients.

TeleStax Application Server Platform	Dialogic JSR 309 Connector
mss-3.1.633-apache-tomcat-7.0.64 TelScale-SIP-Servlets-7.0.3.329-apache-tomcat-7.0.64	dialogic309-5.x.xxxx-tomcat7.tar
restcomm-sip-servlets-3.1.699-apache-tomcat-8.0.26 restcomm-sip-servlets-4.0.76-apache-tomcat-8.0.26 TelScale-SIP-Servlets-7.0.3.329-apache-tomcat-8.0.26	dialogic309-5.x.xxxx-tomcat8.tar

2. Contents of the Distribution

This section lists and describes the files in the Dialogic JSR 309 Connector distribution.

There are number of platforms which are supported by the Dialogic JSR 309 Connector. Their distributions are based on supported platforms as well as version of Jave they support.

This document covers support for TeleStax open source (Mobicents) as well as equivalent comertial versions (TelScale) distributions which are based on Apache Tomcat.

- Java 1.7 based platform:
dialogic309-M.m.BBBB-tomcat7.tar
- Java 1.8 based platform:
dialogic309-M.m.BBBB-tomcat8.tar

Where:

- *M* defines a major version number.
- *N* defines a minor version number.
- *BBBB* defines the four-digit build number.

This package contains the following structure:

Dialogic JSR 309 Connector Files	Description
<u>DIR:</u> <i>/DlgcJSR309/application/</i> <u>CONTENTS:</u> <i>dlgc_sample_demo.war</i> <i>Dialogic.mp4</i> <i>DialogicSampleDemo/</i>	Directory that contains the Dialogic JSR 309 Verification Application <i>dlgc_sample_demo.war</i> ready to be deployed and the <i>Dialogic.mp4</i> media file used by the Verification Application (which will be part of upcoming PowerMedia XMS installs). Directory also contains the <i>DialogicSampleDemo</i> directory, which has all of the necessary items to build <i>dlgc_sample_demo.war</i> . Refer to Dialogic JSR 309 Verification Application for details.
<u>DIR:</u> <i>/DlgcJSR309/Dlgc309Connector/</i> <i>/DlgcJSR309/3rdPartyLibs/</i>	Directory that contains the <i>Dlgc309Connector</i> , which has all of the Dialogic connector files, and the <i>3rdPartyLibs</i> directory, which has all necessary third-party JAR files.

Dialogic JSR 309 Connector Files	Description
<u>DIR:</u> <i>/DlgcJSR309/properties/</i> <u>CONTENTS:</u> <i>dlgc_sample_demo.properties</i> <i>log4j2.xml</i>	<p>Directory that contains Verification Application properties files used to set up its configuration and the configuration parameters for the Dialogic JSR 309 Connector.</p> <p>Directory also contains the <i>log4j2.xml</i> log configuration file used for Dialogic JSR 309 Connector and Verification Application logging.</p>

For Developers

To make it easier for developers, Dialogic provides connector files for direct download from an HTTPS URL:

- <https://www.dialogic.com/files/jsr-309/3.2GA/dialogic309-3.2-GA-14621-tomcat7.jar>
- <https://www.dialogic.com/files/jsr-309/3.2GA/dialogic309-3.2-GA-14621-tomcat8.jar>
- <https://www.dialogic.com/files/jsr-309/3.2GA/dialogicmsmltypes-3.2-GA-14621.jar>
- <https://www.dialogic.com/files/jsr-309/3.2GA/dialogicsmiltypes-3.2-GA-14621.jar>

Note: Make sure you choose the appropriate JDK build connector.

3. Installation and Configuration

This section describes how to install and use the Dialogic JSR 309 Connector. The Dialogic JSR 309 Connector adds the Media Control API interface to an application running in a J2EE platform. The connector and the application need to be correctly configured on a platform for proper operation.

The following steps are necessary to configure and run the Dialogic JSR 309 Connector:

1. [Preparing the J2EE Converged Application Server](#)
2. [Installing the Dialogic JSR 309 Connector](#)
3. [Deploying the Verification Application Using the Dialogic JSR 309 Connector](#)
4. [Configuring the PowerMedia XMS Media File](#)
5. [Running the Dialogic JSR 309 Verification Application](#)

For system requirements and supported platforms, see [Dialogic JSR 309 Connector Requirements](#).

Preparing the J2EE Converged Application Server

The Dialogic JSR 309 Connector has been deployed and tested on specific versions of TeleStax Apache-Tomcat Application Servers. For quick instructions on how to install and configure the desired Application Server (AS) to use with the Dialogic JSR 309 Connector, refer to [Appendix A: Dialogic JSR 309 Connector Environment Setup](#).

Installing the Dialogic JSR 309 Connector

The Dialogic JSR 309 Connector is created as a library (JAR file) that can be used by an application. The application is responsible for packaging it as part of the application WAR file.

The Dialogic JSR 309 Connector Verification Application is used to verify Dialogic JSR 309 Connector installation and configuration and to illustrate the necessary steps used in the Dialogic Media Server Control API features. These necessary application level steps are clearly described in [Dialogic JSR 309 Verification Application](#).

Now let's focus on the necessary steps correctly configure the Dialogic JSR 309 Connector and how to verify its proper operation.

The distribution package needs to be extracted onto the target system because various components (files) will be needed to correctly complete each step. Refer to [Contents of the Distribution](#), which describes the content in detail.

Configure the Application Server Platform

Place the package TAR file on TeleStax Apache-Tomcat Linux server and run the following command:

```
tar -xvf Dlgc309-5.x.xxxx-tomcatX.tar
```

This will create a *DlgcJSR309* directory, which includes all necessary files as described in [Contents of the Distribution](#).

Note: These directories are referenced throughout this document for content required by the Dialogic JSR 309 Connector.

In order to properly configure Application Server platform the following steps need to be completed:

1. [Set Up the Environment Variables](#)
2. [Set Up and Configure Logging Facility](#)

Set Up the Environment Variables

In addition to the required platform environments, the location of the Dialogic Verification Application properties file needs to be specified. The following procedure is an example of using `.bashrc` to accomplish this.

1. Edit system user's `.bashrc` file using the following command.

```
vi .bashrc
```

2. Add the following lines marked in RED below. In this example, the *mss-4.0.21-apache-tomcat-8.0.26* Application Server platform is used and the user "apachetomcat8" was created on the Linux system. Therefore, the file to be edited will be found in this user's root directory (for example, */home/apachetomcat8*).

```
# .bashrc

### Dialogic additions

export JAVA_HOME=/usr/java/jdk1.8.0_60
export CATALINA_HOME=/home/apachetomcat8/mss-4.0.21-apache-tomcat-8.0.26
export SAMPLE_PROPERTY_FILE=${CATALINA_HOME}/conf/Dialogic/dlqc_sample_demo.properties

### END - Dialogic additions

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions
```

Note the following about Dialogic additions:

- `SAMPLE_PROPERTY_FILE` points to the configuration properties file used by the Dialogic JSR 309 verification demo.

Note: This will not be needed if a custom application is being deployed.

In order for the above changes to take effect, a user either needs to log out and log back in or execute the "source" command for the modified `.bashrc` file:

```
source /home/apachetomcat8/.bashrc
```

Set Up and Configure Logging Facility

The Log4j2 logging facility is implemented through five third-party Log4j2 library files (JAR files) that need to be placed in the appropriate platform lib directory. From the Dialogic JSR 309 Connector distribution, copy the following five JAR files (located in *3rdPartyLibs* directory):

- *log4j-api-2.2.jar*
- *log4j-core-2.2.jar*
- *log4j-slf4j-impl-2.2.jar*
- *slf4j-api-1.7.5.jar*
- *org.osgi-3.0.0.jar*

Place the five JAR files in the platform specific *lib* directory:

```
${CATALINA_HOME}/lib
```

Now, edit *catalina.sh* file located in located in:

```
${CATALINA_HOME}/bin/catalina.sh
```

Add the following lines exactly located as shown below:

```
# Uncomment the following line to make the umask available when using the
# org.apache.catalina.security.SecurityListener
#JAVA_OPTS="$JAVA_OPTS -Dorg.apache.catalina.security.SecurityListener.UMASK=`umask`"

### Dialogic additions
JAVA_OPTS="$JAVA_OPTS -Dlog4j.configurationFile=${CATALINA_HOME}/conf/Dialogic/log4j2.xml"
### END - Dialogic additions

# ----- Execute The Requested Command -----
```

From the distribution directory, copy *log4j2.xml* into the following platform directory:

```
${CATALINA_HOME}/conf/Dialogic/log4j2.xml"
```

Note: A "Dialogic" directory will need to be created if it does not already exist.

Note the following about the logging facility:

- Due to a known Log4j version 1 Thread Deadlock issue, the Dialogic JSR 309 Connector has been built using Log4j2 (version 2). Modification of a platform startup script is required to configure the Log4j2 logging facility and to define a reference to its configuration .xml file.
- The *Dialogic.log* file, when generated, is found here:

```
${CATALINA_HOME}/logs/Dialogic.log"
```
- Default logging configuration is set to ERROR. Configuration file, *log4j2.xml*, can be edited if one wishes to change the logging levels.

Note: The *log4j2.xml* file changes go into effect automatically as governed by the configuration parameter in the *log4j2.xml* file. Details of *log4j2.xml* as provided can be found in the Troubleshooting section under Logging.

Deploying the Verification Application Using the Dialogic JSR 309 Connector

The Dialogic JSR 309 Verification Application, like any other application that wants to use J2EE Media Server Control (JSR 309) API, needs to do the following:

1. Take specific steps to correctly initialize itself for the J2EE platform to correctly deploy it.
2. Take specific steps to correctly initialize the Dialogic JSR 309 Connector for its use.

Note: For further details on the application architecture refer to [Dialogic JSR 309 Verification Application](#).

The following steps are necessary to deploy the Dialogic JSR 309 Verification Application:

- Configure *dlgc_sample_demo.properties* file
- Copy all JAVA library (JAR) files
- Deploy the Dialogic JSR 309 Verification Application
- Run the Dialogic JSR 309 Verification Application

Configure the *dlgc_sample_demo.properties* File

From the distribution directory, copy *dlgc_sample_demo.properties* file and place it in the following directory:

```
${CATALINA_HOME}/conf/Dialogic"
```

Note: The "Dialogic" directory should already exist and contain the *log4j2.xml* file.

From the platform's Dialogic directory, edit the *dlgc_sample_demo.properties* file. Provide the required parameters as illustrated in **RED** below:

```
# Dialogic JSR 309 Verification Application configuration parameters:
# Dialogic JSR 309 Connector Configuration
    dlgc.jsr309.driver.name=com.dialogic.dlg309
    connector.sip.address=xxx.xxx.xxx.xxx
    connector.sip.port=xxxx
    connector.sip.transport=udp
# Dialogic Media Server Configuration
    mediaserver.sessionTimer.switch=off
    mediaserver.sessionTimer.maxTimeout=120
    mediaserver.sessionTimer.minTimeout=100
    mediaserver.sip.address=xxx.xxx.xxx.xxx
    mediaserver.sip.port=xxxx

# Application runtime parameters:
play.prompt=file:///en_US/verification/Dialogic.mp4
```

Refer to the following information for details on the *dlgc_sample_demo.properties* file:

1. Dialogic JSR 309 Connector Configuration:

- `connector.sip.address` – IP address of the SIP interface used by the platform. The connector detects an IP address automatically. However, if there is more than one interface available on the platform, the application will have to be able to set the appropriate IP address for the connector to use. The demo will take this IP address and if different from the automatically discovered IP address, the demo will use it.
- `connector.sip.port` – Port address of the SIP interface used by the platform. The connector detects a port automatically, which goes with detected IP address. However, if the port address is incorrect, the application will have to set the appropriate one for connector to use. The demo will take this port address, and if different from the automatically discovered port address, the demo will use it.

2. Dialogic Media Server Configuration:

- `mediaserver.sip.address` – User needs to specify an IP of a Dialogic PowerMedia XMS to be used by the Dialogic JSR 309 Connector.
- `mediaserver.sip.port` – User needs to specify a port of a Dialogic PowerMedia XMS to be used by the Dialogic JSR 309 Connector.

Copy All JAVA Library (JAR) Files

The Dialogic JSR 309 Connector and its Verification Application are dependent on a few third-party JAVA library files.

Some library JAR files need to be part of the application itself and others need to be copied into platform's *lib* directory. Here is a view of the directory structure of the Dialogic JSR 309 Verification Application and highlighted is the *lib* directory where all required JAR files need to be placed:

```
\DemoGUI\index.html
\DemoGUI\webrtc.js
\META-INF\MANIFEST.MF
\WEB-INF\web.xml
\WEB-INF\sip.xml
\WEB-INF\classes\base\ConfigProperty.class
\WEB-INF\classes\base\WebClientPing$1.class
\WEB-INF\classes\base\WebClientPing.class
\WEB-INF\classes\play\AsyncPlayer$PlayerEventListener.class
\WEB-INF\classes\play\AsyncPlayer$PlayerJoinEventListener.class
\WEB-INF\classes\play\AsyncPlayer$SdpPortsListener.class
\WEB-INF\classes\play\AsyncPlayer.class
\WEB-INF\lib\Dialogic309-5.#.###-tomcat#.jar
\WEB-INF\lib\Dialogic309msmltypes-5.#.###-tomcat#.jar
```

The rest of required JAR files need to be placed in the platform's *lib* directory:

```
${CATALINA_HOME}/lib
```

Below is the list of JAR files that need to be copied:

- *dialogic309smiltypes-5.0.0.0-tomcat#.jar*
- *geronimo-commonj_1.1_spec-1.0.jar*
- *json_simple-1.1.jar*
- *jsr173_1.0_api.jar*
- *log4j-api-2.2.jar*
- *log4j-core-2.2.jar*
- *log4j-slf4j-impl-2.2.jar*
- *mscontrol.jar*
- *org.osgi-3.0.0.jar*
- *slf4j-api-1.7.5.jar*
- *xbean.jar*

Note: Further details on application WAR file content, refer to [Dialogic JSR 309 Verification Application](#).

Deploy the Dialogic JSR 309 Verification Application

The Dialogic JSR 309 Connector demo application needs to be deployed in the TeleStax Application Server.

There are two ways to deploy an application WAR file in TeleStax Application Server.

- [Deploying via Application Server Web Administration Console](#)
- [Deploying Directly to the Server](#)

Deploying via Application Server Web Administration Console

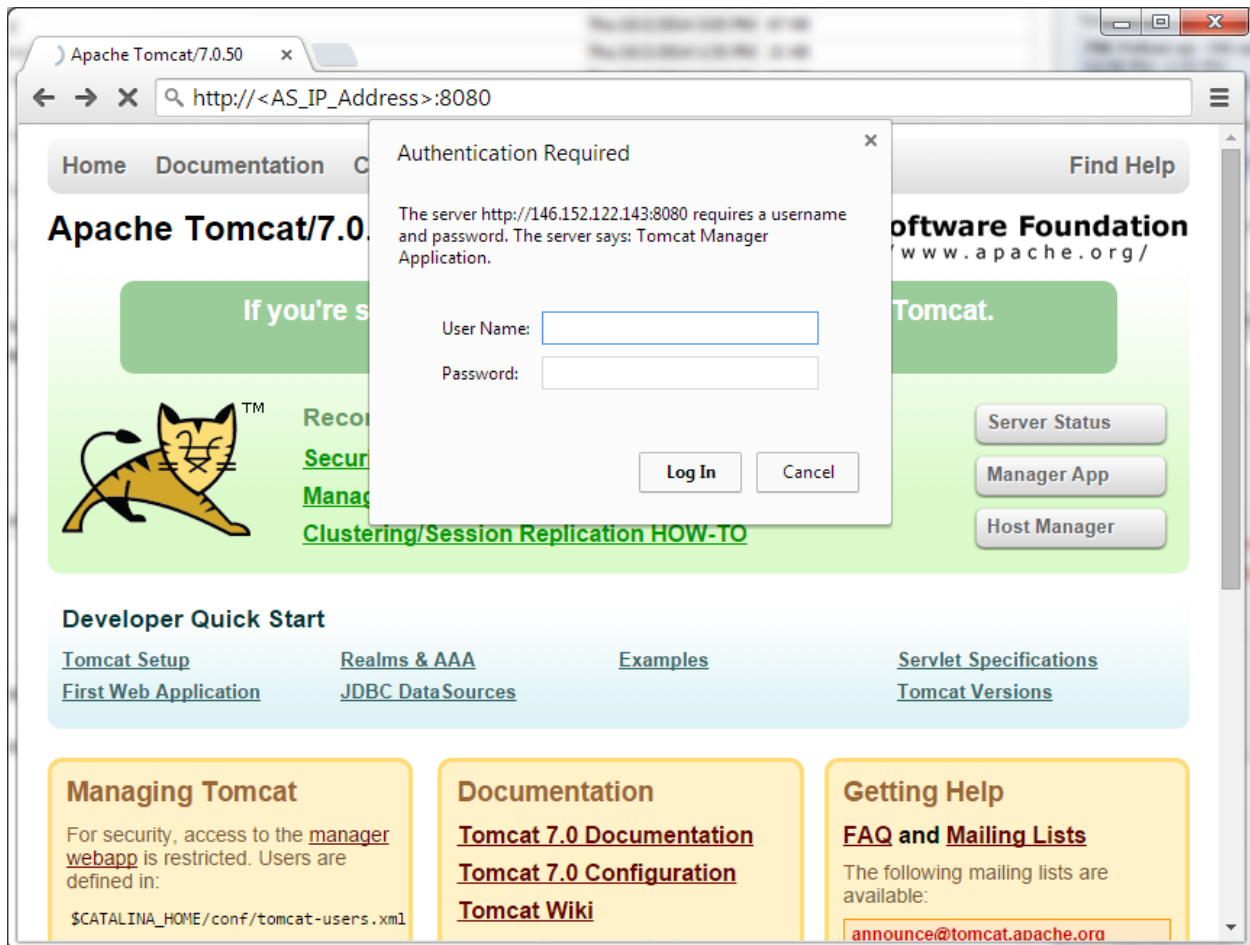
1. Start TeleStax Application Server from:

```
${CATALINA_HOME}/bin
```

2. Execute the following script. If the server starts without any errors/exceptions, the application is ready to be deployed.

```
./catalina.sh run
```

3. Navigate to the Administration Console, enter the appropriate login credentials, and then click **Manager App** on the top right of the window.



4. Scroll down to the **Deploy** section.

Deploy	
Deploy directory or WAR file located on server	
Context Path (required):	<input type="text"/>
XML Configuration file URL:	<input type="text"/>
WAR or Directory URL:	<input type="text"/>
<input type="button" value="Deploy"/>	
WAR file to deploy	
Select WAR file to upload	<input type="button" value="Choose File"/> No file chosen
<input type="button" value="Deploy"/>	

- Under the **WAR file to deploy** section, click **Choose File**, select the *dlgmsc_tests.war* file in the *TeleStaxTomcat-msc#.#/DlgcJSR309/applications* directory, and click **Deploy**.

[Deploy](#)

WAR file to deploy

Select WAR file to upload [Choose File](#) *dlgmsc_tests.war*

[Deploy](#)

Once successfully deployed, the application */dlgmsc_tests* will appear under the **Applications** section at the top of the page.

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	<div>Start Stop Reload Undeploy</div> <div>Expire sessions with idle ≥ 30 minutes</div>
/Click2CallAsync	None specified	ClickToCallAsyncApplication	true	0	<div>Start Stop Reload Undeploy</div> <div>Expire sessions with idle ≥ 30 minutes</div>
/dlgmsc_tests	None specified	Dialogic-Samples	true	0	<div>Start Stop Reload Undeploy</div> <div>Expire sessions with idle ≥ 30 minutes</div>
					Start Stop Reload Undeploy

The **Console** window will also indicate the successful deployment.

```

2014-04-02 20:47:04,826 WARN [DlgsXMsControlFactory] (http-bio-8080-exec-11) Assuming TCK is not enabled
2014-04-02 20:47:04,933 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11)
org.mobicens.servlet.sip.core.SipApplicationDispatcherImpl@4eed6301 the following sip servlet application has been added : Dialogic-
Samples
2014-04-02 20:47:04,933 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) It contains the following Sip Servlets :
2014-04-02 20:47:04,933 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : DlgcPlayerPerformanceTest
2014-04-02 20:47:04,933 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : DlgcRecorderTest
2014-04-02 20:47:04,933 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : DlgcMsMonitorServlet
2014-04-02 20:47:04,934 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : DlgcReferenceConferenceWithoutBCallServlet
2014-04-02 20:47:04,934 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : DlgcDtmfPromptCollectTest
2014-04-02 20:47:04,934 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : JMCConferenceServlet
2014-04-02 20:47:04,934 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : DlgcBridgeServlet
2014-04-02 20:47:04,934 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : DlgcDtmfAsyncTest
2014-04-02 20:47:04,934 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-
Samples/ServletName : DlgcEMBridgeServlet
2014-04-02 20:47:04,934 INFO [SipApplicationDispatcherImpl] (http-bio-8080-exec-11) SipApplicationName : Dialogic-

```

Deploying Directly to the Server

Another way to deploy the application is to place the WAR file directly in the following directory:

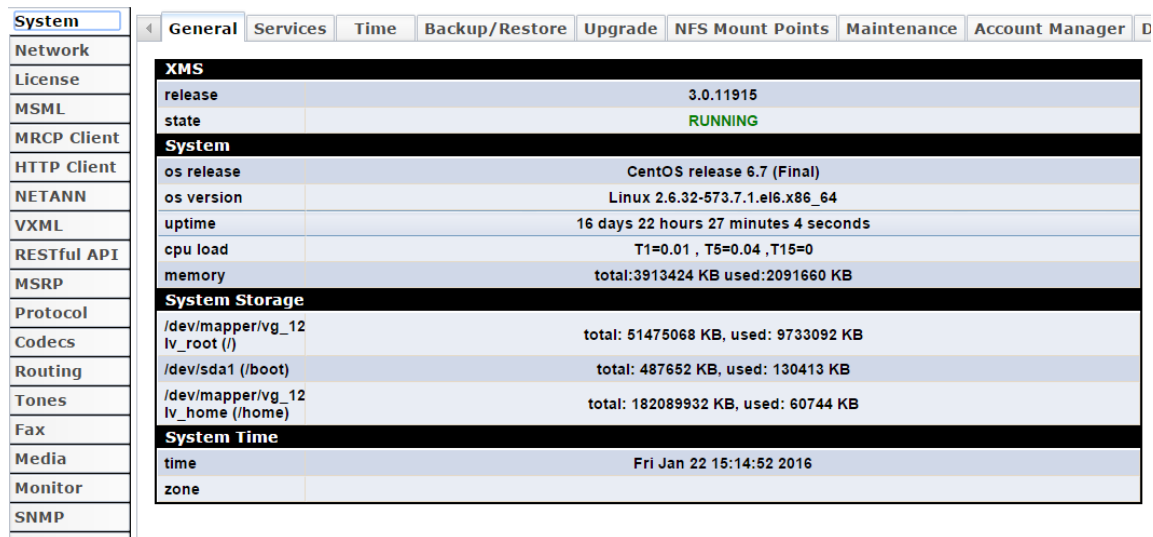
```
${CATALINA_HOME}/webapps
```

The Application Server is monitoring the deployments directory. If a new or updated WAR file is detected, it will attempt to start automatically.

Configuring the PowerMedia XMS Media File

The Dialogic JSR 309 Verification Application has been developed to use the *Dialogic.mp4* media file. This media file will become part of Dialogic PowerMedia XMS distribution; however, as of PowerMedia XMS release 3.0 Service Update 1, the media file is not part of distribution and must be installed manually.

1. To install *Dialogic.mp4* manually, log in to PowerMedia XMS WebGUI, and then click **Media**.



The screenshot shows the PowerMedia XMS WebGUI interface with the 'System' tab selected. The left sidebar contains a list of system components: System, Network, License, MSML, MRCP Client, HTTP Client, NETANN, VXML, RESTful API, MSRP, Protocol, Codecs, Routing, Tones, Fax, Media, Monitor, and SNMP. The main content area displays system information organized into sections: XMS, System, System Storage, and System Time.

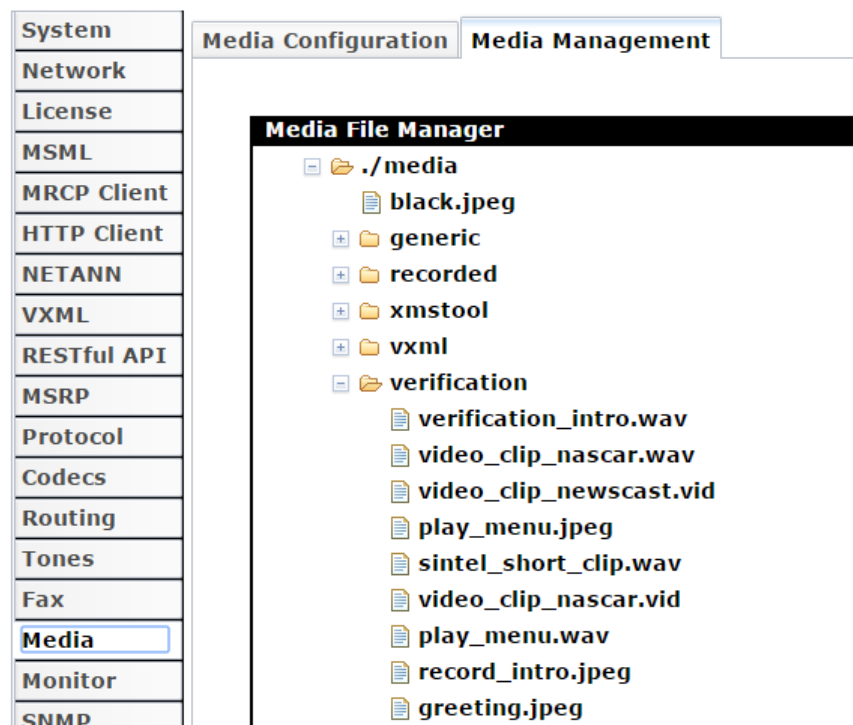
XMS	
release	3.0.11915
state	RUNNING

System	
os release	CentOS release 6.7 (Final)
os version	Linux 2.6.32-573.7.1.el6.x86_64
uptime	16 days 22 hours 27 minutes 4 seconds
cpu load	T1=0.01 , T5=0.04 , T15=0
memory	total:3913424 KB used:2091660 KB

System Storage	
/dev/mapper/vg_12 lv_root (/)	total: 51475068 KB, used: 9733092 KB
/dev/sda1 (/boot)	total: 487652 KB, used: 130413 KB
/dev/mapper/vg_12 lv_home (/home)	total: 182089932 KB, used: 60744 KB

System Time	
time	Fri Jan 22 15:14:52 2016
zone	

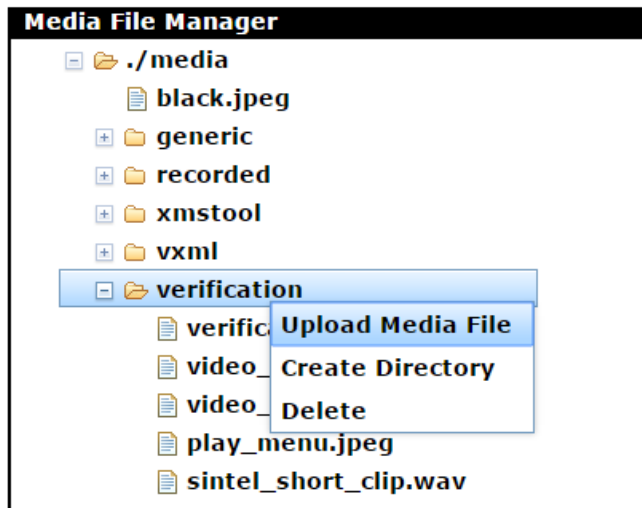
2. Click the **Media Management** tab.



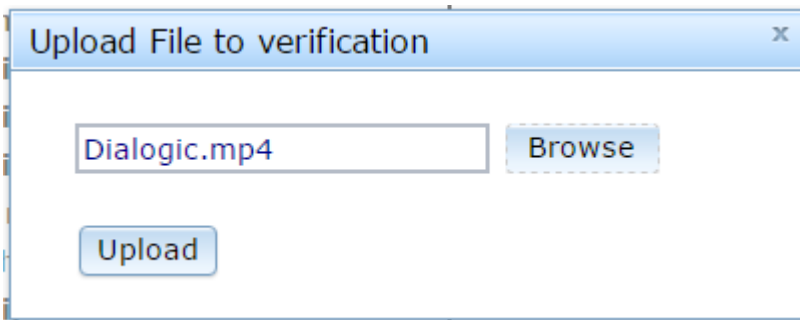
The screenshot shows the PowerMedia XMS WebGUI interface with the 'Media Management' tab selected. The left sidebar is the same as in the previous screenshot, but the 'Media' component is highlighted. The main content area displays the 'Media File Manager' interface, showing a directory tree for the media files.

```
./media
├── black.jpeg
├── generic
├── recorded
├── xmstool
├── vxml
└── verification
    ├── verification_intro.wav
    ├── video_clip_nascar.wav
    ├── video_clip_newscast.vid
    ├── play_menu.jpeg
    ├── sintel_short_clip.wav
    ├── video_clip_nascar.vid
    ├── play_menu.wav
    ├── record_intro.jpeg
    └── greeting.jpeg
```

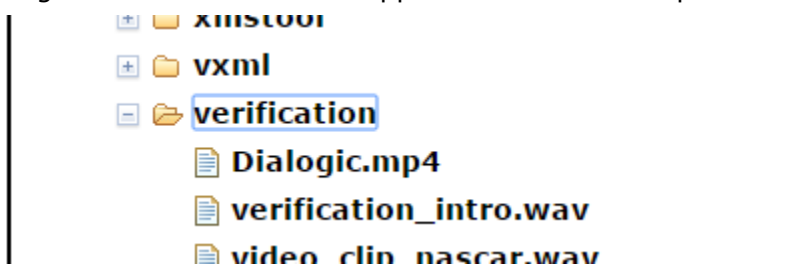

3. In the **Media File Manager** section, Right-click the **verification** directory and click **Upload Media File**.



4. Click **Browse**, select *Dialogic.mp4* in the Dialogic JSR 309 Connector distribution directory, and click **Upload**.



Once uploaded, the *Dialogic.mp4* media file is stored in the appropriate location for the Dialogic JSR 309 Verification Application to use it as per its configuration.



Running the Dialogic JSR 309 Verification Application

The Verification Application listens for incoming SIP and WebRTC calls and plays a *Dialogic.mp4* audio/video file. Based on the type of connection with the client (audio only, audio/video, or video only), it will play the appropriate media type to the client. For example, if the client supports an audio only connection, then only the audio portion of the sample .mp4 file will be played.

Follow these steps:

- [Configure Platform SIP Routing Using SIP Servlets Management](#)
- [Dial in to the Dialogic JSR 309 Verification Demo](#)

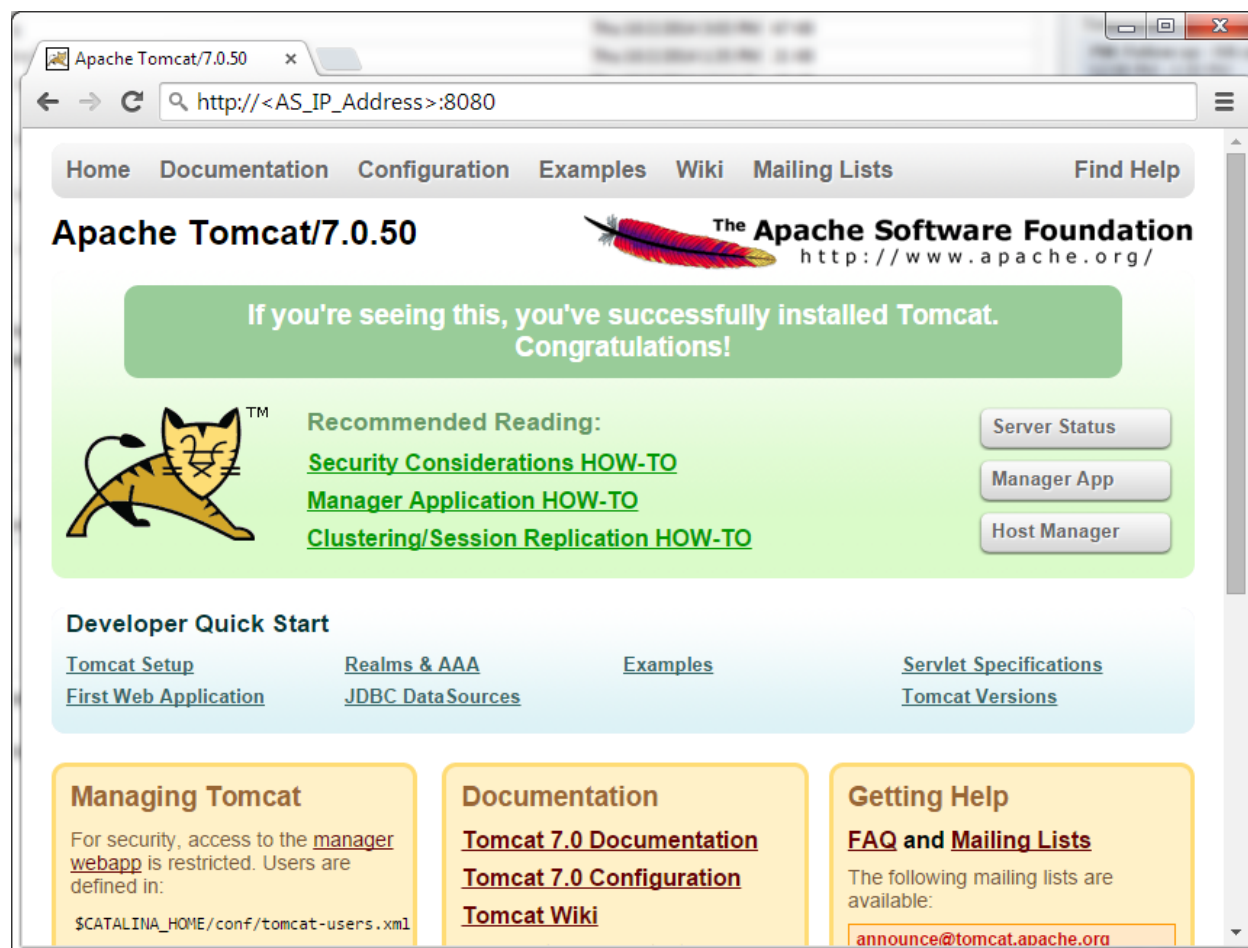
Configure Platform SIP Routing Using SIP Servlets Management

Before the deployed application can process SIP messages, it needs to be configured in the SIP Servlets Management console:

`http://<AS_IP_Address>:8080/`

Log in with the credentials set up in *tomcat-users.xml*. Refer to [TeleStax Verification](#) for details, and then click **Manager App** on the top right of the window.

Note: You might be asked to enter your login credentials again.



1. Click **sip-servlets-management**.

Context Path	Wrapper	Application Name	Enabled	Max Threads	Actions
/host-manager	None specified	Tomcat Host Manager Application	true	0	Undeploy Expire sessions with idle ≥ 30 minutes
/jolokia	None specified	JSON JMX Agent	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	2	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/sip-servlets-management	None specified	Mobicents SIP Servlets Management	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/websockets-sip-servlet	None specified	WebsocketSample	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy
Deploy directory or WAR file located on server

2. In the **SIP Servlets Management Console**, click the **INVITE** tab and select **dlgc_sample_demo** in the **SIP Application Name** field.

ALL REGISTER INVITE OPTIONS MESSAGE SUBSCRIBE NOTIFY

SIP Application Name : + Add Application

dlgc_sample_demo ▼

Subscriber Identity :

DAR:From

3. Click **Apply Changes** button. The following message on the bottom of the screen will confirm success if the Application Server is configured to use the newly deployed application.

[INFO] DAR Information successfully updated

Dial in to the Dialogic JSR 309 Verification Demo

The Dialogic JSR 309 Verification Demo (player) has been written to support both SIP and WebRTC (Chrome and Firefox only at this time) clients.

Dial in to a SIP client as follows:

1. Have a SIP client configured for the supported audio codec.
2. Place a call into the Application Server with the following URI:

```
player@<as_ip_address>:5080
```

With successful configuration, the sample verification .mp4 should be heard and/or seen.

Dial in to a WebRTC client as follows:

Note: With the newest versions of browsers, WebRTC media functionality and WebSocket support are only allowed via a secure web connection (HTTPS). The J2EE platform will need to be configured for HTTPS in order for a WebRTC Client to work. Follow the online documentation of the platform for HTTPS configuration instructions.

1. Open a Chrome or Firefox web browser.
2. Navigate to the following URL:

```
https://<AS_Platform_IP>:8443/dlgc_sample_demo/DemoGUI/index.html
```

3. Click **Play**.

4. Dialogic JSR 309 Verification Application

About

The Dialogic JSR 309 Verification Application is provided with each platform specific package for two reasons:

1. The application (WAR file), which uses the Dialogic JSR 309 Connector, is provided as a tool to verify the Application Server platform and Dialogic PowerMedia XMS operation.
2. The application project source has all the necessary components required to create a platform-specific application using the Dialogic JSR 309 Connector. This can quickly help clarify various steps that are required in the J2EE application using the Dialogic JSR 309 Connector. It includes the following:
 - a. Provides steps on how to create an application (WAR file) to run in a specific J2EE AS platform
 - b. Illustrates application initialization steps
 - c. Illustrates application initialization steps necessary for use with the Dialogic JSR 309 Connector
 - d. Illustrates the steps the application needs to take in order to work with SIP and/or web based multimedia (WebRTC) clients (provided that the chosen platform provides support for server side WebSockets)

Details

This section details the different areas of the Verification Application for a better understanding of the basic, necessary steps for any application.

- [Application WAR File Content](#)
- [Application Initialization Steps](#)
- [Application Steps to Initialize the Dialogic JSR 309 Connector](#)

Application WAR File Content

Minimum content of the application is illustrated in the Dialogic JSR 309 Verification Application WAR file. The WAR package contains several necessary items. Please refer to *build.xml* to get familiar with how the WAR file is generated.

The *dlgc_sample_demo.war* file consists of three directories:

- The */META-INF* directory contains a *MANIFEST.MF*, which is a standard way of providing information about the package that contains it.
- The */WEB-INF* directory contains the following:
 - The *classes* directory, which contains JAVA .class files.
 - The *lib* directory, which contains all JAR files required by the deployment application WAR file.
 - The *jboss-deployment-structure.xml* file used to exclude some automatic dependencies.
 - The *sip.xml* file, which defines SIP servlet-mapping for the deployment application WAR.

- The */DemoGUI* directory contains content for the application that supports WebRTC. It includes necessary .html and .js files for the verification demo.

Application Initialization Steps

Below is an example of a basic application structure used in this platform. For further details please reference a source of Dialogic JSR 309 Verification Application.

```
package play;

@javax.servlet.sip.annotation.SipServlet(description = "Dialogic Sample Demo", applicationName =
"DialogicSampleDemo", name="AsyncPlayer", loadOnStartup=2)

@SipListener
public class AsyncPlayer extends SipServlet implements Serializable, SipServletListener
{
    @Override
    public void init(ServletConfig cfg) throws ServletException
    {
    }

    @Override
    public void servletInitialized(SipServletContextEvent evt)
    {
    }
}
```

Note the following:

1. Note the mandatory *@javax.servlet.sip.annotation.SipServlet* annotation. This annotation allows for the Sip Servlet metadata to be declared without having to create the deployment descriptor:
 - a. description – Any string describing the application.
 - b. applicationName – Must match servlet mapping as defined in *sip.xml* for this application. The Dialogic JSR 309 Connector Verification Application *sip.xml* is shown below for reference:

```
<?xml version="1.0" encoding="UTF-8"?>
<sip-app>

    <app-name>DialogicSampleDemo</app-name>
    <display-name>DialogicSampleDemo</display-name>

    <session-config>
        <session-timeout>0</session-timeout>
    </session-config>

    <servlet-selection>
        <servlet-mapping>
            <servlet-name>AsyncPlayer</servlet-name>
            <pattern>
                <and>
```

```

        <equal>
            <var>request.method</var>
            <value>INVITE</value>
        </equal>
        <equal>
            <var>request.to.uri.user</var>
            <value>player</value>
        </equal>
    </and>
</pattern>
</servlet-mapping>
</servlet-selection>

</sip-app>

```

- c. name – The name of the SIP servlet class:

```

@SipListener
public class AsyncPlayer extends SipServlet implements Serializable,
SipServletListener
{

```

- d. loadOnStartup – Defines the order in which the SIP container should start this SIP Servlet class. Since this application depends on JSR 309, it should start after the Dialogic JSR 309 Connector. Since Dialogic JSR 309 Connector Sip Servlet annotation is set (loadOnStartup) to 1, the application should be a number greater than 1.
2. When the platform starts the application, it will invoke an init() function. This function should contain application specific initialization procedures. This is where the Verification Application reads the application properties file and stores its content in local storage to be used later when initializing the JSR 309 interface.
 3. Once the platform's SIP container is started, it will call the application's servletInitialized() method to inform it that the SIP stack is now ready for application usage. At this stage, the application can start to initialize the Dialogic JSR 309 Connector.

Application Steps to Initialize the Dialogic JSR 309 Connector

Once the application's servletInitialized() method is invoked, the SIP container has been initialized and the application can now take steps to initialize the JSR 309 interface. After validating the request in the Verification Application servletInitialized() method, the application will issue initDriver() method.

The application obtains the Dialogic JSR 309 Connector configuration:

```

protected boolean initDriver()
{
    dlgcDriver = DriverManager.getDriver(DLGC_309_DRIVER_NAME);
    PropertyInfo connectorProperty[] = dlgcDriver.getFactoryPropertyInfo();
    ...

```

Connector driver is able to discover some of the parameters that it needs but not all. The parameters required by the driver to work correctly are:

- `connector.sip.address` – Platform SIP IP address used by the SIP container. The connector provides the ability to change the address in case the platform has multiple IP interfaces and the default IP address picked by connector needs to be changed.
- `connector.sip.port` – Platform SIP port address used by SIP container. The connector provides ability to change the address in case the platform has multiple IP interfaces and the proper one is defined for different port number.
- `connector.sip.transport` – Platform SIP transport. Supported values are "udp" or "tcp". Default: "udp".
- `mediaserver.sip.ipaddress` – Dialogic XMS Media Server SIP IP address to be used by the Dialogic JSR 309 Connector.
- `mediaserver.sip.port` – Dialogic XMS Media Server SIP port to be used by the Dialogic JSR 309 Connector.

Optionally, the Dialogic JSR 309 Connector supports turning on SIP Session Timers between the JSR 309 driver and the Dialogic PowerMedia XMS. In this version of the JSR 309 Connector, the SIP Session Timers are turned on by default. The application can modify the parameters for the SIP Session Timers when configuring factory properties:

- `mediaserver.sessionTimer.maxTimeout` – defines SIP Session timeout in seconds. Default: 1800 (seconds).
- `mediaserver.sessionTimer.switch` – Turns the SIP Session Timer function on or off. Allowed values are: "on" or "off". Default: "on".

The Verification Application configures the Dialogic JSR 309 Connector properties:

```
...
Properties factoryProperties = new Properties();
for ( PropertyInfo prop: connectorProperty ) {
    log.debug("initDriver() - =====");
    log.debug("initDriver() - Name: " + prop.name);
    log.debug("initDriver() - Description: " + prop.description);
    log.debug("initDriver() - Required: " + new Boolean(prop.required).toString() );
    log.debug("initDriver() - Value: " + prop.defaultValue);
    if ( prop.name.compareToIgnoreCase("connector.sip.address") == 0 )
    {
        if (prop.defaultValue.compareToIgnoreCase(new_connector_sip_address.toString()) != 0)
        {
            log.debug("initDriver() - New Value: " + new_connector_sip_address);
            prop.defaultValue = new_connector_sip_address;
        }
    }
    .....
    factoryProperties.setProperty(prop.name, prop.defaultValue);
}
```


The application creates a new properties factory in which it will store all required parameters for the Dialogic JSR 309 Connector to start properly. It reads the locally stored application properties file configuration of each required parameter and compares it to the value automatically picked up by the Dialogic JSR 309 Connector. It then takes the newest value for each of the required parameters and stores it in new properties factory.

The Dialogic JSR 309 Connector factory is created with the new set of parameters.

```
....  
    mscFactory = dlgcDriver.getFactory(factoryProperties);  
....
```

The Dialogic JSR 309 Connector factory (mscFactory) is now created with a new set of required parameters. Now, the Dialogic JSR 309 Connector interface can be used.

5. Troubleshooting

This section provides basic troubleshooting techniques for the Dialogic JSR 309 Connector.

Logging

The Dialogic JSR 309 Connector and its Verification Application uses the Apache Log4j2 (version 2) logging facility. The connector makes use of *log4j2.xml* file for its logging configuration. With the introduction of log4j2, it is possible to change the log levels without stopping/restarting any components. All the user needs to do is open *log4j2.xml* and change the logging to desired level. Log configuration file, *log4j2.xml* can be found at:

```
${CATALINA_HOME}/conf/Dialogic/log4j2.xml
```

As per *log4j2.xml* configuration the Dialogic JSR 309 Connector and Verification Applications log output file can be found in the *\$CATALINA_HOME/logs/* directory called *Dialogic.log*.

Refer to the following to see *log4j2.xml* in detail.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration monitorInterval="10" status="ERROR">
  <Appenders>
    <File name="dialogic" fileName="../../logs/Dialogic.log" append="false">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %class{36} %L %M - %msg%xEx%n"/>
    </File>
    <Console name="STDOUT" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %class{36} %L %M - %msg%xEx%n"/>
    </Console>
  </Appenders>

  <Loggers>
    <Logger name="com.vendor.dialogic" level="ERROR">
      <AppenderRef ref="dialogic"/>
      <!-- AppenderRef ref="STDOUT"/ -->
    </Logger>
    <Logger name="play" level="ERROR">
      <AppenderRef ref="dialogic"/>
      <!-- AppenderRef ref="STDOUT"/ -->
    </Logger>
    <Logger name="base" level="ERROR">
      <AppenderRef ref="dialogic"/>
      <!-- AppenderRef ref="STDOUT"/ -->
    </Logger>
  </Loggers>
</Configuration>
```

For details of the *log4j2.xml* configuration, refer to the following information:

- **monitorInterval** – Parameter defines how often Log4j2 facility will automatically detect changes to the configuration file and reconfigure itself. The default is 10 seconds.
 - **Appenders:**
 - Provided *log4j2.xml* file defines two streams (Appenders) that it will send logging to: a file (*Dialogic.log*) and a system console. Each individual logger has a choice of which appender to use.
 - **Loggers:**
 - Provided *log4j2.xml* file Loggers section provides a logger configuration for various Java source packages:
 - `com.vendor.dialogic` is a Dialogic JSR 309 Connector.
 - `play & base` is a Dialogic JSR 309 Connector Verification Application.

Note: Each logger can be set individually to the appropriate level of logging and each logger can be individually configured to log to file, STDOUT, or both.

Note that default logging level is set to *ERROR*, which will cause the *Dialogic.log* file to be empty unless there are errors.

Refer to the Apache Log4j 2 documentation at <http://logging.apache.org/log4j/2.x> for details.

Additional platform component logging, configuration, and modifications can be accomplished via appropriate Application Server Administration page. Refer to the platform specific documentation for details.

Dialogic JSR 309 Connector and Verification Application Troubleshooting

1. The Verification Application first opens the application properties. If the path is not set or the properties file does not exist, the DEBUG log file will show an error as follows:

```
13:53:55.851 INFO   play.AsyncPlayer 136 init - init() - Entering
13:53:55.853 DEBUG  play.AsyncPlayer 138 init - init() - servletName: AsyncPlayer
13:53:55.854 INFO   base.ConfigProperty 40 <init> - ConfigProperty() - Entering
13:53:55.854 INFO   base.ConfigProperty 56 LoadProperties - LoadProperties() - Entering
13:53:55.855 INFO   base.ConfigProperty 60 LoadProperties - LoadProperties() - Properties
File = /home/jboss/mss-3.1.633-jboss-as-
7.2.0.Final/standalone/configuration/Dialogic/dlgc_sample_demo.properties
13:53:55.855 ERROR  base.ConfigProperty 77 LoadProperties - java.io.FileNotFoundException:
/home/jboss/mss-3.1.633-jboss-as-
7.2.0.Final/standalone/configuration/Dialogic/dlgc sample demo.properties (No such file
or directory)
13:53:55.856 ERROR  base.ConfigProperty 78 LoadProperties - LoadProperties() - base
Configuration File: /home/apachetomcat8/mss-4.0.21-apache-tomcat-8.0.26
/conf/Dialogic/dlgc_sample_demo.properties load failed
13:53:55.856 INFO   base.ConfigProperty 81 LoadProperties - LoadProperties() - Exiting
```

Successful loading of the properties file will be shown as an INFO message as follows:

```
13:55:46.315 INFO   play.AsyncPlayer 136 init - init() - Entering
13:55:46.317 DEBUG  play.AsyncPlayer 138 init - init() - servletName: AsyncPlayer
13:55:46.318 INFO   base.ConfigProperty 40 <init> - ConfigProperty() - Entering
13:55:46.319 INFO   base.ConfigProperty 56 LoadProperties - LoadProperties() - Entering
13:55:46.320 INFO   base.ConfigProperty 60 LoadProperties - LoadProperties() - Properties
File = /home/jboss/mss-3.1.633-jboss-as-
7.2.0.Final/standalone/configuration/Dialogic/dlgc_sample_demo.properties
13:55:46.321 INFO   base.ConfigProperty 73 LoadProperties - LoadProperties() - base
Configuration File: /home/apachetomcat8/mss-4.0.21-apache-tomcat-8.0.26
/conf/Dialogic/dlgc_sample_demo.properties Successfully Loaded
13:55:46.322 INFO   base.ConfigProperty 81 LoadProperties - LoadProperties() - Exiting
```

2. Make sure that the Dialogic Connector SIP Servlet gets initialized first. If successful, you will see following DEBUG print:

```
09:58:49.959 DEBUG  com.vendor.dialogic.javax.media.mscontrol.spi.DlgcDriver 147
registerDialogic309Driver - DlgcDriver::registerDialogic309Driver() - Application
Platformloading..loading driver now
09:58:49.959 DEBUG  com.vendor.dialogic.javax.media.mscontrol.spi.DlgcDriver 148
registerDialogic309Driver - DlgcDriver:registerDialogic309Drive() calling
DriverManager.re
09:58:49.961 DEBUG  com.vendor.dialogic.javax.media.mscontrol.spi.DlgcDriver 150
registerDialogic309Driver - DlgcDriver:registerDialogic309Drive() returned from
DriverMana
09:58:49.962 DEBUG  com.vendor.dialogic.javax.media.mscontrol.sip.DlgcSipServlet 137 init
- Dialogic Servlet Initialized...
```

The Verification Application will take the new set of required parameters and create a the Dialogic JSR 309 Factory. Successful creation of the JSR 309factory will be shown in the DEBUG logs as follows:

```
13:55:46.700 DEBUG  com.vendor.dialogic.javax.media.mscontrol.spi.DlgcDriver 408
getControlFactory - DlgcDriver::getControlFactory() property passed in:
13:55:46.706 DEBUG  com.vendor.dialogic.javax.media.mscontrol.DlgcMsControlFactory 103
<init> - DlgcMsControlFactory:: CTOR using Dynamic Factory Configuration
13:55:46.709 DEBUG  com.vendor.dialogic.javax.media.mscontrol.sip.DlgcMediaServer 235
<init> - DlgcMediaServer CTOR supporting Dynamic Configuration:
13:55:46.710 DEBUG  com.vendor.dialogic.javax.media.mscontrol.sip.DlgcMediaServer 251
<init> - DlgcMediaServer CTOR using the following XMS SIP Values:  username: msml= Media
Server IP: 146.152.122.4 Media Server SIP Port: 5060
13:55:46.711 DEBUG  com.vendor.dialogic.javax.media.mscontrol.sip.DlgcMediaServer 252
<init> - DlgcMediaServer CTOR using the following XMS Connector AS SIP Values: AS Server
IP: 146.152.122.146 Connector AS SIP Port: 5080
```

SIP Errors

If the PowerMedia XMS returns "503 Service Unavailable", make sure the network is correctly set up by performing the following actions:

1. Verify the available PowerMedia XMS licenses.
2. Check the `/etc/hosts` file configuration.

6. Building and Debugging Sample Demos in Eclipse IDE

The Dialogic JSR 309 Connector distribution package comes with all necessary configuration files and content needed for anyone to build the Verification Application on their own. This section provides the steps to create, compile, build, and debug provided demo application using Eclipse IDE.

Prerequisites

The following components must be installed:

- Latest JDK version supported by desired J2EE platform.
- Eclipse KDE (Version used here: Mars.1 Release (4.5.1)).
- In order to build provided demo applications, obtain two TeleStax platform dependent libraries that are NOT provided as part of the Dialogic JSR 309 Connector distribution package. They can be found under the following directories:
 - (TelScale)
 - *sip-servlets-spec-7.0.3.xxx.jar* (*x* represents the appropriate version of the platform used)
 - *servlet-api.jar*
 - *websocket-api.jar*
 - (Mobicents)
 - *sip-servlets-spec-x.x.xxx.jar* (*x* represents the appropriate version of the platform used)
 - *servlet-api.jar*
 - *websocket-api.jar*

Creating the Build Environment

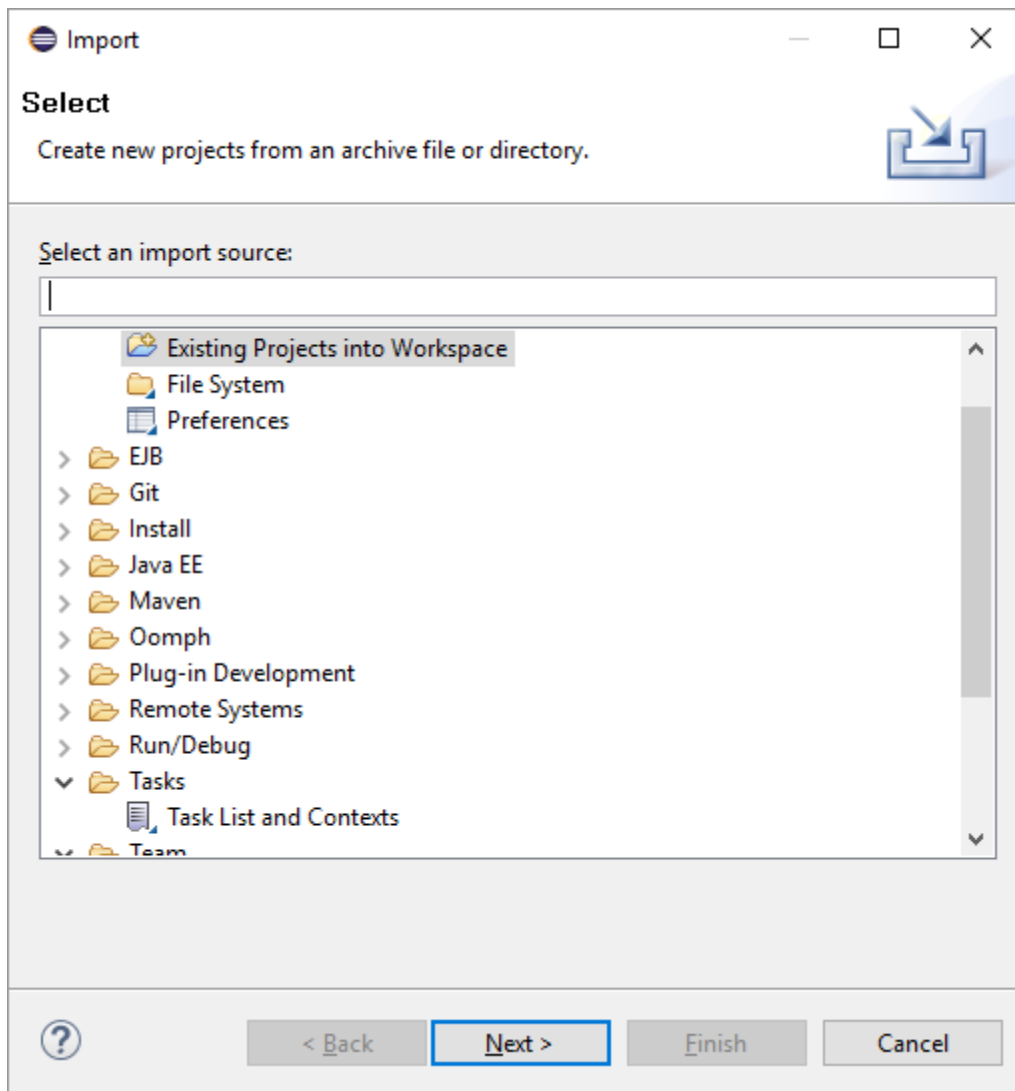
The Dialogic JSR 309 Verification Application source project comes with all the necessary components to compile and build the application WAR file. Follow these steps to create a Dialogic demo build environment:

Prepare the Eclipse Workspace

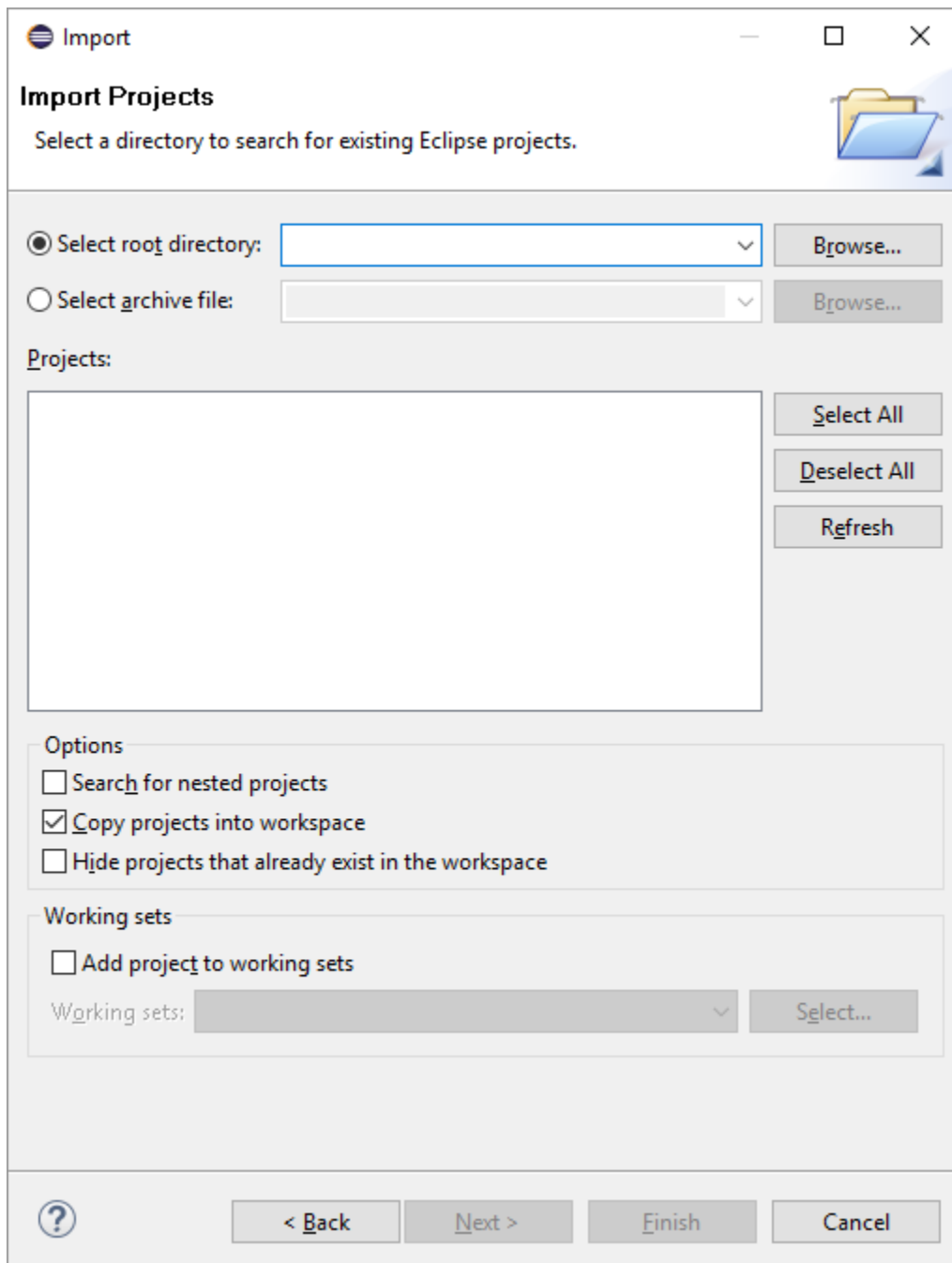
First, prepare the Eclipse workspace:

1. From the distribution package, copy the *DlgcJSR309/application/DialogicSampleDemo* directory and put it in a known location on the system.
2. The *Project/lib/3rdParty* directory contains all required third-party JAR files including the Dialogic JSR 309 Connector JAR files.
3. Copy the required Application Server platform specific libraries into the *Project/lib/AS* directory.
4. Open Eclipse IDE and click **File > Import**.

5. Select **Existing Project into Workspace** and click **Next**.



6. Click **Browse** and navigate to the *Project* directory on the system.



The image shows the 'Import Projects' dialog box in Eclipse. The title bar says 'Import'. The main heading is 'Import Projects' with a subtitle 'Select a directory to search for existing Eclipse projects.' and a folder icon. There are two radio buttons: 'Select root directory:' (selected) and 'Select archive file:'. Each has a text field and a 'Browse...' button. Below is a 'Projects:' section with a large empty list box and three buttons: 'Select All', 'Deselect All', and 'Refresh'. At the bottom are two sections: 'Options' with checkboxes for 'Search for nested projects' (unchecked), 'Copy projects into workspace' (checked), and 'Hide projects that already exist in the workspace' (unchecked); and 'Working sets' with an unchecked checkbox 'Add project to working sets' and a 'Working sets:' dropdown with a 'Select...' button. The footer contains a help icon, '< Back', 'Next >', 'Finish', and 'Cancel' buttons.

Import

Import Projects

Select a directory to search for existing Eclipse projects.

☒ Select root directory: **Browse...**

☐ Select archive file: **Browse...**

Projects:

Options

☐ Search for nested projects


☒ Copy projects into workspace

☐ Hide projects that already exist in the workspace

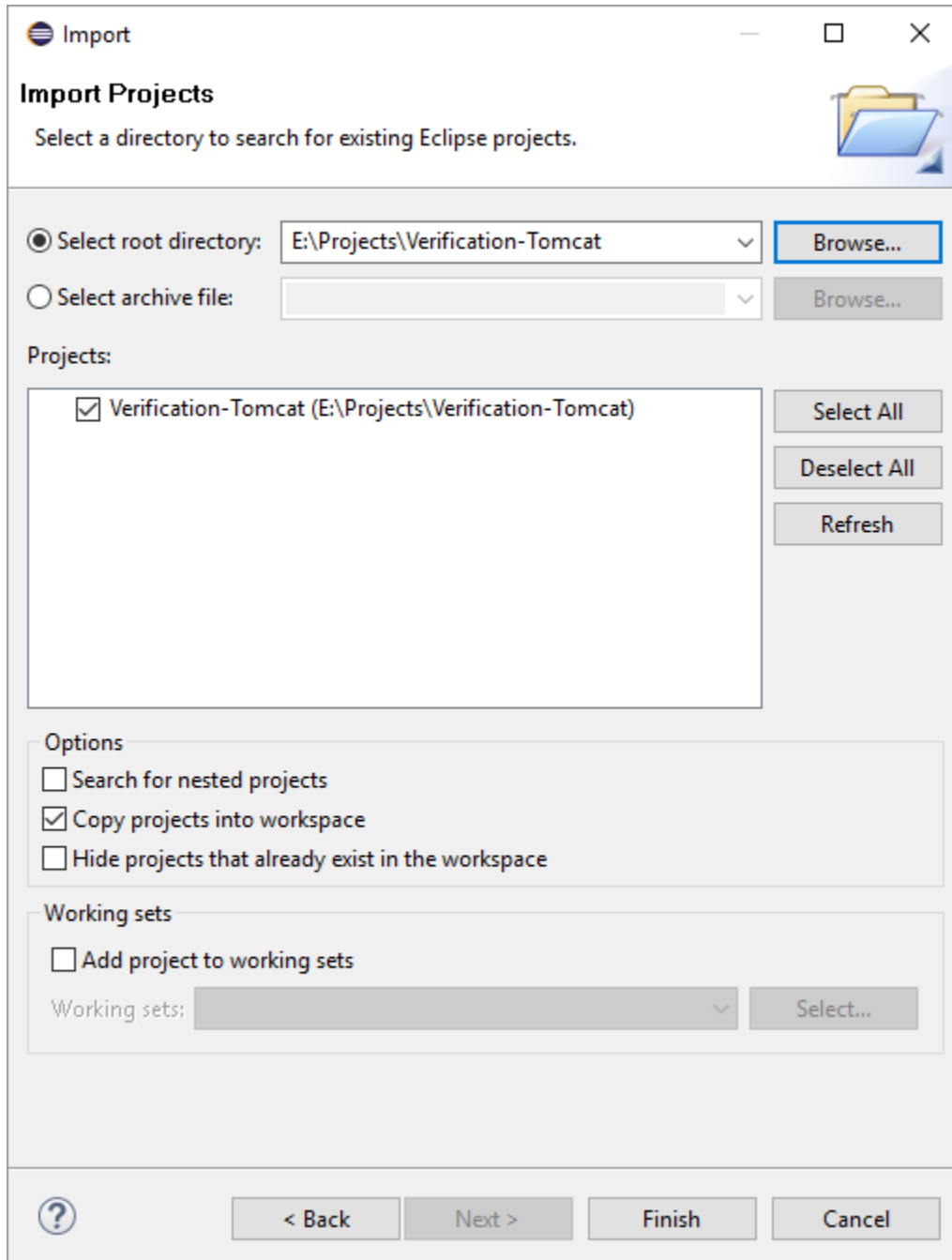
Working sets

☐ Add project to working sets

Working sets: **Select...**



7. Click **Finish**. The Project has been imported to the Eclipse workspace.



The image shows the 'Import Projects' dialog box in Eclipse. The title bar says 'Import'. The main heading is 'Import Projects' with a subtitle 'Select a directory to search for existing Eclipse projects.' and a folder icon. There are two radio buttons: 'Select root directory:' (selected) and 'Select archive file:'. The 'Select root directory:' option has a text field containing 'E:\Projects\Verification-Tomcat' and a 'Browse...' button. The 'Select archive file:' option has an empty text field and a 'Browse...' button. Below these is a list of 'Projects:' with one entry: 'Verification-Tomcat (E:\Projects\Verification-Tomcat)' which is checked. To the right of the list are buttons for 'Select All', 'Deselect All', and 'Refresh'. Below the projects list are two sections: 'Options' and 'Working sets'. The 'Options' section has three checkboxes: 'Search for nested projects' (unchecked), 'Copy projects into workspace' (checked), and 'Hide projects that already exist in the workspace' (unchecked). The 'Working sets' section has a checkbox 'Add project to working sets' (unchecked) and a 'Working sets:' label followed by a dropdown menu and a 'Select...' button. At the bottom of the dialog are four buttons: a help button (question mark), '< Back', 'Next >', and 'Finish'. The 'Finish' button is highlighted with a blue border.

Import

Import Projects

Select a directory to search for existing Eclipse projects.

☒ Select root directory: E:\Projects\Verification-Tomcat

☐ Select archive file:

Projects:

- ☒ Verification-Tomcat (E:\Projects\Verification-Tomcat)

Options

- ☐ Search for nested projects
- ☒ Copy projects into workspace
- ☐ Hide projects that already exist in the workspace

Working sets

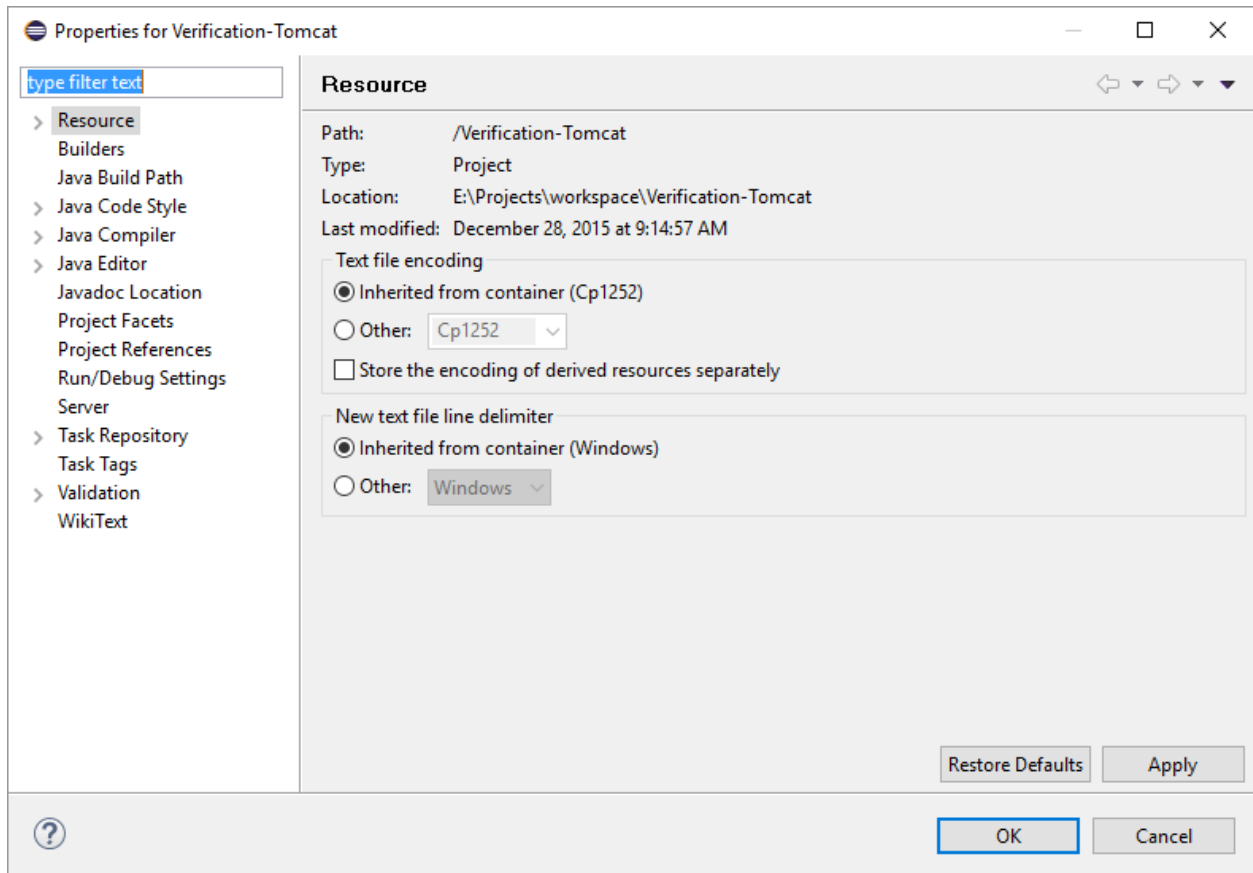
☐ Add project to working sets

Working sets:

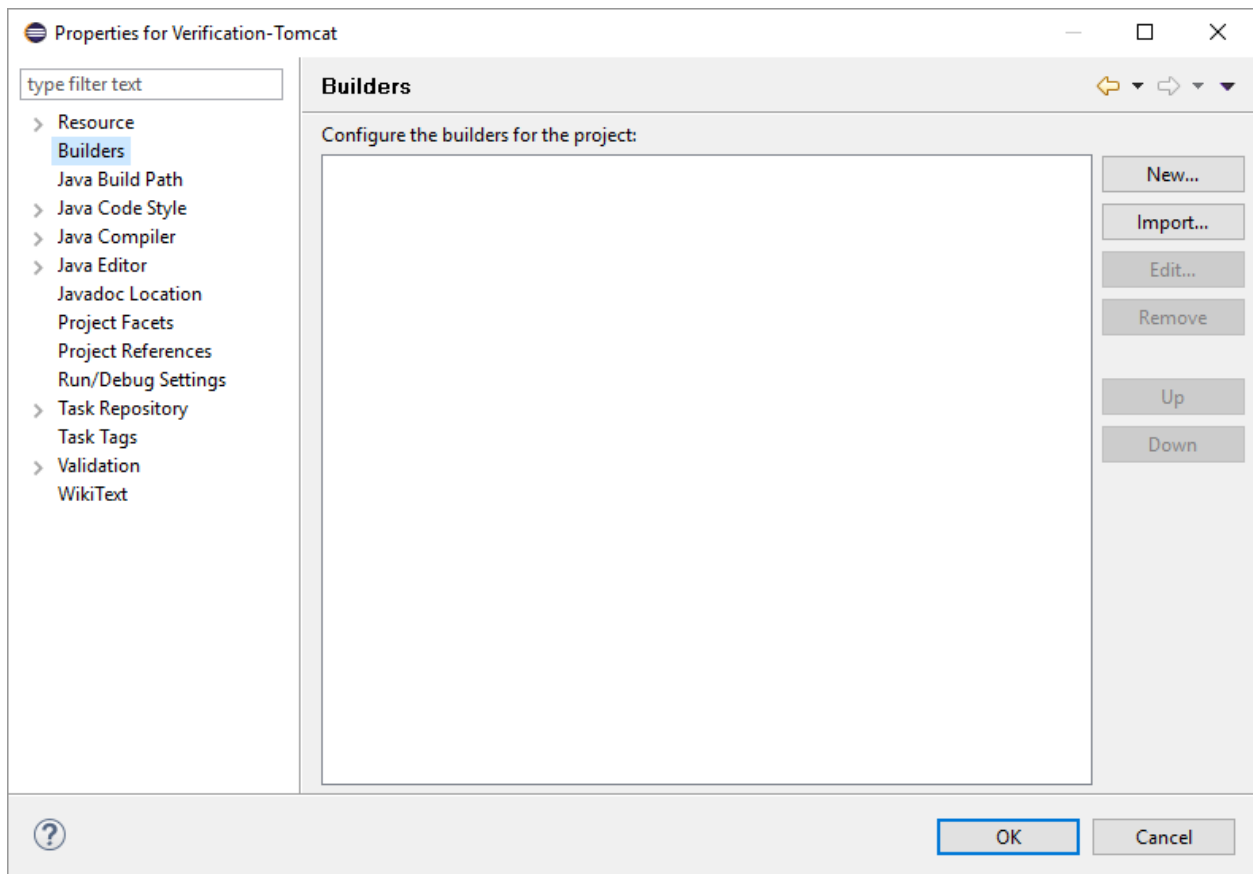
Configure the Application

To configure the application, edit the following settings:

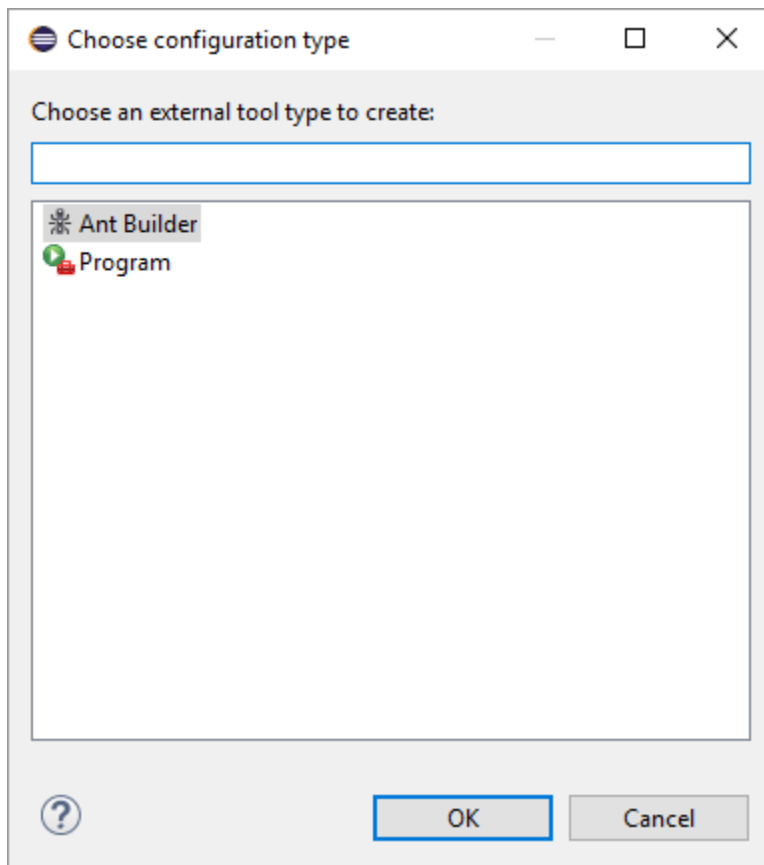
1. Right-click the **Verification-Tomcat** project directory (for this example) and select **Properties**.



2. Click **Builders**, and then click **New**.



3. Select **Ant Builder** and click **OK**.



4. Select any name for this ANT build process ("New_Build" is the default).

The screenshot shows the 'Edit Configuration' dialog box for an ANT build process. The title bar reads 'Edit Configuration' with a close button. Below the title bar, the text 'Edit launch configuration properties' is displayed, followed by the instruction 'Please specify the location of the external tool you would like to configure.' and a green play button icon with a red toolbox icon.

The 'Name' field is set to 'New_Builder'. Below this is a tabbed interface with the following tabs: 'Main', 'Refresh', 'Targets', 'Classpath', 'Properties', 'JRE', 'Environment', and 'Build Options'. The 'Main' tab is selected.

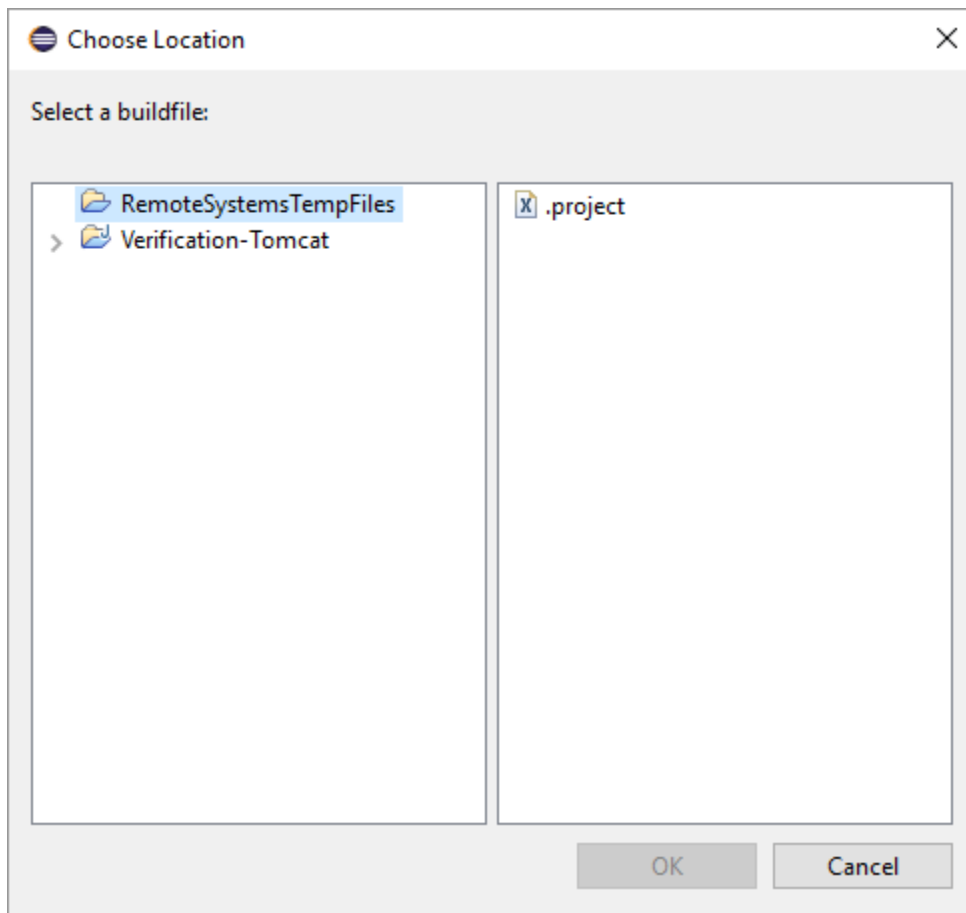
Under the 'Main' tab, there are three sections:

- Buildfile:** A text field with a 'Browse Workspace...' button, a 'Browse File System...' button, and a 'Variables...' button.
- Base Directory:** A text field with a 'Browse Workspace...' button, a 'Browse File System...' button, and a 'Variables...' button.
- Arguments:** A large text area with a 'Variables...' button at the bottom right.

A note at the bottom of the 'Arguments' section states: 'Note: Enclose an argument containing spaces using double-quotes (").'

At the bottom of the dialog, there is a checkbox labeled 'Set an Input handler' which is checked. Below this are 'Revert' and 'Apply' buttons. At the very bottom are 'OK' and 'Cancel' buttons.

5. In the **Buildfile** section, click **Browse Workplace**.
6. Click the project directory in the left window, select *build.xml* in the right window, and then click **OK**.



7. In the **Base Directory** field, click **Browse Workspace**.

Edit Configuration

Edit launch configuration properties

Create a configuration that will run an Ant build file during a build.

Name: New_Builder

Main Refresh Targets Classpath Properties JRE Environment Build Options

Buildfile:

`${workspace_loc:/Verification-Tomcat/build.xml}`

Browse Workspace... Browse File System... Variables...

Base Directory:

Browse Workspace... Browse File System... Variables...

Arguments:

Variables...

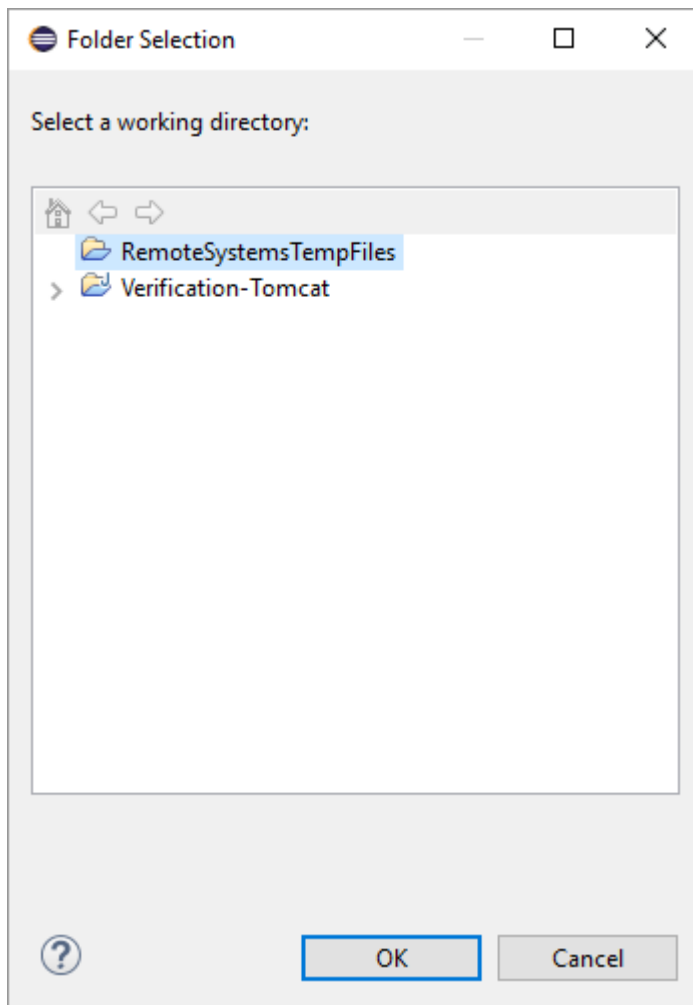
Note: Enclose an argument containing spaces using double-quotes (").

☒ Set an Input handler

Revert Apply

? OK Cancel

8. Select the project directory and click **OK**.



9. Click the **Targets** tab.

Edit Configuration

Edit launch configuration properties

Create a configuration that will run an Ant build file during a build.

Name:

Buildfile:

Base Directory:

Arguments:

Note: Enclose an argument containing spaces using double-quotes ("").

☒ Set an Input handler

Revert Apply

OK Cancel

10. For **After a "Clean"**, **Manual Build**, and **Auto Build**, set the default target by clicking **Set Targets** and selecting **demo [default]** for each field.

The screenshot shows the 'Edit Configuration' dialog box with the title 'Edit Configuration' and a close button. Below the title bar, the text 'Edit launch configuration properties' is displayed, followed by the instruction 'Create a configuration that will run an Ant build file during a build.' and a green play button icon. The main area of the dialog is titled 'Name: New_Builder'. Below this, there is a tabbed interface with tabs for 'Main', 'Refresh', 'Targets' (which is selected), 'Classpath', 'Properties', 'JRE', 'Environment', and 'Build Options'. The 'Targets' tab contains four sections: 'After a "Clean":', 'Manual Build:', 'Auto Build:', and 'During a "Clean":'. Each section has a text field and a 'Set Targets...' button. The 'After a "Clean":' field contains '<default target selected>'. The 'Manual Build:' field contains '<default target selected>'. The 'Auto Build:' field contains '<default target selected>'. The 'During a "Clean":' field contains 'clean'. At the bottom of the dialog, there are buttons for 'Revert', 'Apply', 'OK', and 'Cancel'. A help icon (?) is located in the bottom left corner.

Edit Configuration

Edit launch configuration properties

Create a configuration that will run an Ant build file during a build.

Name: New_Builder

Main Refresh **Targets** Classpath Properties JRE Environment Build Options

After a "Clean":
<default target selected> Set Targets...

Manual Build:
<default target selected> Set Targets...

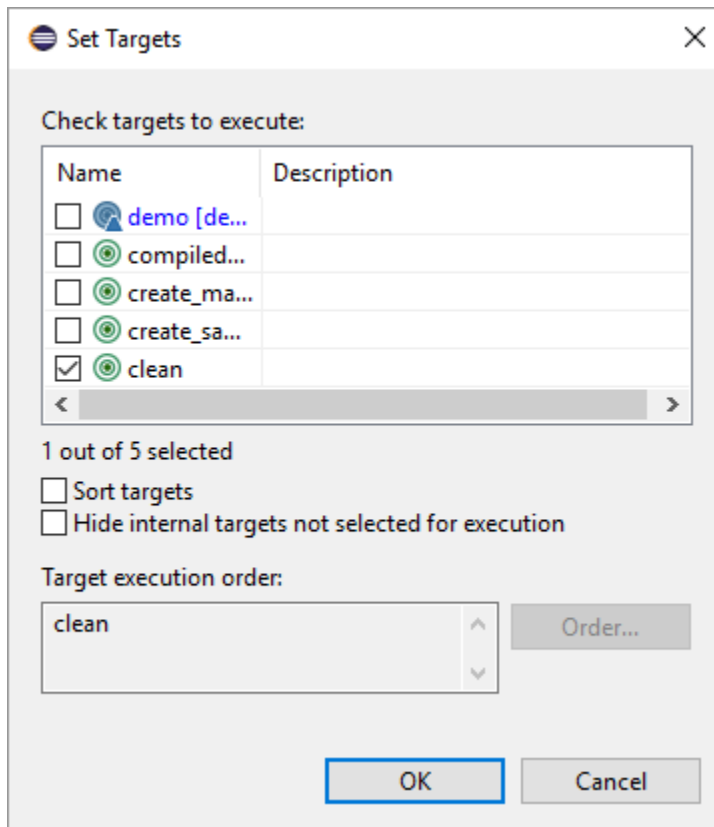
Auto Build:
<default target selected> Set Targets...

During a "Clean":
clean Set Targets...

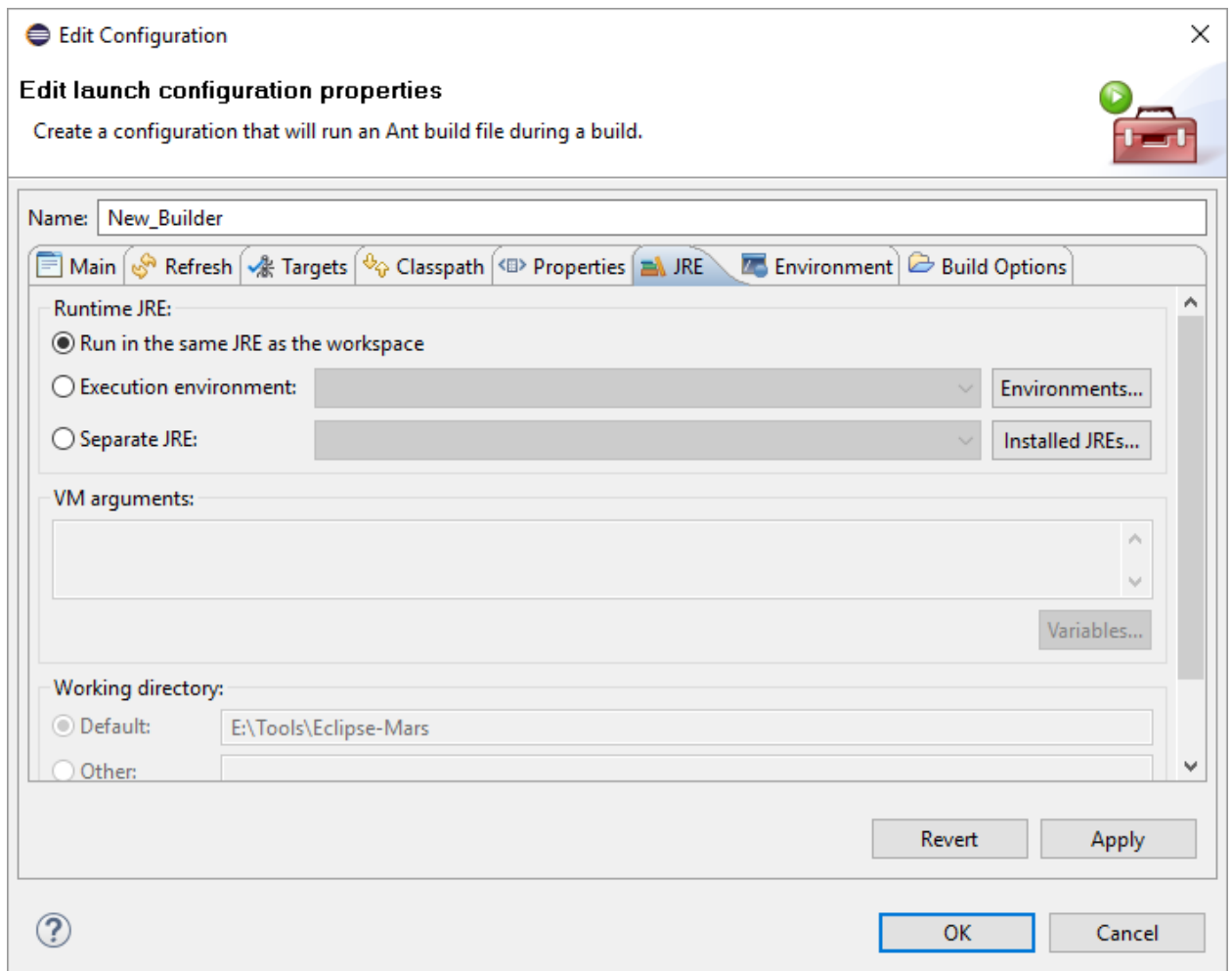
Revert Apply

? OK Cancel

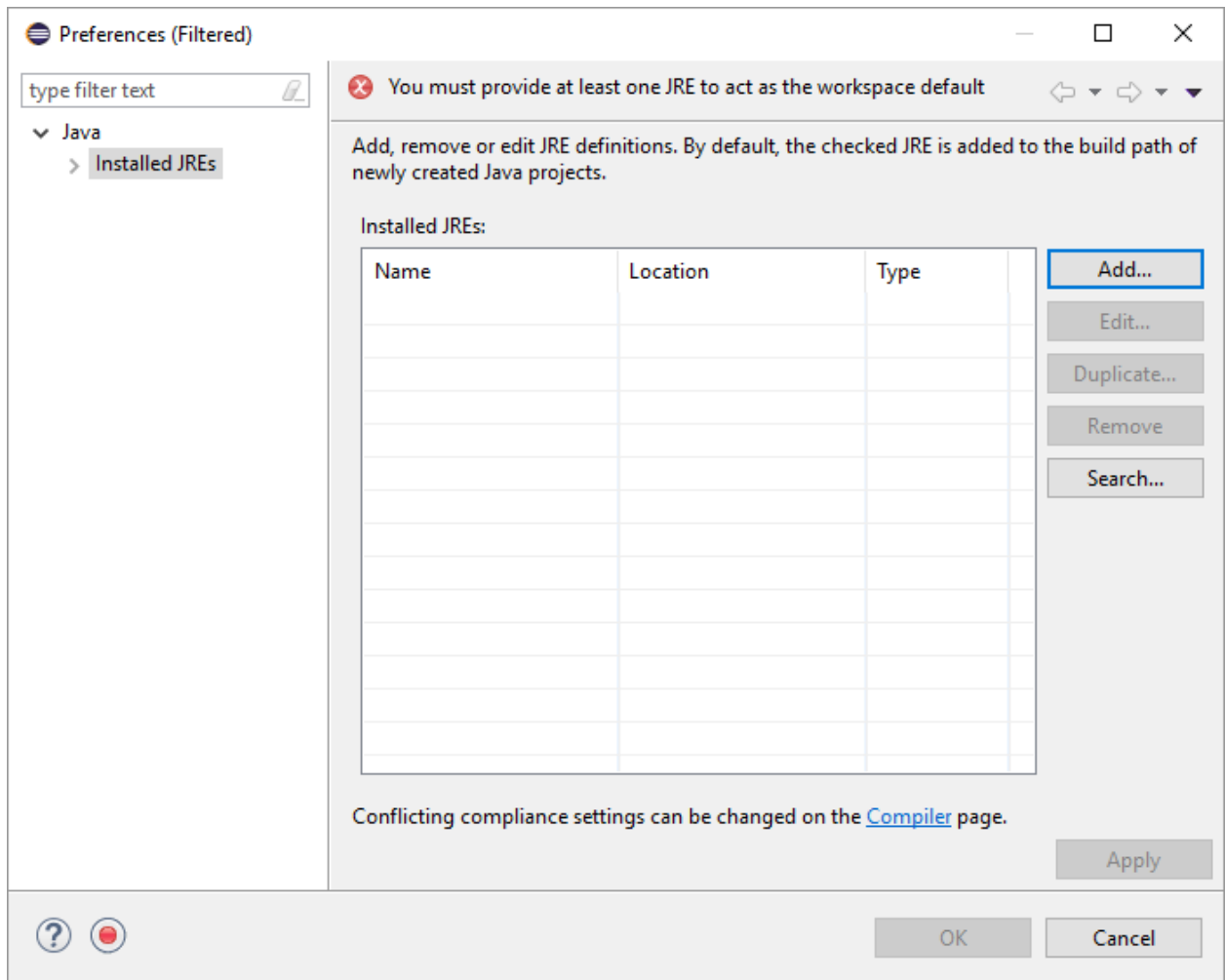
11. For **During a "Clean"**, click **Set Targets**, make sure that **clean** is the only target selected, and then click **OK**.



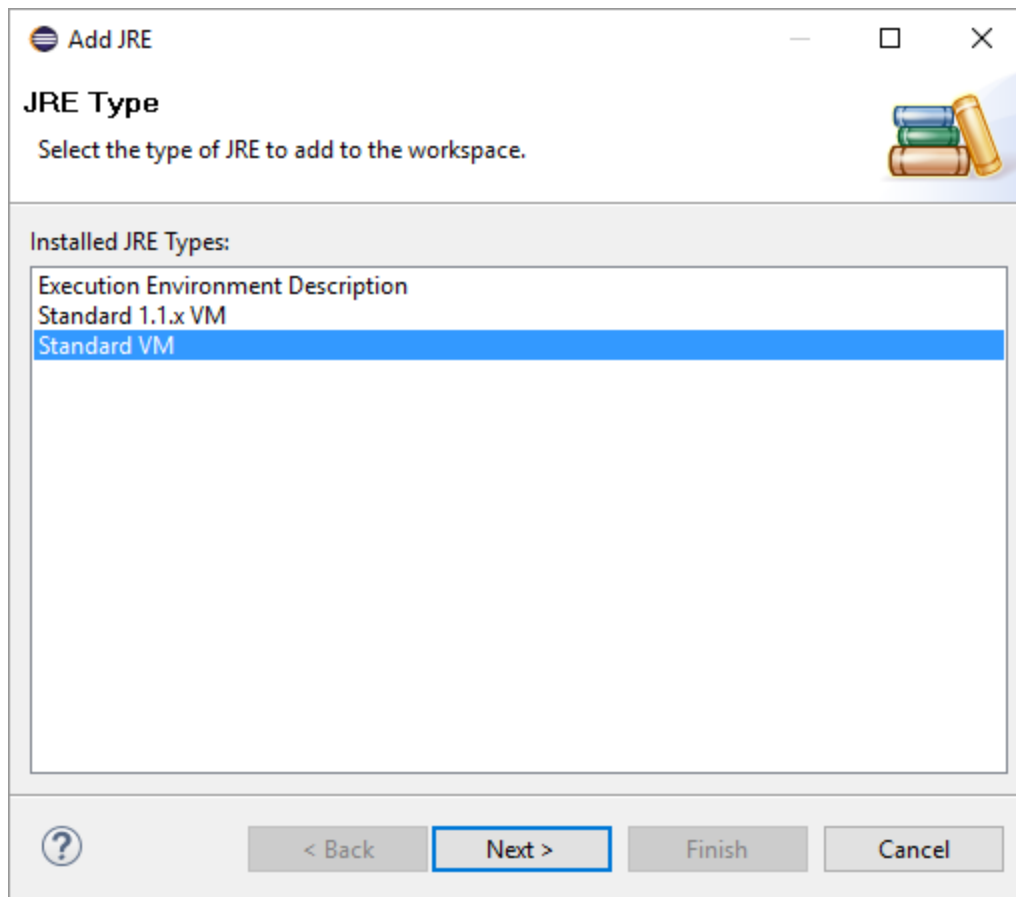
12. Click the **JRE** tab.



13. In the **Separate JRE** drop-down menu, select the appropriate JDK as required by the platform. If not present there, click **Installed JREs** and then click **Add**.



14. Click **Next**.



15. Click **Directory** and navigate to the appropriate JDK. When done, click **Finish**.

Add JRE

JRE Definition
Specify attributes for a JRE

JRE home:

JRE name:

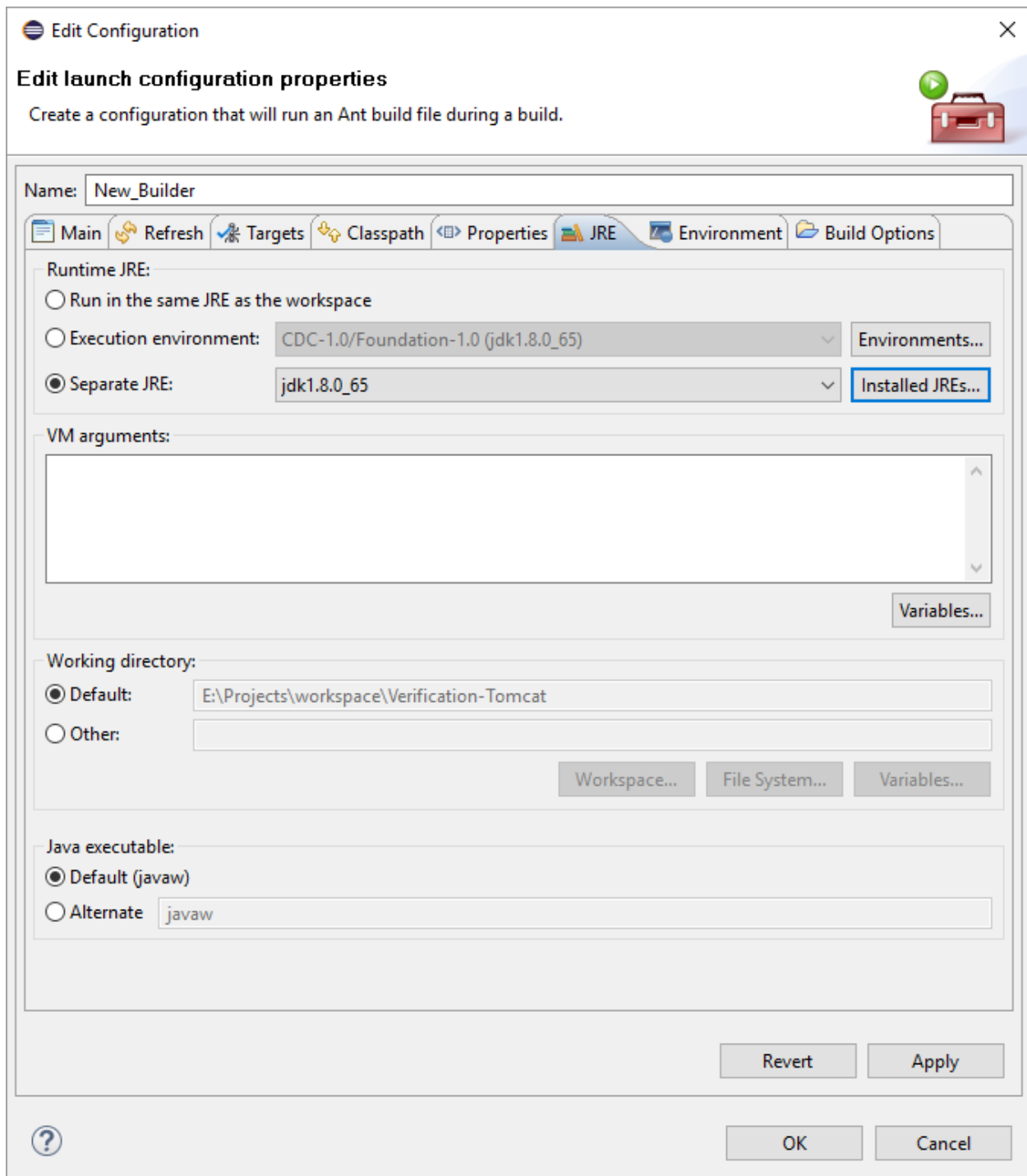
Default VM arguments:

JRE system libraries:

- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\resources.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\rt.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\jsse.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\jce.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\charsets.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\jfr.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\ext\access-bri
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\ext\clrddata.ja
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\ext\dnsns.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\ext\jaccess.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\ext\jfxrt.jar
- > C:\Program Files\Java\jdk1.8.0_65\jre\lib\ext\localedata

[illegible]

17. Verify that the **Separate JRE** field points to the desired JDK, and then click **OK**.



Edit Configuration

Edit launch configuration properties
Create a configuration that will run an Ant build file during a build.

Name:

Main Refresh Targets Classpath Properties **JRE** Environment Build Options

Runtime JRE:

- ☐ Run in the same JRE as the workspace
- ☐ Execution environment: CDC-1.0/Foundation-1.0 (jdk1.8.0_65) Environments...
- ☒ **Separate JRE:** Installed JREs...

VM arguments:

Variables...

Working directory:

- ☒ **Default:**
- ☐ **Other:**

Workspace... File System... Variables...

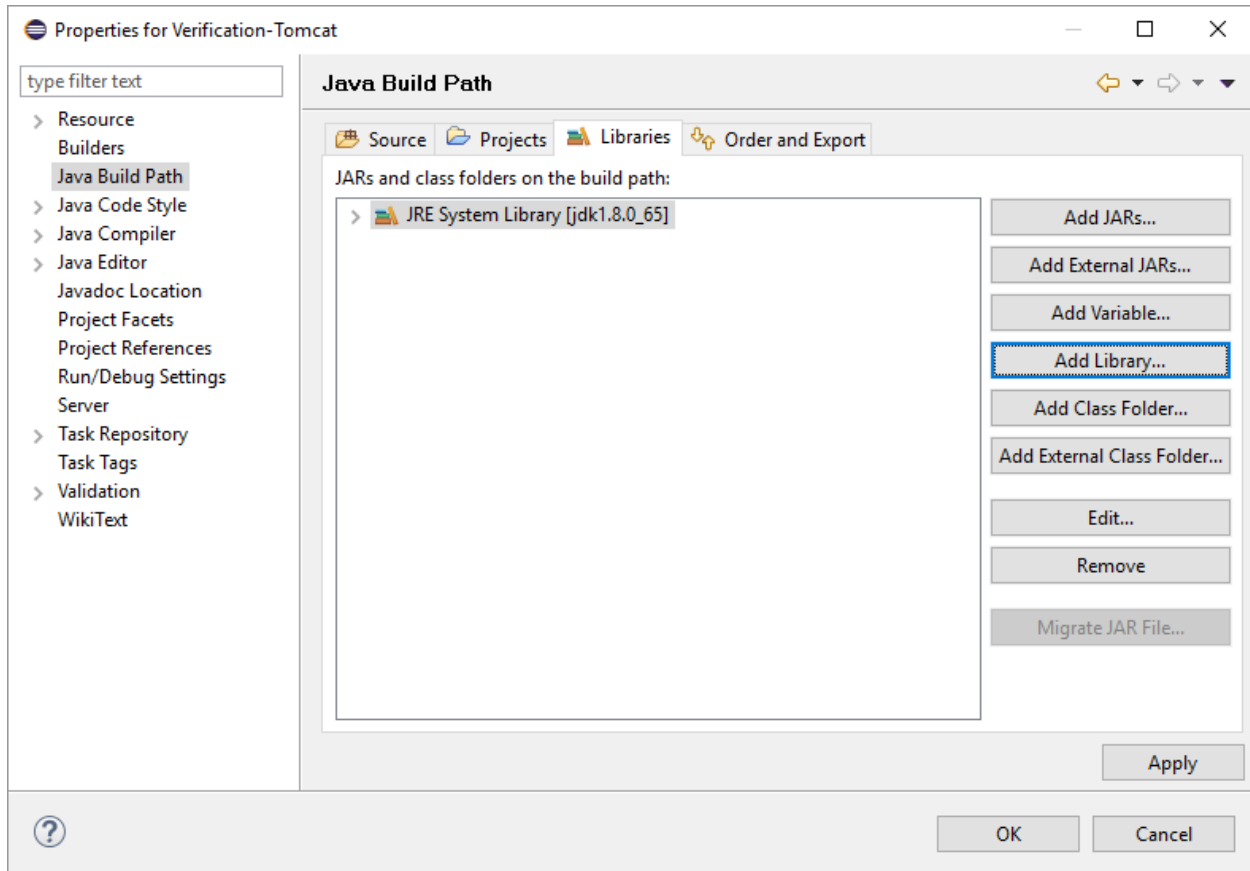
Java executable:

- ☒ **Default (javaw)**
- ☐ **Alternate**

Revert Apply

? OK Cancel

18. Click **Java Build Path** and verify that all JAR files are correctly pointed to. If not, click **Add JARs** and select all the JAR files from within *Project/lib* directories. Once that is done, click **OK**.



Building the Project

After a successful project installation and configuration, a project can be built. In Eclipse, select the newly created project. In the **Project** menu, click **Build All**. Successful build content will be shown in the **Console** view in Eclipse as follows.

```
<terminated> New_Builder [Ant Builder] C:\Program Files\Java\jdk1.8.0_65\bin\javaw.exe (Dec 28, 2015, 10:41:32 AM)
Buildfile: E:\Projects\workspace\Verification-Tomcat\build.xml
create manifest:
compiledemo:
[mkdir] Created dir: E:\Projects\workspace\Verification-Tomcat\WEB-INF\classes
[javac] Compiling 3 source files to E:\Projects\workspace\Verification-Tomcat\WEB-INF\classes
create sample demo war:
[copy] Copying 1 file to E:\Projects\workspace\Verification-Tomcat\WEB-INF
[copy] Copying 1 file to E:\Projects\workspace\Verification-Tomcat\WEB-INF
[jar] Building jar: E:\Projects\workspace\Verification-Tomcat\demo_app\dlgc_sample_demo.war
[delete] Deleting directory E:\Projects\workspace\Verification-Tomcat\WEB-INF
[delete] Deleting: E:\Projects\workspace\Verification-Tomcat\MANIFEST.MF
demo:
BUILD SUCCESSFUL
Total time: 1 second
```

The newly built application WAR file will be located under the *demo_app* directory named *dlgc_sample_demo.war*. In order to deploy this application, follow the same deployment instructions as described in [Deploy the Dialogic JSR 309 Verification Application](#).

Configuring Eclipse Project and TeleStax Application Server Deployed Application for Remote Debugging

In order to connect the newly created project to the deployed WAR file in the Application Server for debugging purposes, developers need to do the following:

1. Configure Application Server platform for remote debugging.
2. Have Eclipse successfully build the Dialogic JSR 309 Connector Demo Application WAR file and deploy it in the desired Application Server platform. Refer to [Deploy the Dialogic JSR 309 Verification Application](#).

Configuring the Application Server Platform for Remote Debugging

Configure the Application Server platform for remote debugging as follows:

1. Stop the Application Server.
2. Edit *catalina.sh*:

```
${CATALINA_HOME}/bin/catalina.sh
```

3. Script the file and add the following line to enable remote debugging as illustrated below in **RED**:

```
# ----- Execute The Requested Command -----

### Dialogic additions
JAVA_OPTS="$JAVA_OPTS -
Dlog4j.configurationFile=${CATALINA_HOME}/conf/Dialogic/log4j2.xml"
#Optional For Remote Debugging
CATALINA_OPTS="$CATALINA_OPTS -Xdebug -
Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=n"
### END - Dialogic additions
# Bugzilla 37848: only output this if we have a TTY
```

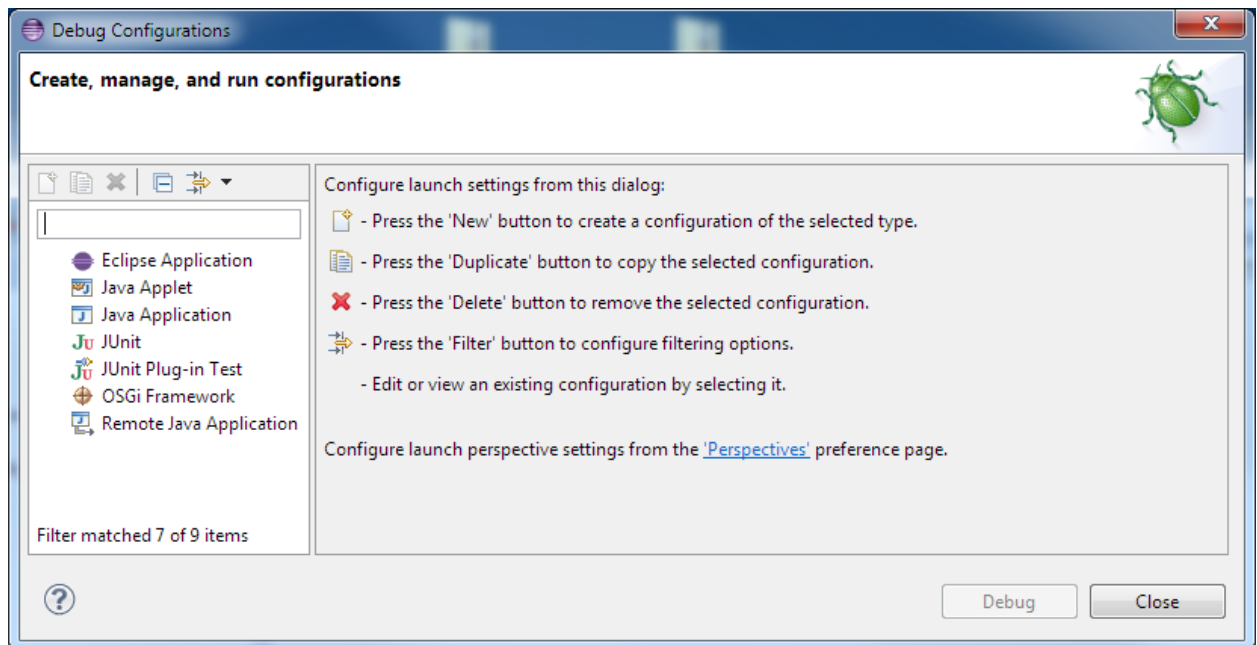
Note: The socket address specified above is 8787 but any port of choice can be used. Any port used needs to be enabled in a firewall in order to allow communication through it.

4. Start the Application Server and make sure there are no errors in the console.

Eclipse Project Configuration for Remote Debugging

To configure the existing and working Dialogic JSR 309 Connector project, the remote debugging section needs to be configured. In Eclipse, go to the **Run** menu and click **Debug Configurations**.

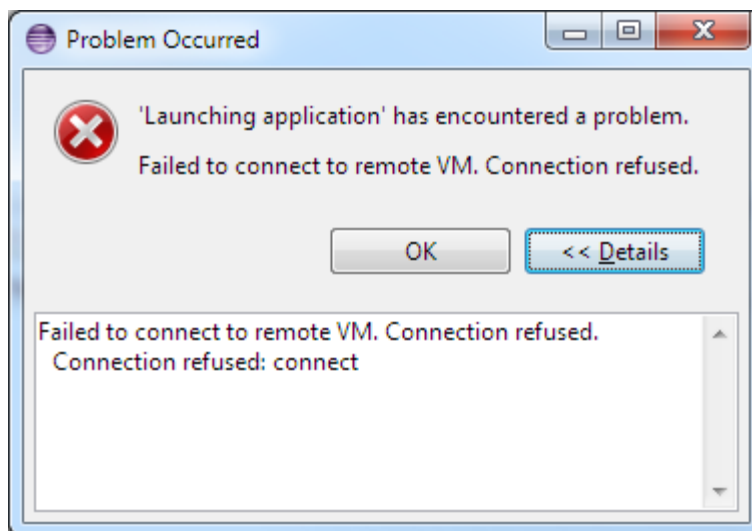
1. In the **Debug Configurations** window, double-click **Remote Java Application**.



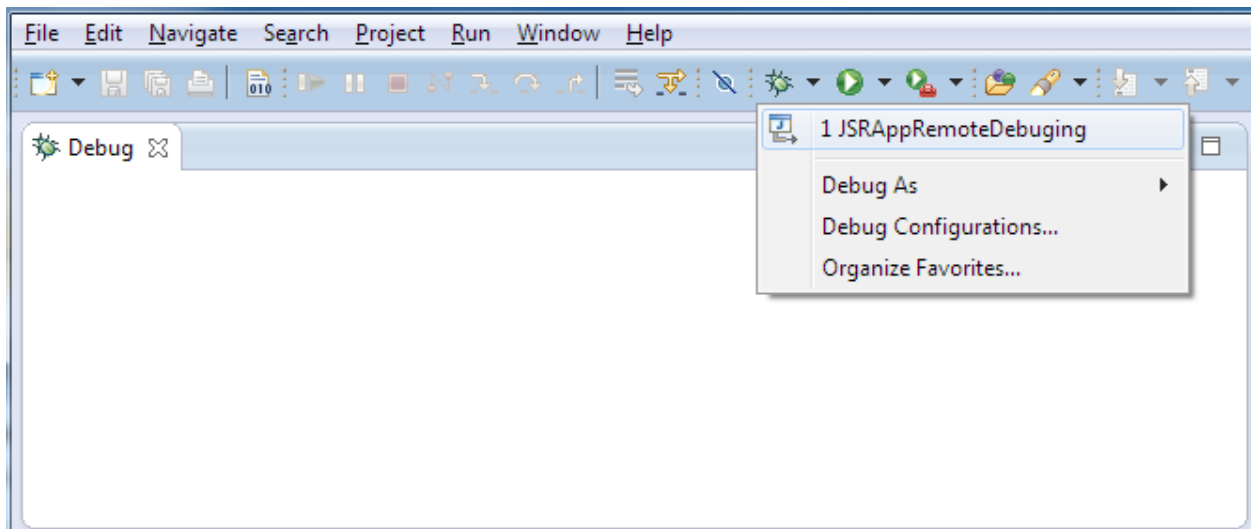
- Specify the **Name** for this remote debugging configuration (for example, JSRAppRemoteDebugging) and specify the **Host** and **Port** address of the Application Server for the debugger to connect to.

The screenshot shows the 'Remote Debugging Configuration' dialog box in Eclipse. The 'Name' field is set to 'JSRAppRemoteDebugging'. The 'Connect' tab is selected, showing the 'Project' as 'Verification-Tomcat', 'Connection Type' as 'Standard (Socket Attach)', 'Host' as '146.152.122.146', and 'Port' as '8787'. There is an unchecked checkbox for 'Allow termination of remote VM'. At the bottom, there are 'Revert', 'Apply', 'Debug', and 'Close' buttons.

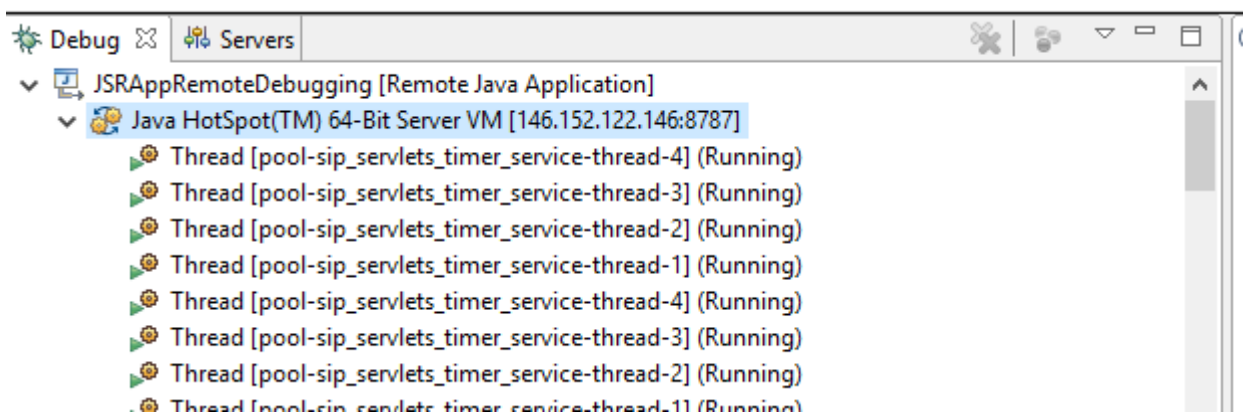
- Click **Debug**. The Application Server needs to be running at this point. If not, Eclipse will report a connection error message. If the Application Server is running but Eclipse is still reporting a connection error, this could be due to either a port mismatch between Eclipse and the Application Server firewall settings so the specified port cannot be used, or there was a port conflict.



4. Open the **Debug** perspective in Eclipse (**Windows > Open Perspective > Debug**). If nothing appears in the **Debug** section, then a connection to the Application Server has not been established. To connect/reconnect, go to the debug icon on the toolbar and choose the newly created remote debugging configuration.



Once the **remote debugging configuration** is selected and a connection is established, the content of the **Debug** window should show running threads. The Eclipse project is connected to the build application that is deployed in TeleStax Application Server.



7. Appendix A: Dialogic JSR 309 Connector Environment Setup

This section describes a quick way to set up the environment for the Dialogic JSR 309 Connector.

For system requirements and supported platforms, see [Dialogic JSR 309 Connector Requirements](#).

This section does not go into the details of the Application Server platform, but it will help build the Application Server quickly for use.

It should be noted that OS level configuration should include the following:

- Enable NTP (Network Time Protocol)
- Enable ports in firewall (if applicable)

Note: The following IP ports must be enabled in the firewall for the system to operate correctly: 8080 (TCP), 9990 (TCP), 5080 (UDP and TCP), and optional remote debugging port 8787 (TCP).

If you need further details on TeleStax Application Server, visit www.telestax.com.

Installing and Configuring the TeleStax Apache-Tomcat Application Server

Note: If you are familiar with TeleStax AS or are planning to deploy on an existing TeleStax setup, proceed to [Installing the Dialogic JSR 309 Connector](#).

This section describes the installation and configuration instructions for the Application Server:

- [Preinstallation Setup](#)
- [TeleStax Installation](#)
- [TeleStax Configuration](#)
- [TeleStax Startup](#)
- [TeleStax Verification](#)

Preinstallation Setup

Install the OS supported by TeleStax. Refer to www.telestax.com for details. For the purpose of this documentation, a CentOS 6.4/6.5 64-bit operating system with minimum installation options was used. Follow the steps below:

1. Log in to the newly installed operating system and install zip/unzip package:

```
yum install zip unzip
```

2. Copy and install latest 1.8 version of JDK .rpm package, which can be downloaded from www.oracle.com.

```
rpm -ivh jdk-8u60-linux-x64.rpm
```

3. Under the user home root directory, edit the `.bashrc` file and include the following export lines:

For TelScale AS:

```
export JAVA_HOME=/usr/java/jdk1.8.0_60
export CATALINA_HOME=/home/apachetomcat8/TelScale-SIP-Servlets-7.0.3.329-apache-tomcat-8.0.26
```

For Mobicents AS:

```
export JAVA_HOME=/usr/java/jdk1.8.0_60
export CATALINA_HOME=/home/apachetomcat8/mss-4.0.21-apache-tomcat-8.0.26
```

4. Save the file and execute the following command for the changes to take effect:

```
source /home/jboss/.bashrc
```

5. Edit the `/etc/hosts` file and add a line at the very top of the file that corresponds to your system's IP address and the hostname.

Here is an example of `hosts` file content:

```
xxx.xxx.xxx.xxx TeleStaxAS
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

Note: You will have to be a root to have privileges to do modify the hosts file.

Note: The hostname should match exactly what is returned by executing "hostname" from the command prompt.

Note: This must be the first line in the `/etc/hosts` file. If not, you might encounter a "503 Service Unavailable" error.

6. To activate the changes made to the `hosts` file, run the following command at the prompt:

```
service network restart
```

Firewall Configuration

Several ports must be allowed to go through the firewall. Ports that are going to be in use are 8080, 9990, 5080, and the optional remote debugging port 8787. To do this, edit the `/etc/sysconfig/iptables` file:

```
vi /etc/sysconfig/iptables
```

Then, add the ports as illustrated below:

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 8080 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 8443 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 9990 -j ACCEPT
```

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 5080 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 5080 -j ACCEPT
#optional port needs to be opened if platform remote debugging will be used.
-A INPUT -m state --state NEW -m tcp -p tcp --dport 8787 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Save the file and restart the firewall for the changes to take effect by executing the following command:

```
service iptables restart
```

TeleStax Installation

There are many ways to install the platform, but for this example, the user "apachetomcat8" is created and used. The example "apachetomcat8" user account will be used throughout the installation process as a home directory for the platform.

After creating "apachetomcat8" user account in the system, log in to that account. Then, copy the desired platform distribution to the user home directory. In this example, the user home directory is */home/apachetomcat8*.

Copy the following to the system under the */home/apachetomcat8* directory and unzip them.

```
(TelScale AS) - TelScale-SIP-Servlets-7.0.3.329-apache-tomcat-8.0.26.zip
(Mobicents AS) - mss-4.0.21-apache-tomcat-8.0.26.zip
```

TeleStax Configuration

To properly configure the newly installed platform in a system, specific modifications of the system configuration need to take place.

Edit the *server.xml* file:

```
${CATALINA_HOME}/conf/server.xml
```

Locate the lines that need to be modified. They are identified below in **RED** in the "Before" section. Replace the lines in **RED** in the "Before" section with the lines in **RED** from the "After" section.

Before

```
<!-- Define a SIP Connector on port 5080 -->
<Connector port="5080"
  ipAddress = "127.0.0.1"
  protocol="org.mobicents.servlet.sip.startup.SipProtocolHandler"
  signalingTransport="udp"/>

<!-- Define the default TCP SIP Connector -->
<Connector port="5080"
  ipAddress = "127.0.0.1"
  protocol="org.mobicents.servlet.sip.startup.SipProtocolHandler"
  signalingTransport="tcp"/>

<!-- Define the default TLS SIP Connector -->
<Connector port="5081"
```



```

        ipAddress = "127.0.0.1"
        protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"
        signalingTransport="tls"/>

<!-- Define the default SIP Over WebSockets Connector -->
<Connector port="5082"
    ipAddress = "127.0.0.1"
    protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"
    signalingTransport="ws"/>

```

After

```

<!-- Define a SIP Connector on port 5080 -->
<Connector port="5080"
    ipAddress = "TeleStaxAS"
    hostNames = "TeleStaxAS"
    protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"
    signalingTransport="udp"/>

<!-- Define the default TCP SIP Connector -->
<Connector port="5080"
    ipAddress = "TeleStaxAS"
    hostNames = "TeleStaxAS"
    protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"
    signalingTransport="tcp"/>

<!-- Define the default TLS SIP Connector -->
<Connector port="5081"
    ipAddress = "TeleStaxAS"
    hostNames = "TeleStaxAS"
    protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"
    signalingTransport="tls"/>

<!-- Define the default SIP Over WebSockets Connector -->
<Connector port="5082"
    ipAddress = "TeleStaxAS"
    hostNames = "TeleStaxAS"
    protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"
    signalingTransport="ws"/>

```

If the Application Server is configured in a cloud where it has an internal (private) IP address and externally bound IP address, the following needs to be configured in order for the Application Server to use an external IP address instead of a private one:

```
staticServerAddress="xxx.xxx.xxx.xxx"  
staticServerPort="xxxx"  
useStaticAddress="true"
```

Example

```
<!-- Define a SIP Connector on port 5080 -->  
<Connector port="5080"  
  ipAddress = "TelScaleApacheTomcat"  
  protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"  
  signalingTransport="udp"  
  staticServerAddress="xxx.xxx.xxx.xxx"  
  staticServerPort="5080"  
  useStaticAddress="true"/>  
  
<!-- Define the default TCP SIP Connector -->  
<Connector port="5080"  
  ipAddress = "TelScaleApacheTomcat"  
  protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"  
  signalingTransport="tcp"  
  staticServerAddress="xxx.xxx.xxx.xxx"  
  staticServerPort="5080"  
  useStaticAddress="true"/>  
  
<!-- Define the default TLS SIP Connector -->  
<Connector port="5081"  
  ipAddress = "TelScaleApacheTomcat"  
  protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"  
  signalingTransport="tls"  
  staticServerAddress="xxx.xxx.xxx.xxx"  
  staticServerPort="5081"  
  useStaticAddress="true"/>  
  
<!-- Define the default SIP Over WebSockets Connector -->  
<Connector port="5082"  
  ipAddress = "TelScaleApacheTomcat"  
  protocol="org.mobicens.servlet.sip.startup.SipProtocolHandler"  
  signalingTransport="ws"  
  staticServerAddress="xxx.xxx.xxx.xxx"  
  staticServerPort="5082"  
  useStaticAddress="true"/>
```

TeleStax Startup

To run the Application Server, go to the following directory:

```
${CATALINA_HOME}/bin
```

Then, execute the following command:

```
./catalina.sh run
```

```
=====
==
==      Thank you for running Mobicents Community code      ==
== For Commercial Grade Support, please request a TelScale Subscription ==
==      http://www.telestax.com/                             ==
==
=====

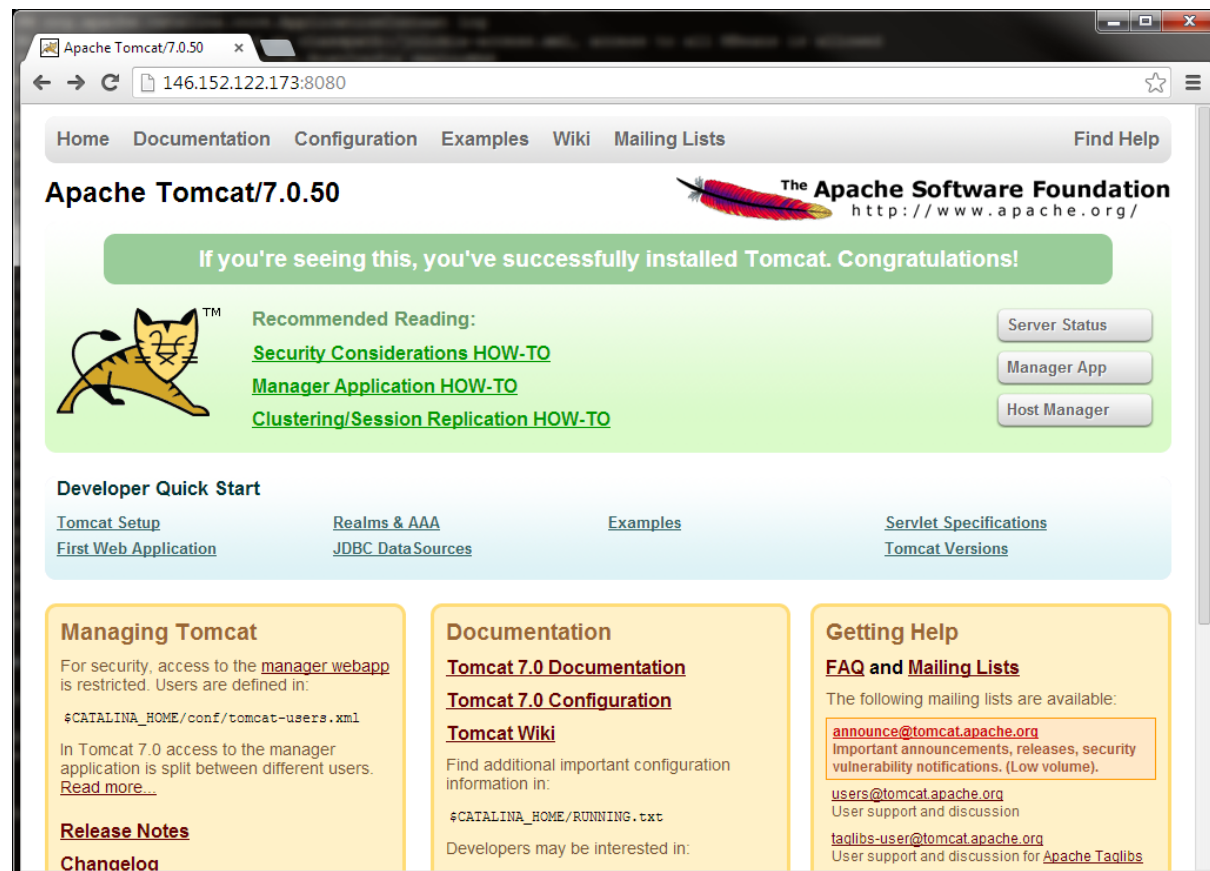
2014-04-22 11:59:44,936 WARN [SipStackImpl] (main) Could not register the stack as a Notification L
istener of jboss.system:service=Logging,type=Log4jService runtime changes to log4j.xml won't affect
SIP Stack Logging
Apr 22, 2014 11:59:44 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 23281 ms
```

To stop the service, press **Ctrl+C**.

TeleStax Verification

Once the Application Server service is started, access the TeleStax Apache-Tomcat Web Administration from any browser by going to the following URL:

```
http://<as_ip_address>:8080
```



With the newly installed Application Server, there will be no access to any of the configuration sections. The following steps need to be taken to get access:

1. Edit the *tomcat-users.xml* file:

```
${CATALINA_HOME}/conf/tomcat-users.xml
```

2. Modify the file like the example modifications shown below in **RED**:

```
<!--  
    NOTE:  The sample user and role entries below are wrapped in a comment  
    and thus are ignored when reading this file. Do not forget to remove  
    <!-- ... --> that surrounds them.  
-->  
  
<role rolename="manager-gui"/>  
  
<user username="admin" password="admin" roles="manager-gui"/>  
  
<!--  
    <role rolename="tomcat"/>  
    <role rolename="role1"/>  
-->
```

3. Access any configuration screen using the new credentials (for example, admin/admin as an in example above).

8. Appendix B: Updating the Dialogic JSR 309 Connector

The Dialogic JSR 309 Connector is a set of the three files as described by the following section: [Dialogic JSR 309 Connector Requirements](#).

In the TeleStax Application Server ApacheTomcat version, previous versions of connector files need to be removed and replaced with new versions. They are found here:

```
<application_WAR_File>/WEB-INF/lib/dialogic309-#.#.####-tomcat#.jar
```

```
<application_WAR_File>/WEB-INF/lib/dialogic309msmltypes-#.#.####-tomcat#.jar
```

The older version of the smiltypes JAR file needs to be removed and replaced with new a new version in platform's *lib* directory:

```
${CATALINA_HOME}/lib/dialogic309smiltypes-#.#.####-tomcat#.jar
```