How to debug Denodo custom extensions with Eclipse

Revision 20210630

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

1

2 GOAL

The aim of this document is to provide guidance and advice on how to debug Denodo custom extensions using the Eclipse IDE.

3 DEBUGGING CUSTOM EXTENSIONS

Denodo provides a series of APIs that allow users to develop their own Denodo extensions using Java code. This increases the flexibility of Denodo allowing the user to implement their own business logic or custom connectors to not supported sources.

These APIs allow:

- Creating VDP custom functions, stored procedures or custom wrappers. The Virtual DataPort Developer Guide section of the documentation describes the steps to create these artifacts.
- Creating Scheduler custom exporters or custom handlers. Refer to Scheduler Developer API to find further information about Scheduler extensions.
- Developing ITPilot custom functions. Further information can be found in the ITPilot Developer Guide.

When developing any code, having debugging capability is one of the most important tools to create code and find errors as it allows the developer to check how the code is working step by step.

So the question is, how can a Denodo custom extension be debugged? We have two alternatives.

First, Denodo provides within its installation a component called Denodo4E which is an Eclipse plugin that provides a framework for working with Denodo extensions.

NOTE: The Denodo4E plugin is scheduled to be deprecated in future releases of Denodo 8.0

This component is able to:

- Create any custom extension. When creating a new Denodo extension, the plugin will automatically create an empty project with a simple class ready to be deployed.
- Start the Denodo server in debug mode.
- Package and deploy your extensions.
- Debug the custom extension code.

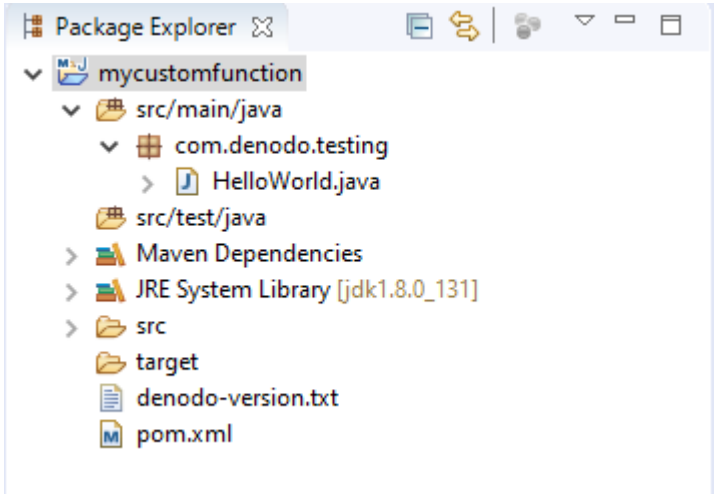
This Eclipse plugin creates a “Denodo project” with its own structure and configuration. Sometimes, due to internal requirements, the custom extension project must have a specific structure or must use specific frameworks. For instance, the project must be created using the Apache Maven project management tool. All these “non Denodo” projects can also be debugged using a second alternative.

3.1 DEBUG ECLIPSE PROJECTS WITHOUT USING DENODO4E

There are a few steps to enable a Denodo server to be used for debugging.Let’s go through that with an example.

We are going to create a VDP custom function to return a “Hello World” message using a Maven project in Eclipse.

First, we have created our own Maven project called “`mycustomfunction`”.



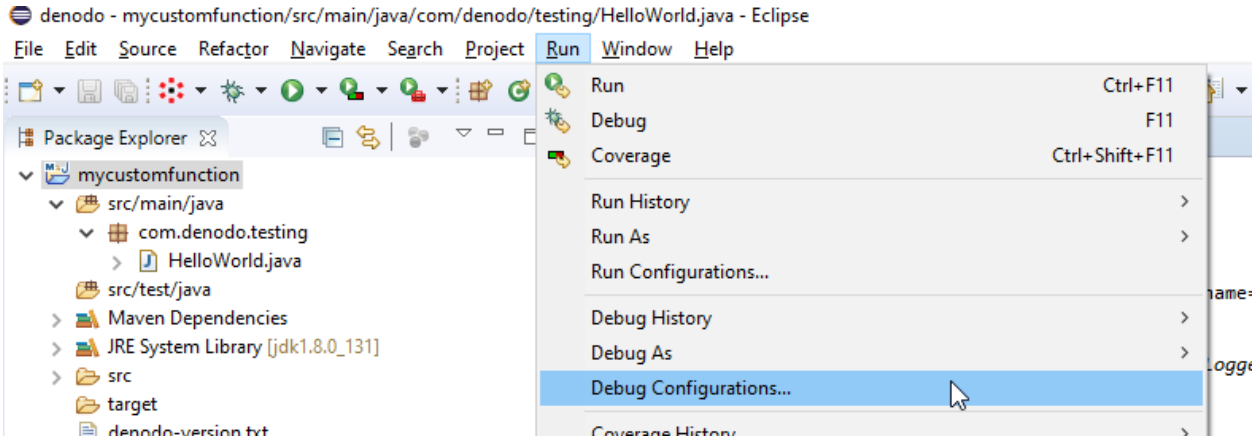
Keep in mind that our HelloWorld.java class has been created according to the Developing Custom Functions section of the Denodo documentation. We will not focus much on the code as this is not the goal of this article.

The code of our Hello World function is pretty simple though.

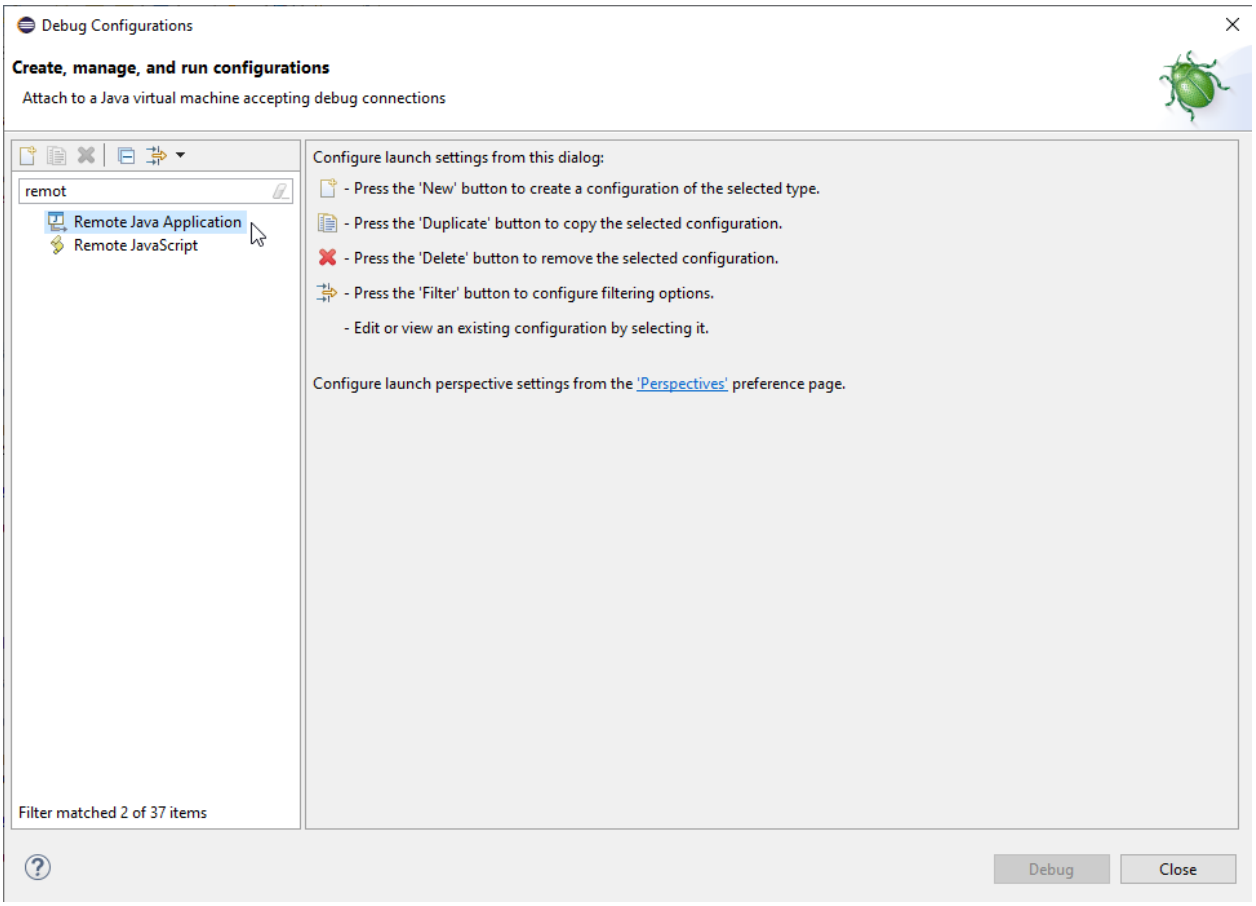
```
package com.denodo.testing;

import org.apache.log4j.Logger;

import com.denodo.common.custom.annotations.CustomElement;
import com.denodo.common.custom.annotations.CustomElementType;
import com.denodo.common.custom.annotations.CustomExecutor;
import com.denodo.common.custom.annotations.CustomExecutorReturnType;


@CustomElement(type=CustomElementType.VDPFUNCTION, name="helloworld")
public class HelloWorld {

                private    static    final    Logger    logger    =
Logger.getLogger(HelloWorld.class);


    /**
     * This method is invoked when the custom function is executed
     *
     * @param input parameters
     *
     * @return custom value
     */
    @CustomExecutor
```

```
        public String execute () {

            logger.debug("Executing Custom Function");

            return "Hello World";
        }

        /**
            * This method is invoked to compute the return type before
    executing the function.
             * If the execute method returns a simple Java type, this method
    is not needed.
             *
             * @param type for execute method input parameters
             *
             * @return custom type that execute method will return
             */
        @CustomExecutorReturnType
        public Class<String> executeReturnType () {

            logger.debug("Getting Custom Function return type");

            return String.class;

        }
    }
```

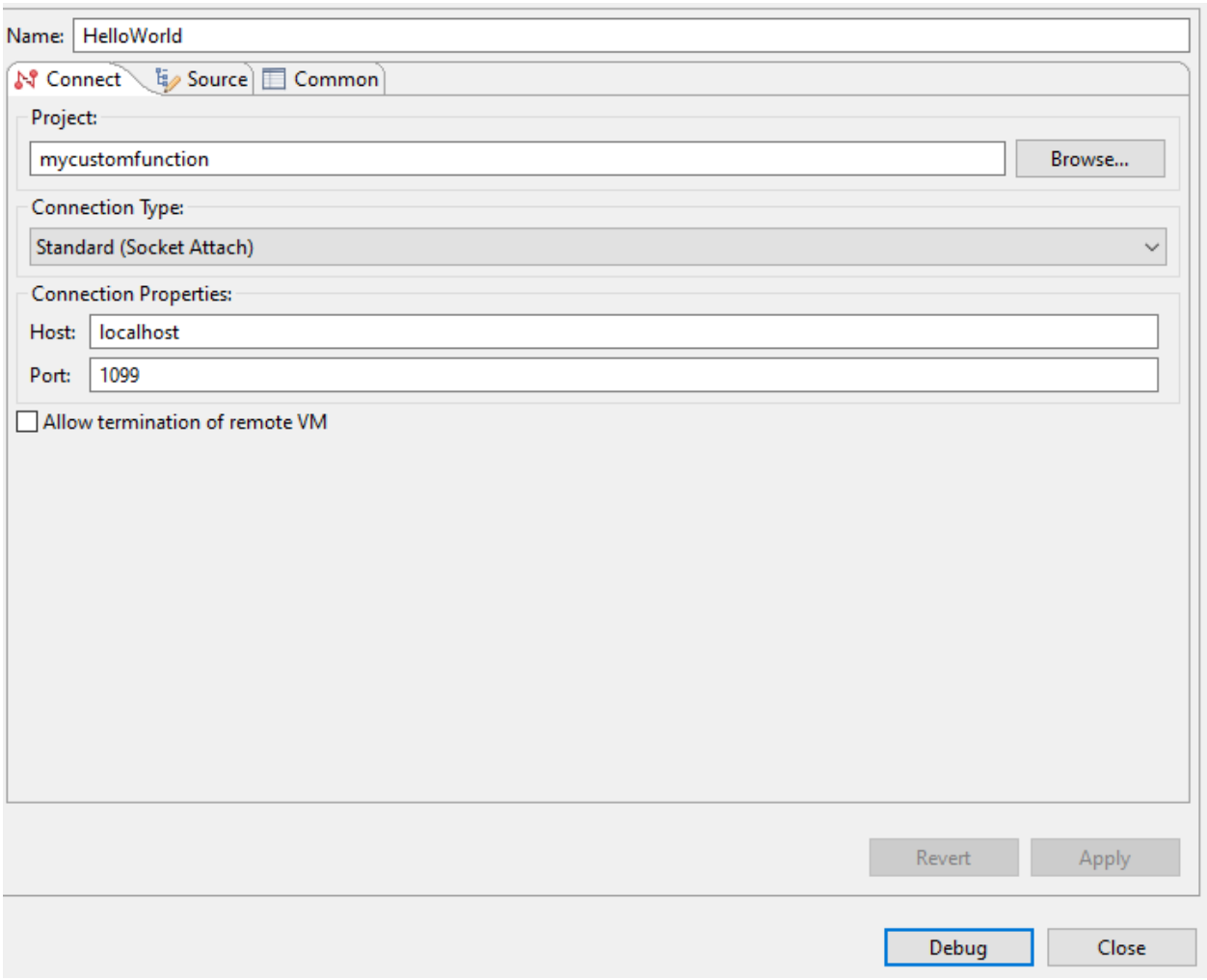
In order to prepare Eclipse for debugging go to `Run -> Debug Configurations`



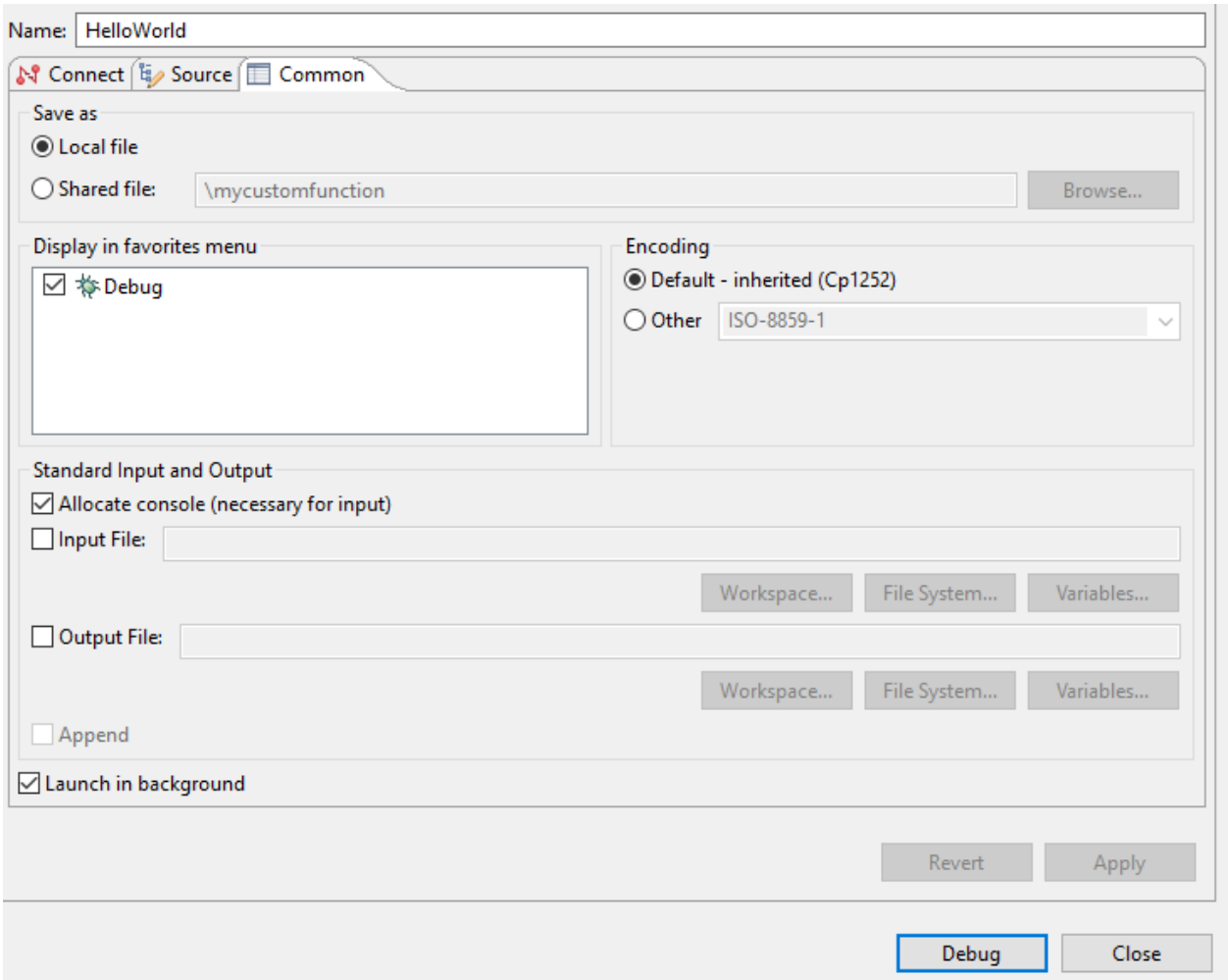
Create a new **Remote Java Application** configuration



Configure the remote application as follows:



Note that the host must be the one where the VDP server is running and the port (1099) will be enabled in the VDP server later. Any other available port can be configured on both ends.

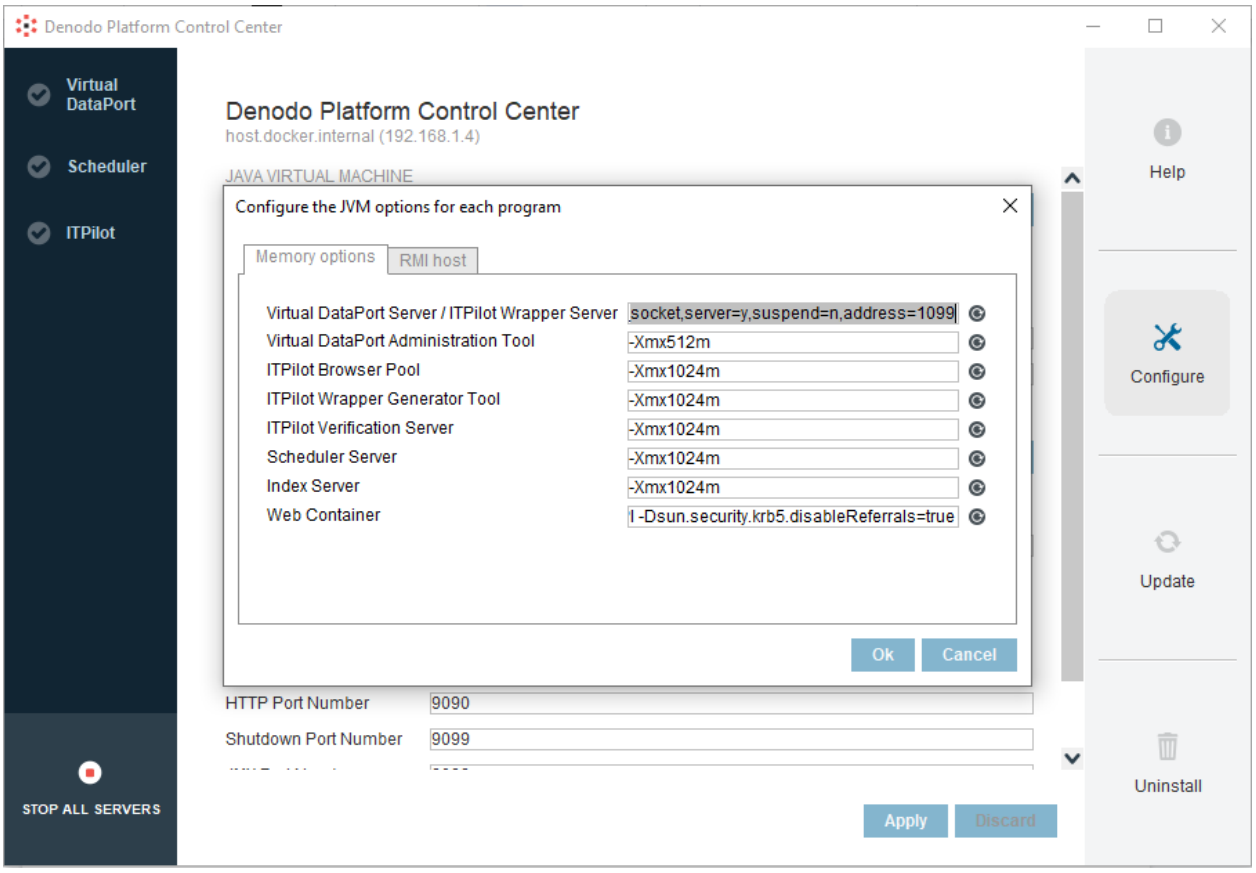


Then click Apply, then Close.

Now it is time to configure the VDP server to enable communications from Eclipse using the port 1099 specified above.

In the Denodo Control Center go to Configure -> JVM Options and append the following settings in the Virtual DataPort Server / ITPilot Wrapper Server option.

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=0.0.0.0:1099
```



Click on “Ok” and “Apply” to persist the new settings, then restart the server.

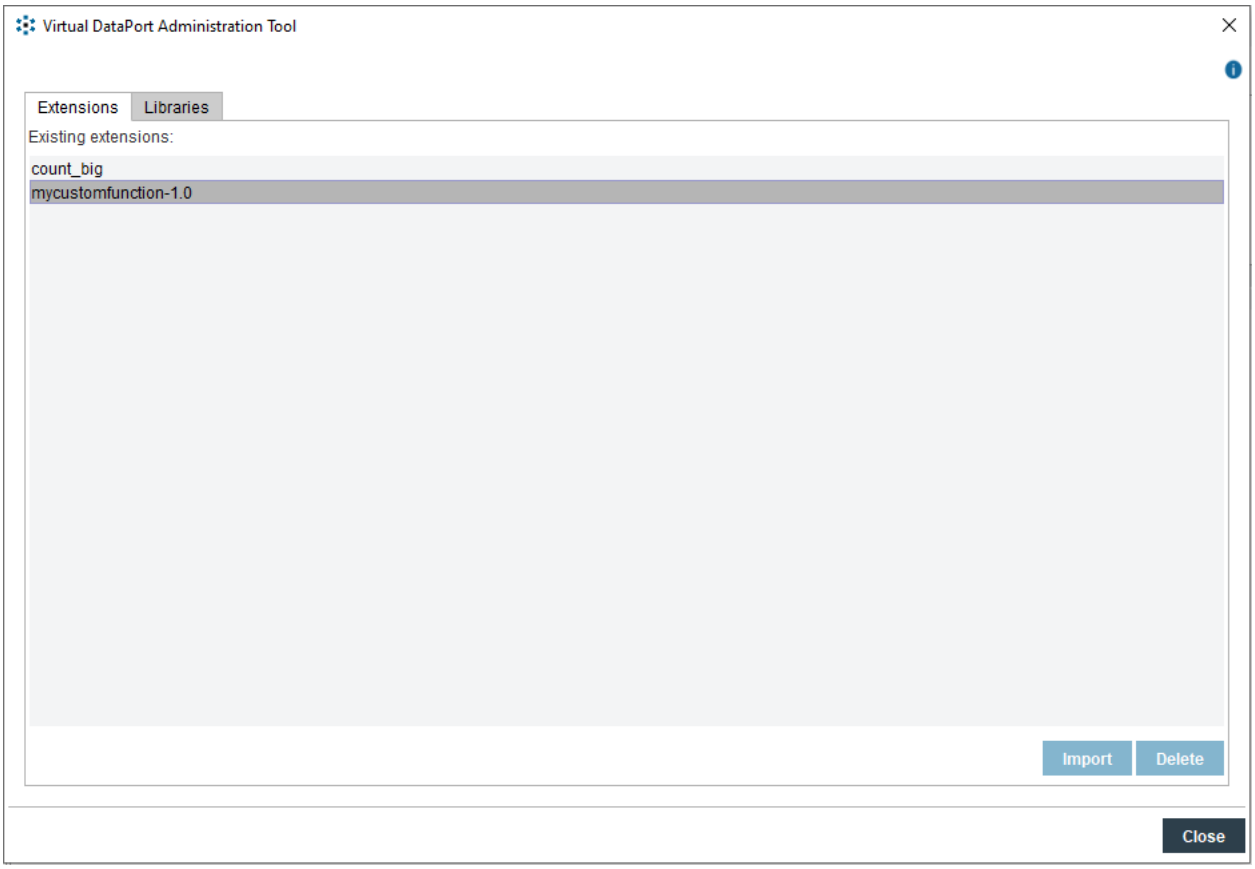
If using a headless installation and the Denodo Control Center is not available, include the aforementioned settings following these steps.

1. Open the `<DENODO_HOME>/conf/vdp/VDBConfiguration.properties` file.
2. Find the `java.env.DENODO_OPTS_START` property and append the settings `-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=0.0.0.0:1099` to the current values.

3. Save the file.
4. For the changes to take effect run the `<DENODO_HOME>/bin/regenerateFiles.(bat/sh)`script.
5. Restart the VDP server.

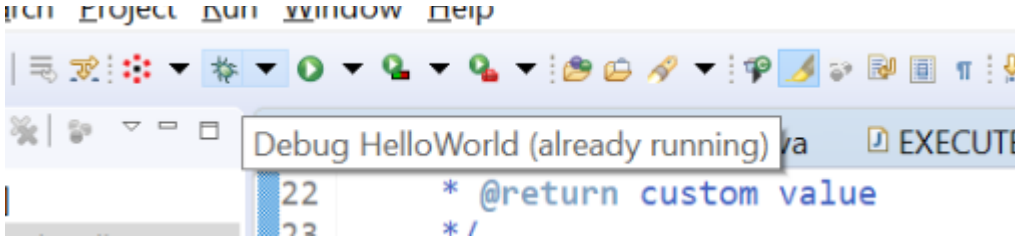
NOTE: Adding this parameter to the JVM may have effects in the performance of the JVM. It is not recommended to modify the configuration in this way in production environments.

Finally, create the package of the Eclipse project ((i.e.) export as a JAR file) and load it in the VDP server using the VDP Admin Tool as described in the Importing Extensions section of the Denodo documentation.

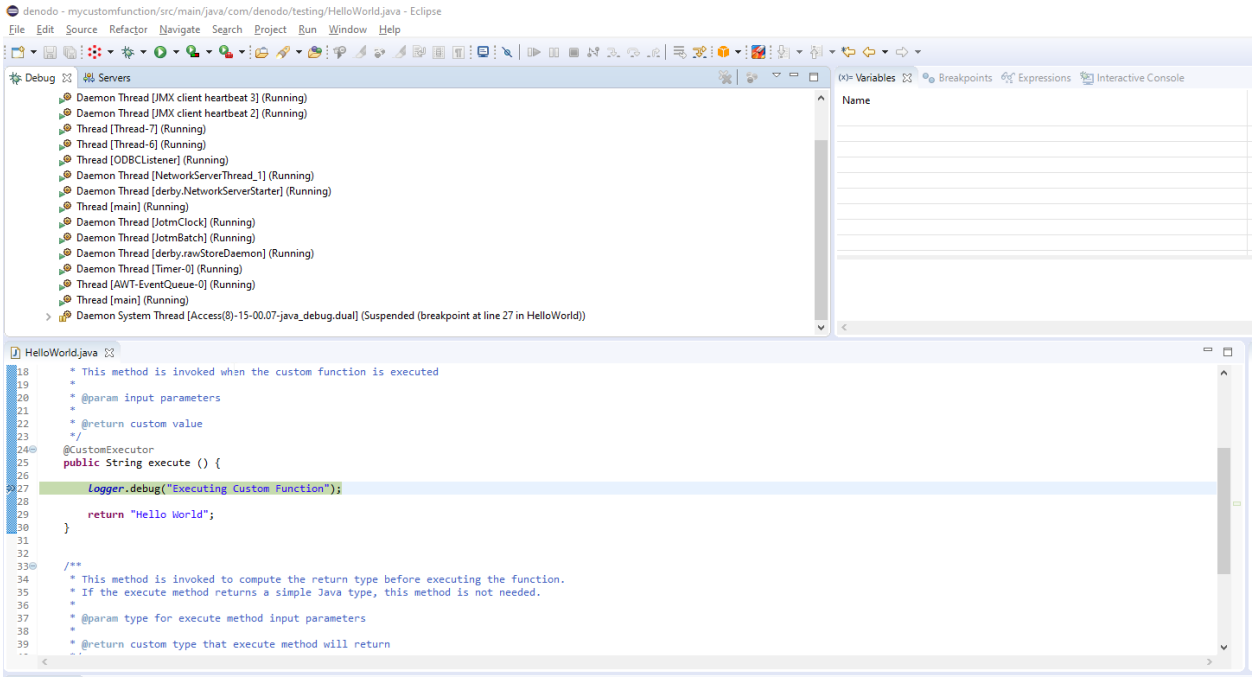


Once this is done, back in eclipse, run the debug configuration. If you get a connection refused error, then remake the configuration, and try running it again.

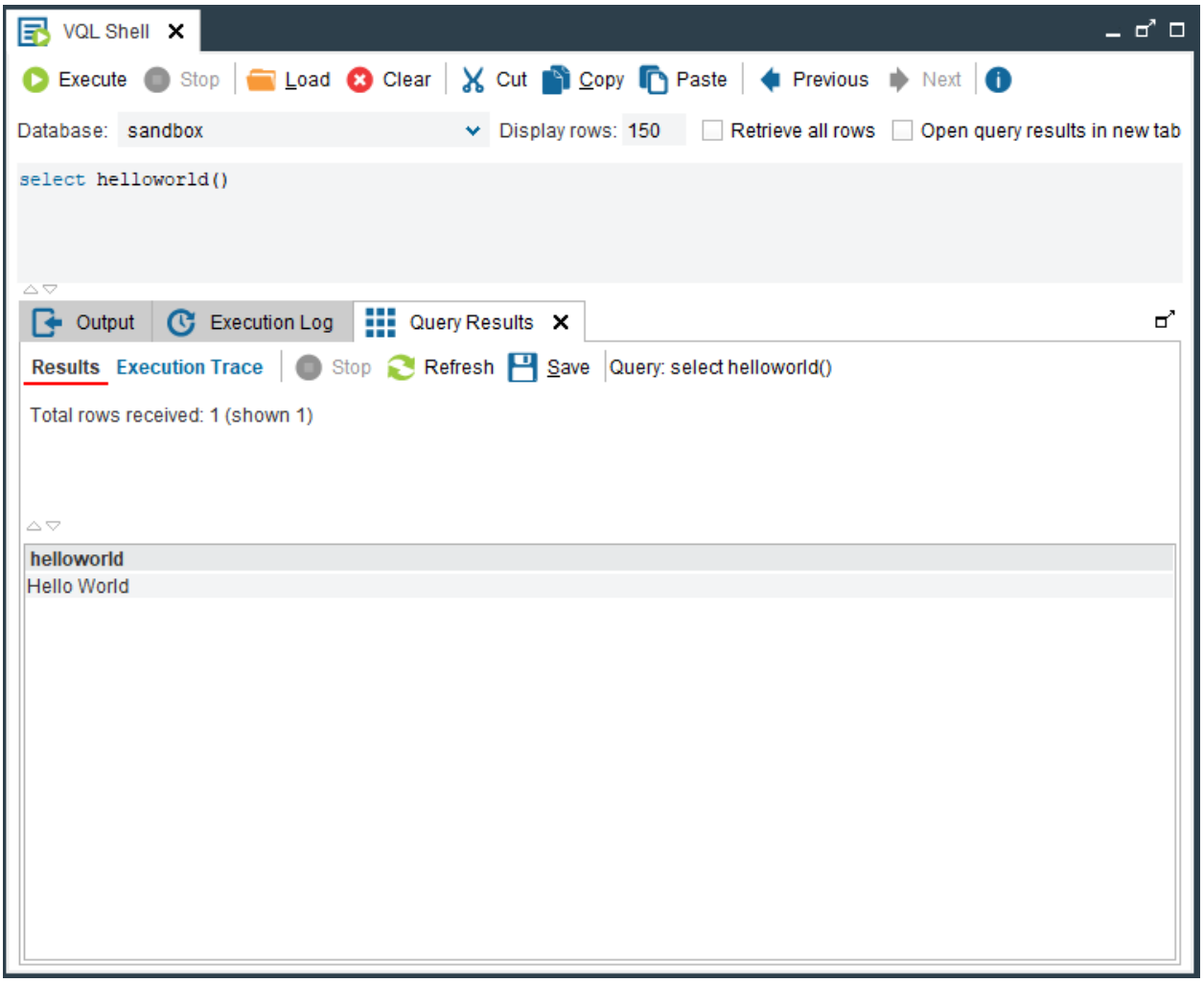
Note: If the Debug configuration is already running then you will get a connection refused error. To find if Debug configuration is already running, hover over the Debug icon. A sample screenshot below



If there are no errors then the connection to the VDP server is established. Now just include a breakpoint in Eclipse and run the custom extension from the VDP Admin tool , and Eclipse will catch the breakpoint.



And the final result of our “Hello World” function is...



References

[Virtual DataPort Developer Guide]
[Scheduler Developer API]
[ITPilot Developer Guide]
[Denodo4E User Guide]
[Importing Extensions]