Predix
Machine services

Step by step lab instructions

Version 10

Predix™

© 2015 General Electric Company.

GE, the GE Monogram, and Predix are either registered trademarks or trademarks of General Electric Company. All other trademarks are the property of their respective owners.

This document may contain Confidential/Proprietary information of GE, GE Global Research, GE Software, and/or its suppliers or vendors. Distribution or reproduction is prohibited without permission.

THIS DOCUMENT AND ITS CONTENTS ARE PROVIDED "AS IS," WITH NO REPRESENTATION OR WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF DESIGN, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. ALL OTHER LIABILITY ARISING FROM RELIANCE UPON ANY INFORMATION CONTAINED HEREIN IS EXPRESSLY DISCLAIMED.

Access to and use of the software described in this document is conditioned on acceptance of the End User License Agreement and compliance with its terms.

This page is intentionally left blank

Table of Contents

About this guide 5

Lab 1: Set up the development environment 6
Duration of lab 6
Objectives 6
Step 1: Locate and update the configuration files 6
Step 2: Configure the Java variables to use OpenJDK (external DevBox ONLY) 7
Step 3: Download Predix SDK 8
Step 4: Enable STS to refresh templates from the Artifactory 8

Lab 2: Using Predix SDK to Generate a Machine 10
Duration of lab 10
Objectives 10
Step 1: Examine and import Predix SDK 10
Step 2: Create a Debug Machine Container 14
Step 3: Export the Debug Machine Container 18
Step 4: Start the Debug Container 20
Step 5: Test Administrator Command Line options 20
Step 6: Start Administrator Web Console 21
Step 7: Test Administrator Web Console options 23

Lab 3: Create a basic consumer bundle 24
Duration of lab 24
Objectives 24
Step 1: Create a new project from archetype 24
Step 2: Examine the project files 26
Step 3: Build and deploy the project 26

Lab 4: Consuming another service:Adding Security to the Consumer Bundle 30
Duration of lab 30
Objectives 30
Step 1: Create a configuration file for the consumer bundle 30
Step 2: Add security package to POM.xml file 31
Step 3: Create a variable for the configuration property 31
Step 4: Add necessary code for consuming a new service 32
Step 5: Implement security APIs 32
Step 6: Build the project and deploy the bundle 33

Lab 5: Generate Gateway Data with Machine Adapter 34
Duration of lab 34
Objectives 34
Step 1: Configure the consumer bundle 34
Step 2: Add necessary code for consuming a new service 35
Step 3: Implement the IMachineAdapter interface 35
Step 4: Retrieve the adapter through the Machine Gateway API 36

Lab 6: Data in Motion: Generate and capture simulated data (DDS Data River) 38
Duration of lab 38

Objectives 38
Step 1: Download and start the Modbus PLC simulator *38*
Step 2: Configure the Modbus adapter in Machine services *42*
Step 3: Configure a spillway for DDS *43*
Step 4: Begin collecting "data forward" data *43*
Step 5: Configure the Data River Sender and Receiver service *44*
Step 6: Restart the Machine and Forward Stored Data *44*

Lab 7: Data in Motion: Connect to the Cloud with HTTP Data River and HTTP Service 47
Duration of lab 47
Objectives 47
Step 1: Cloud Foundry Setup *47*
Step 2: Import HTTP Data Service Sample from Predix SDK *48*
Step 3: Configure Cloud Foundry Credentials *49*
Step 4: Configure spillway for HTTP *49*
Step 5: Configure HTTP River *49*
Step 6: Restart Predix Machine *50*
Step 7: Stop ModbusPal and shut down machine *50*

Troubleshooting 51
Troubleshooting permissions 51

About this guide

This guide is intended for anyone taking the Machine services training course after attending the Predix Fundamentals training. It is the hands on component that demonstrates how Predix Machine works. By giving learners first hand experience developing for an edge device, you will have an understanding of what it takes to create solutions for the data-driven gateway of Predix.

In order to be successful in the labs it is essential to complete the installation before the class. This includes:

- Request a Cloud Foundry account
- Request a Github account
- Set up the most updated DevBox version

This guide is divided into various labs that help you experience the basic steps needed for Predix Machine. Each lab outlines:

- Overall goal of the lab
- What the learner would be able to do
- Approximate time needed to complete the lab
- Step by step "how-to" guidance for the lab
- Common errors and exceptions encountered, if any
- Troubleshooting tips and tricks as applicable

Lab 1: Set up the development environment

Duration of lab

~ 20 minutes

Objectives

Ensure your development environment is configured correctly to complete each of the labs.

Step 1: Locate and update the configuration files

1. Locate the ~**/.m2/settings.xml** file. Update the <servers> and <profile> settings as follows:

```
<server>
        <id>predix.repo</id>
        <username>cloud foundry login</username>
        <password>cloud foundry password</password>
</server>
```

where cloud foundry login is student1 through student74 and password is chang3m3.

```
<profile>
   <repositories>

     <repository>
          <id>predix.repo</id>
          <url>https://artifactory.predix.io/artifactory/PREDIX-
EXT</url>
          <snapshots>
               <enabled>false</enabled>
          </snapshots>
          <releases>
               <enabled>true</enabled>
          </releases>
     </repository>
```

2. Locate the ~**/.m2/archetype-catalog.xml**. Add the following archetype (if not already there):

```
<archetype>
    <groupId>com.ge.predix.solsvc</groupId>
    <artifactId>machine-training-archetype</artifactId>
    <version>15.3.0</version>
</archetype>
```

Step 2: Configure the Java variables to use OpenJDK (external DevBox ONLY)

1. From a terminal window go to **/etc** directory and locate the **bashrc** file. You need to edit this file as system admin in order to update.

   ```
   $ su vi bashrc
   ```

   When prompted, password is "predix".

2. Locate the export $JAVA_HOME statement at the bottom of the file (under ## PREDIX DEVELOPERS VARS). Comment out the existing export and add the following new path

   ```
   #export JAVA_HOME=/usr/java/jdk1.8.0_40/jre
   export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk.x86_64
   ```

3. Update the PATH variable just below this to include the JAVA_HOME variable

   ```
   export PATH=$HOME/bin:$PATH:$JAVA_HOME/bin
   ```

4. Save the file and restart DevBox
5. Once DevBox is up and running, verify $JAVA_HOME is pointing to the correct location. Type the following commands in a Terminal window

at the command line.

```
$ echo $JAVA_HOME
/usr/lib/jvm/java-1.8.0-openjdk.x86_64

$ which keytool
/usr/lib/jvm/java-1.8.0-openjdk.x86_64/bin/keytool
```

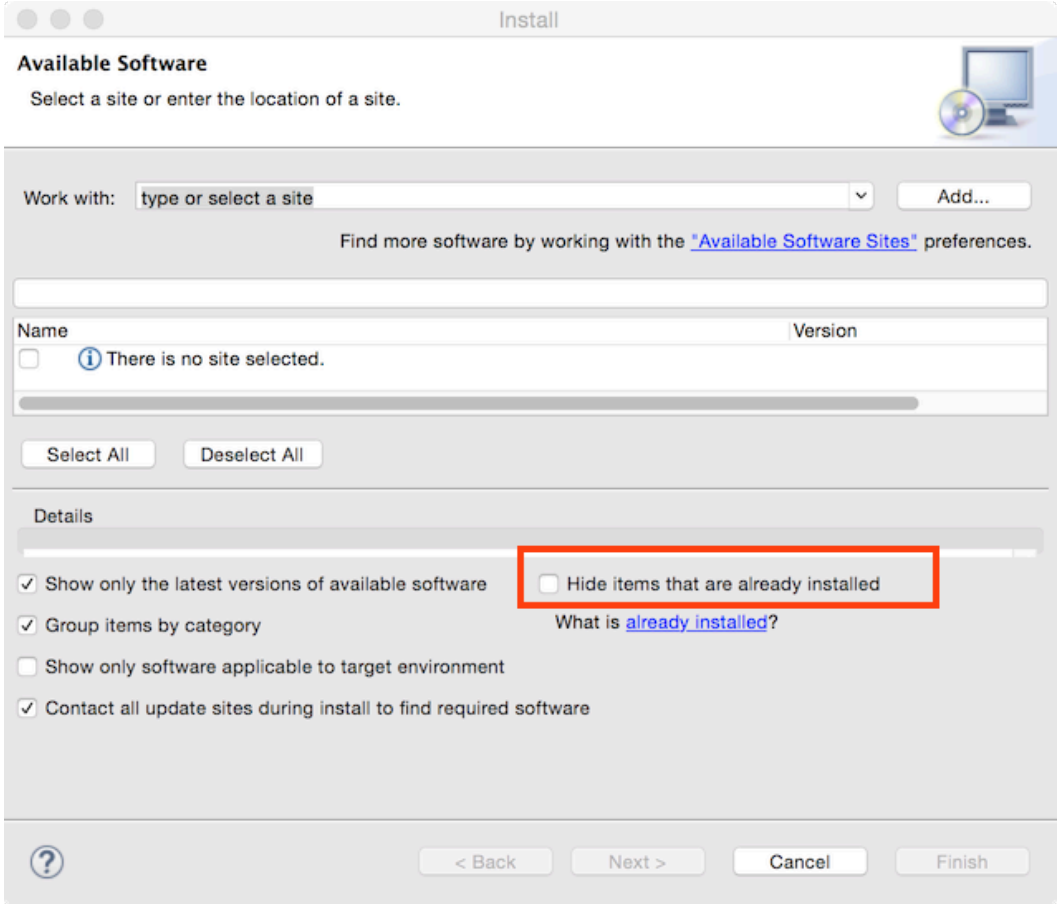
Step 3: Download Predix SDK

1. Go to https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/predixsdk/15.3.0/predixsdk-15.3.0.zip in a browser window. You will be prompted for a login. Enter your cloud foundry login credentials (studentxx | chang3m3).
2. When finished, copy the ZIP file from the download directory to the /predix/PredixApps folder.
3. Open a terminal window and go to predix/PredixApps.
4. Unzip the contents from the command line

```
$ unzip predixsdk-15.3.0.zip
```

Step 4: Enable STS to refresh templates from the Artifactory

1. From the desktop open Spring Tool Suite. Go to Help -> Install New Software. Uncheck the “Hide items that are already installed”.


Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
Name
Version
There is no site selected.
Select All
Deselect All
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Cancel
Finish

Lab 2: Using Predix SDK to Generate a Machine

Duration of lab

~ 25 minutes

Objectives

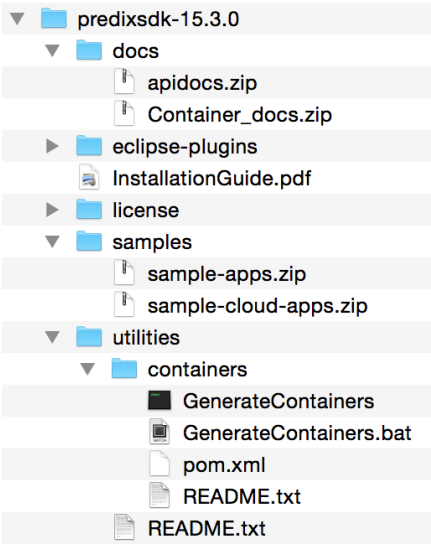
The overall goal of this lab is to familiarize you with the Predix SDK and Predix OSGI Containers. At the end of the lab, you will be able to:

- Explore and import Predix SDK
- Create and navigate a machine OSGI container
- Configure the web console to administer bundles in the container

Step 1: Examine and import Predix SDK

The Predix SDK contains all the scripts and documentation to generate an OSGI container as well as bundles that run inside the container.

3. Locate the Predix SDK in the predix/PredixApps folder (predixsdk-15.3.0)
4. Open the folder and examine the files. Unzip the apidocs.zip.



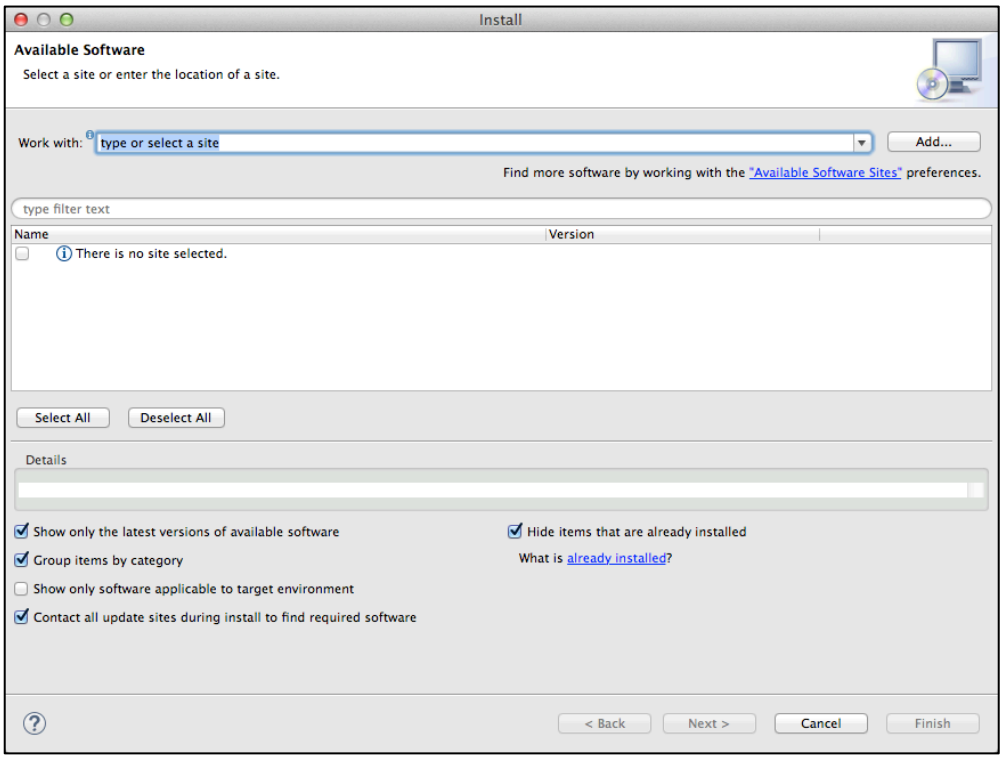
/docs

- Javadoc APIs for Predix machine services
- Prosyst docs for OSGi containers

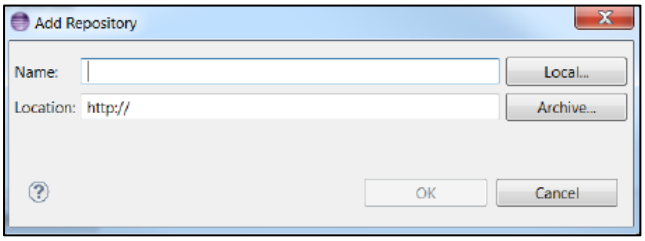
/eclipse-plugins

- Import into your Eclipse-based tool (STS)

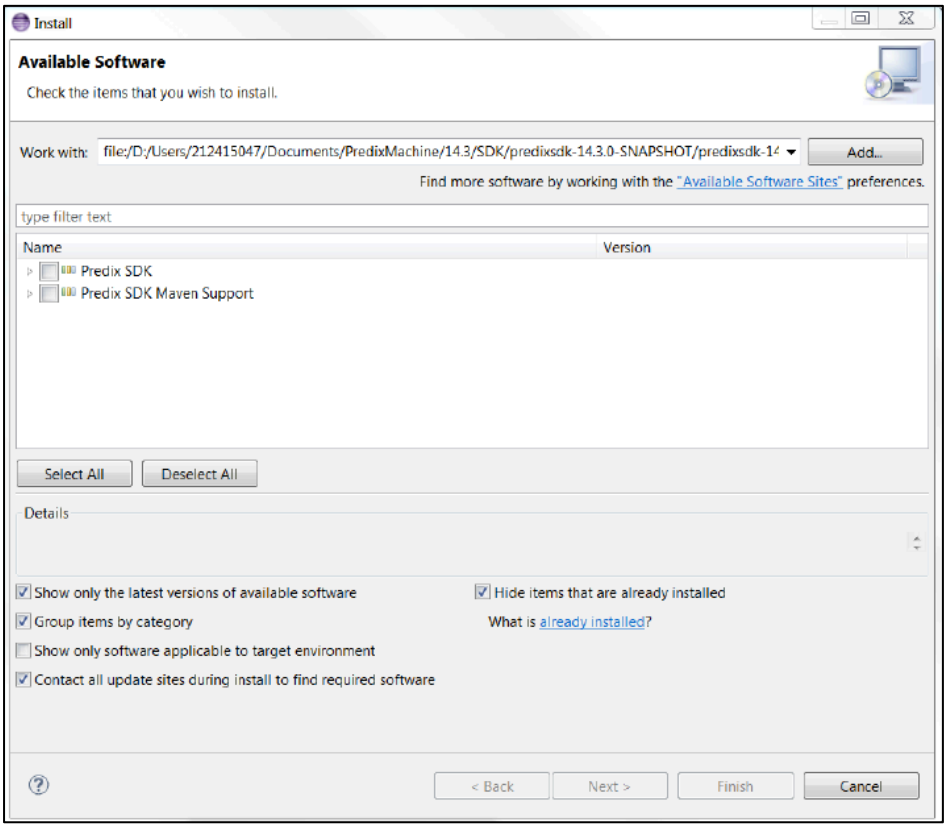
1. Open Eclipse by double clicking the STS icon on the desktop. Import the SDK by clicking on **Help -> Install New Software**. You should see the following screen:



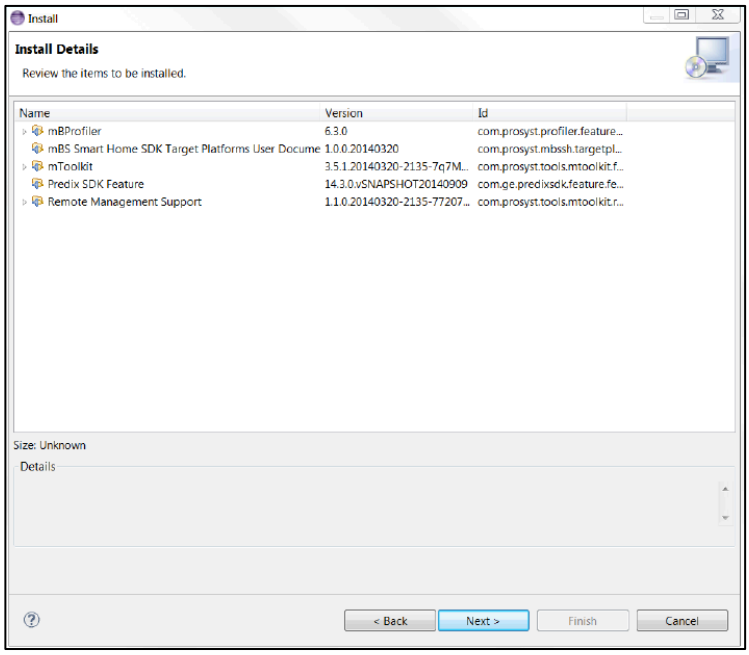
2. Click the Add button. It takes you to the Add Repository window:



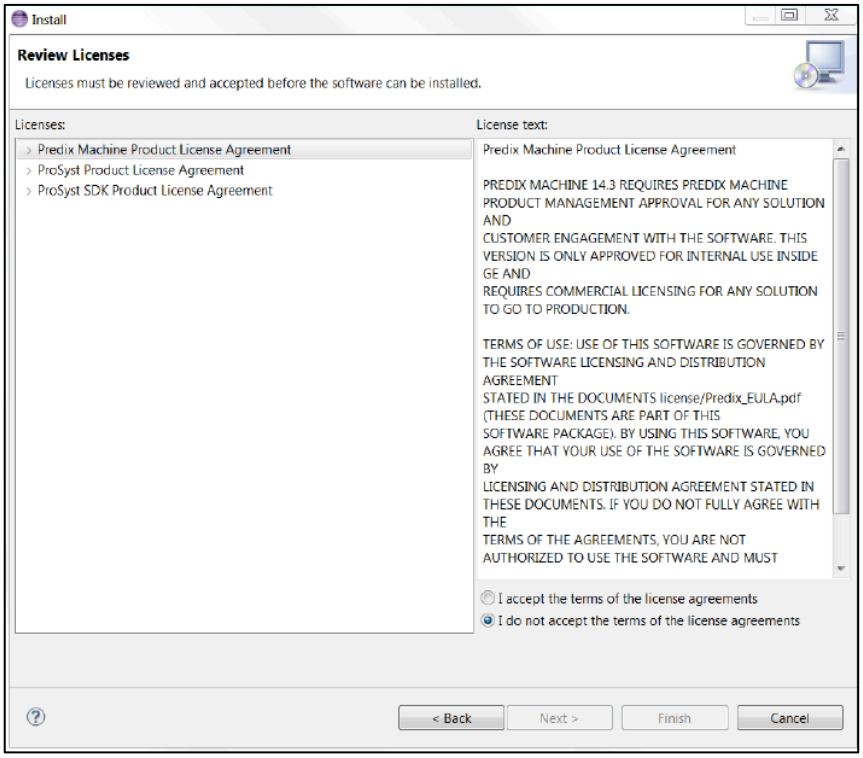
3. Click the **Local** button and browse to the folder where the Predix SDK files are. Select the **eclipse-plugins** folder. The Predix SDK and Predix SDK Maven Support options now appear in the list of available software:



4. Click the **Select All** button to install the SDK and Maven support. Click **Next.**

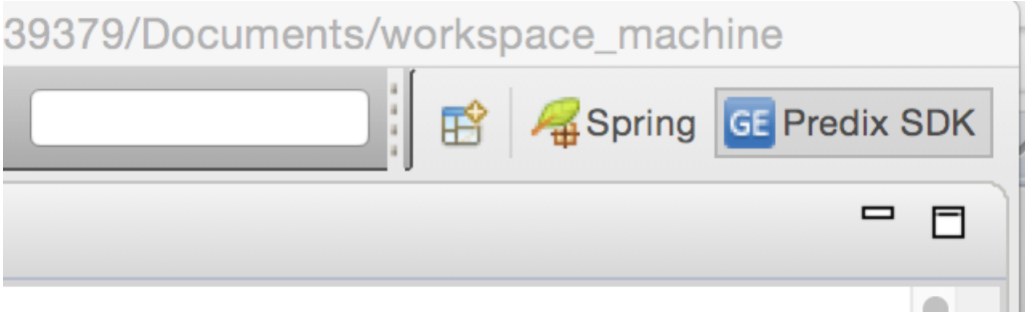


5. Click **Next** on the **Install Details** window to be taken to the **Review Licenses** window.



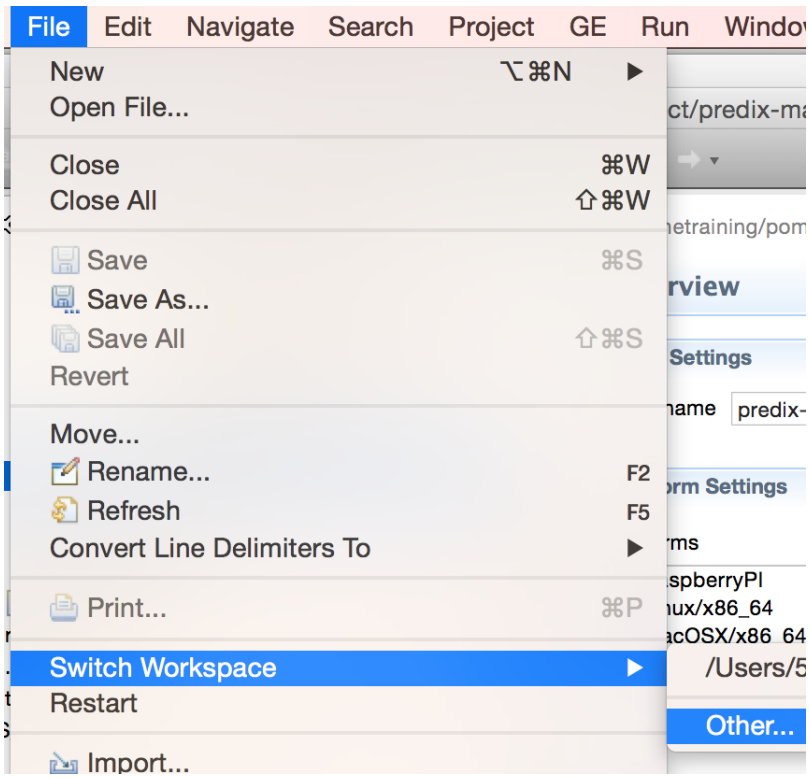
6. Accept the license and click **Finish.**

   If you get a security warning, ignore and click OK. Eclipse should install the Predix SDK and prompt for a restart. Confirm this prompt to relaunch Eclipse. When Eclipse reloads, it will ask if you want to switch to the Predix SDK target platform. Select **Switch** to complete the installation of the Predix SDK. When Eclipse opens it will default to the Predix SDK perspective.



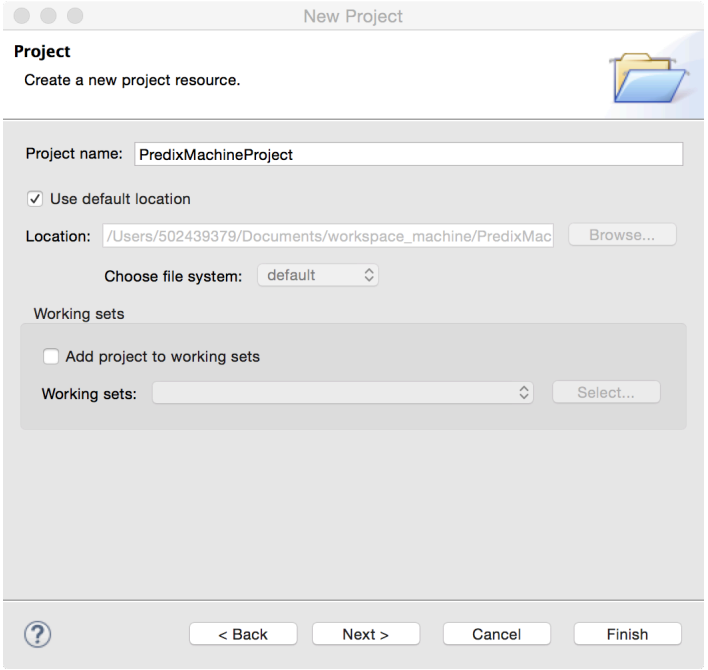
Step 2: Create a Debug Machine Container

1. Change workspace to the machine workspace. Select **File → Switch Workspace → Other**



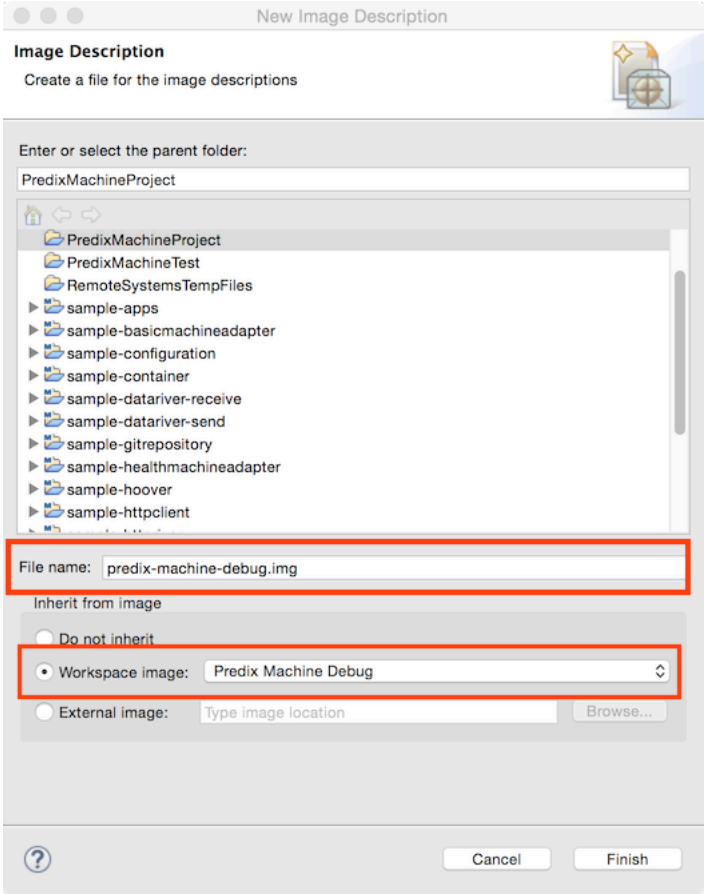
Click browse and locate **/predix/PredixApps/training/machine/workspace**. This will restart Eclipse. All user-generated machine projects will now be saved to this directory.

2. Ensure you are in the Predix SDK perspective (select the GE Predix SDK tab in the upper right corner of the tool). This gives you access to all the templates and modules in the Predix SDK when creating new projects.
3. Create a New Project called PredixMachineProject that will hold the machine container image. Click on **File → New → Other → General → Project**.

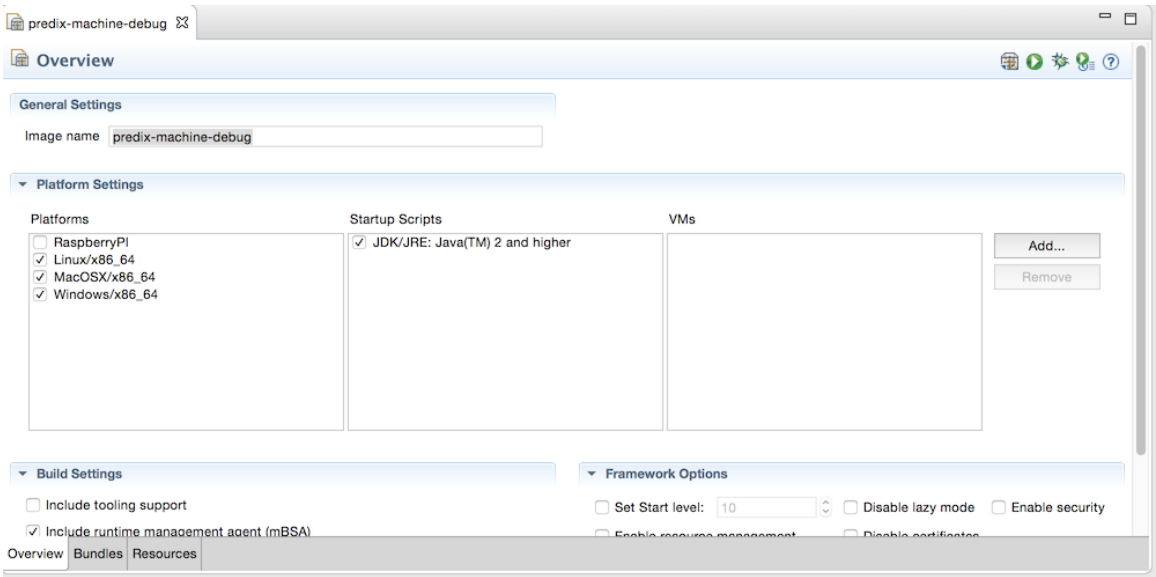


Click finish.

4. Create a container image to use for the remainder of this training. Select the PredixMachineProject and click on **File -> New -> Image Description**.
5. Type “predix-machine-debug” in the File name text box. Use the “Predix Machine Debug” as the workspace image in the Inherit Image section

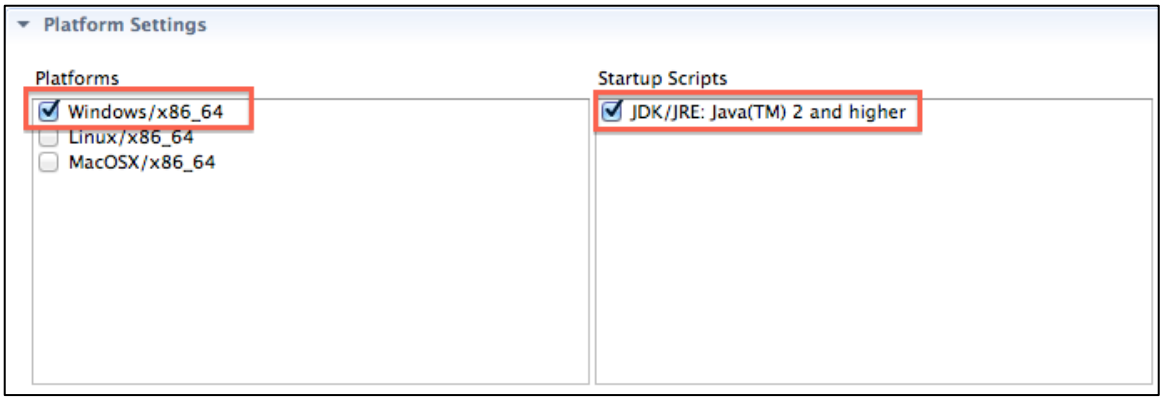


6. Click Finish. A new tab called “predix-machine-debug” will open in the editor window.

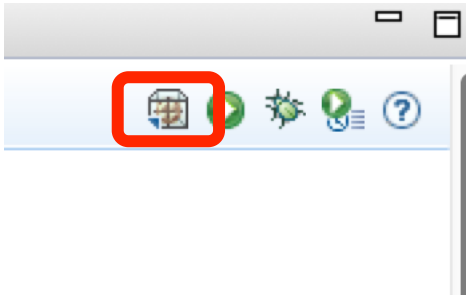


Step 3: Export the Debug Machine Container

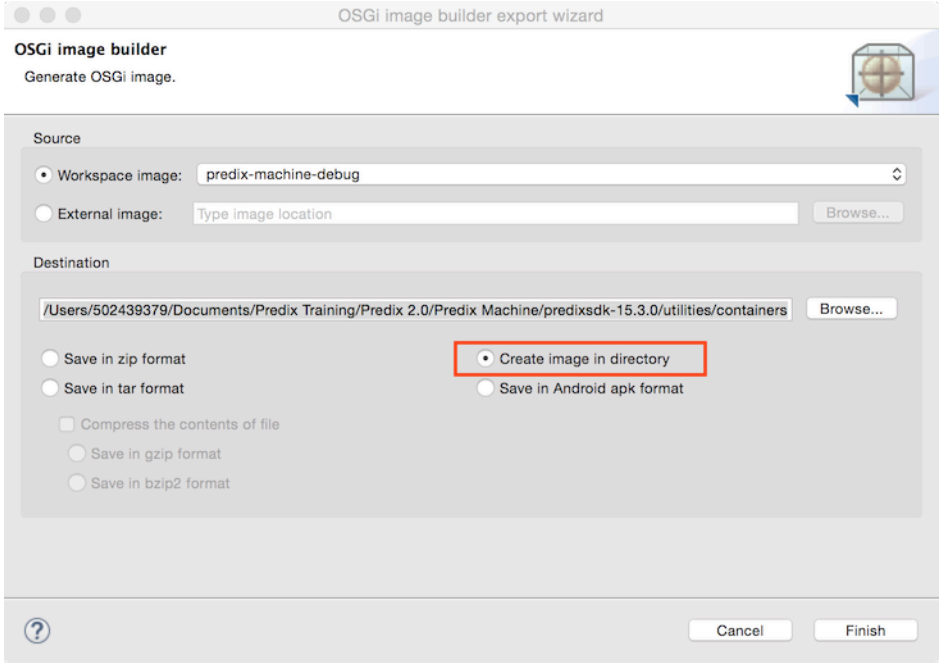
1. Make sure at least one target platform and startup script are selected as shown below. When working with DevBox Select Linux.



2. Export the container image by clicking the Export button in the top right corner

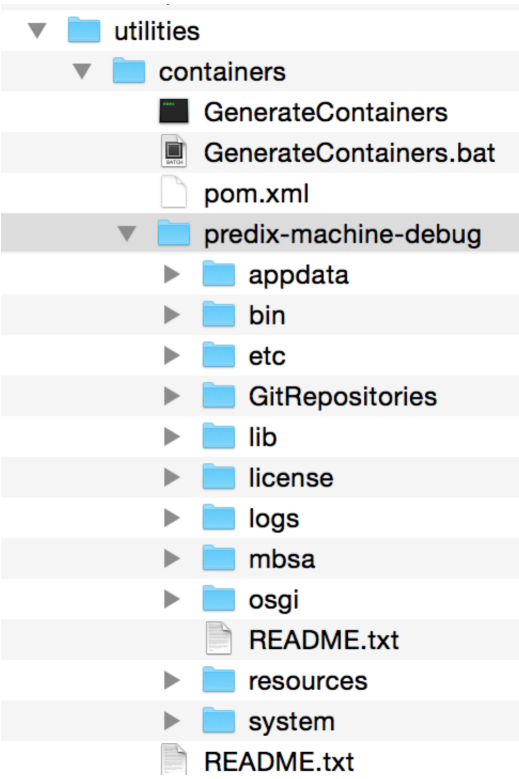


3. Once you have clicked on Export, you will see the following dialog:



4. Select a location by browsing to PredixApps/predixsdk-15.3.0/utilities/containers.

5. Select the “Create image in directory” target for your container. Click Finish. This will create a folder called predix-machine-debug with all the container files in the predixsdk-15.3.0/utilities/containers directory.
6. Locate the exported container in your filesystem. It has the following structure:



Step 4: Start the Debug Container

5. From a terminal window, go to the directory where the startup scripts for generated container are. In our case this is predixsdk-15.3.0/utilities/containers/predix-machine-debug/bin.
6. Run the start script using the following command.

```
$ ./predixmachine clean
```

Step 5: Test Administrator Command Line options

This step will help you discover console commands to interact with the Machine services. If you don't see a prompt in the terminal window, press return until you see fw>$. Try the following basic commands on the console:

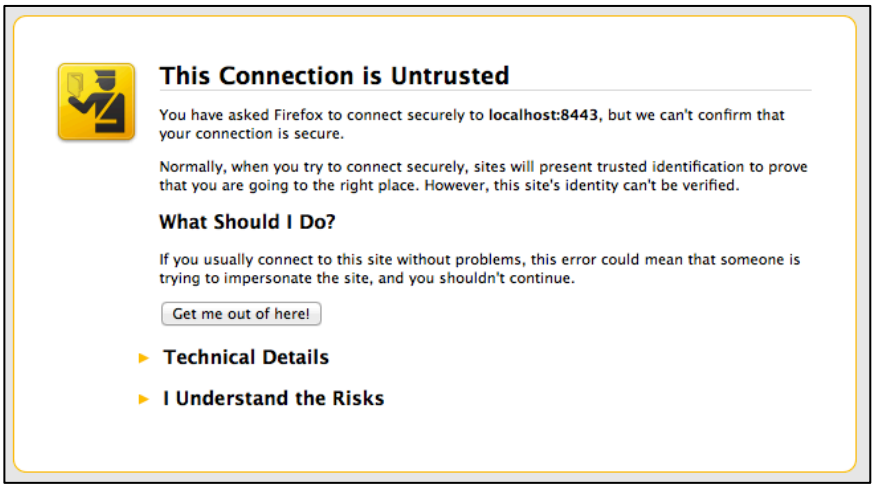
1. `<command>?` : Displays help information for the command
2. `e`: Exit. Ctrl + c will also shut down the container
3. `ls`: Lists the bundles
4. `update 0`: Restart Machine container
5. `bundle <bundle ID>`: Displays bundle details
6. `manifest <bundle ID>`: Displays the manifest of a bundle
7. `services <bundle ID>`: Displays services provided by the bundle
8. `scr.list`: Displays the list of components in the scr group
9. `start <bundle ID>`: Start the bundle
10. `stop <bundle ID>`: Stop the bundle
11. `restart <bundle ID>`: Restart the bundle

Step 6: Start Administrator Web Console

The Debug container can be operated in one of two ways:

- In a terminal window through command line options (see step above)
- Through a web interface. To access the interface, make sure your container is running and browse to https://localhost:8443/system/console

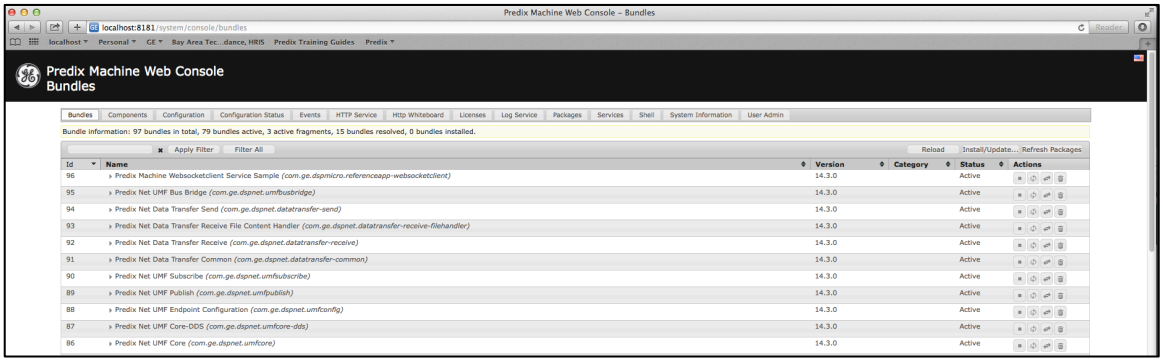
You may see the following screen (or similar depending on your browser).



Machine services use a self-signed certificate by default. You can safely add an exception to allow your browser to connect to Machine. In Firefox, click "I Understand the Risks" -> Add Exception -> Confirm Security Exception. If you are using Chrome, click Advanced -> Proceed to localhost (unsafe). You should then be able to login using the following credentials:

Username: predix

Password: predix2machine



NOTE: Once the container starts, the Web Console administers all of the bundles inside the Machine container. Start, stop, install, reload and other

administrative tasks can be controlled through the UI. It is a personal preference whether to administer the console through the Admin Web Console or at the command line.

Step 7: Test Administrator Web Console options

1. `System->Shell`: Opens terminal window for command line option
2. `OSGi->Bundles`: Displays full list of installed bundles
3. `Technician Console->Configuration`: Displays bundle configuration files
4. `Predix->Store and Forward`: Displays data files waiting for connection to be restored

Lab 3: Create a basic consumer bundle

Duration of lab

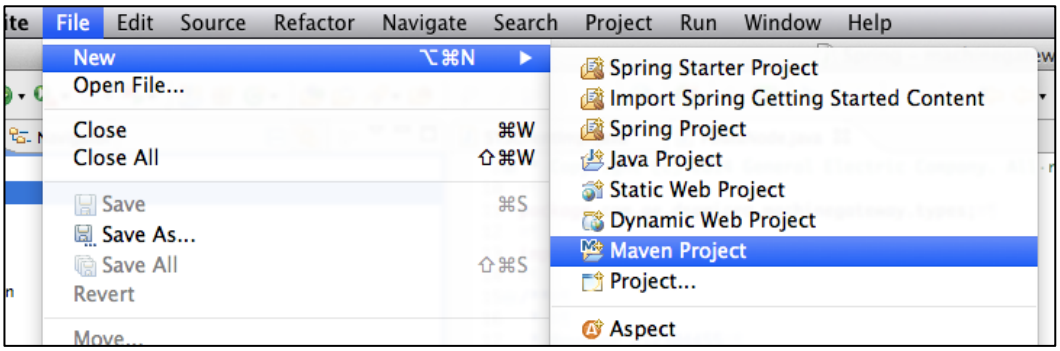
~ 25 minutes

Objectives

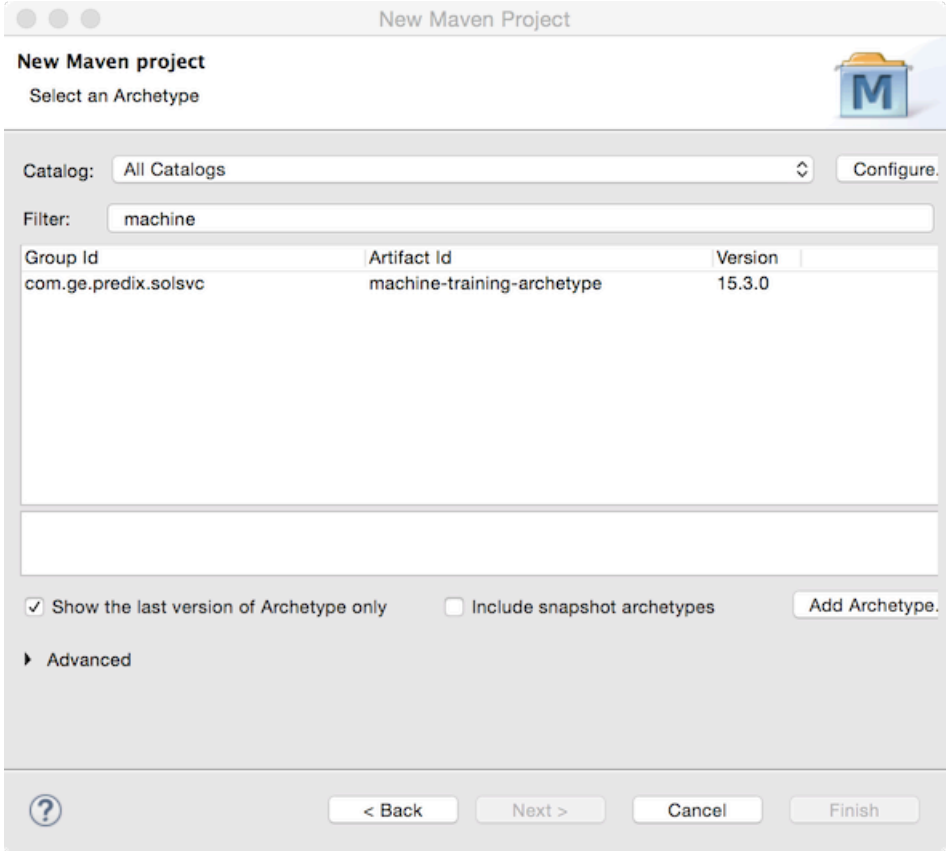
In this lab, you will create a basic bundle that can be deployed into the Machine services container. To simplify the process use a maven archetype. This is a project template plugin provided by maven that allows developers to quickly create projects based on predefined project models. You have created an archetype for Machine services developers to simplify the project creation process.

Step 1: Create a new project from archetype

1. **File->New->Maven Project** (if this option is not available, select **Project** and browse for **Maven Project**)

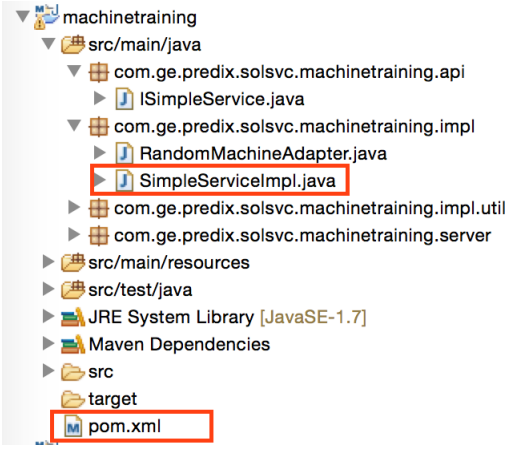


2. Type “machine” in the filter box to locate the machine-training-archetype.



3. Select the archetype **machine-training-archetype**, version 15.3.0. Click Next.
4. Fill out the following fields with the suggested values:
    a. Group Id: com.ge.predix.solsvc
    b. Artifact Id: machinetraining
    c. Package: com.ge.predix.solsvc.machinetraining
5. Click Finish.

Step 2: Examine the project files



1. Open the SimpleServiceImpl.java file and locate the annotations for @Component, @Activate, and @Deactivate.
2. Open the pom.xml file and notice the <packaging> tag. This defines the build as an OSGI bundle. Locate the <instructions> tag at the bottom of this file. These properties become part of the bundle manifest.mf file.

```
<configuration>
    <obrRepository>NONE</obrRepository>
    <instructions>
        <Bundle-SymbolicName>${project.artifactId}</Bundle-Symbol
        <Bundle-Name>${project.artifactId} built from arch-basic-
        <Bundle-Version>${project.version}</Bundle-Version>
        <Bundle-Classpath>.</Bundle-Classpath>
        <Export-Package>com.ge.predix.solsvc.machine.training.api
        <Import-Package>aQute.bnd.annotation.metatype;version=${i
            javax.ws.rs.*;version=${import.javax.ws.rs},
            org.slf4j;version=${import.org.slf4j};provider=paxlog
            org.osgi.service.component;version=${import.org.osgi.
            org.osgi.framework;version=${import.org.osgi.framewor
            org.osgi.service.cm;version=${import.org.osgi.service
            com.ge.predix.solsvc.machine.training.api;version=${i
            com.ge.dspmicro.machinegateway.api;version=${import.c
            com.ge.dspmicro.machinegateway.api.adapter;version=${
            com.ge.dspmicro.machinegateway.types;version=${import
        <Service-Component>*</Service-Component>
    </instructions>
</configuration>
```

Step 3: Build and deploy the project

1. Select the machinetraining project.
2. Right click **Run as->mvn install.** If successful you will see the

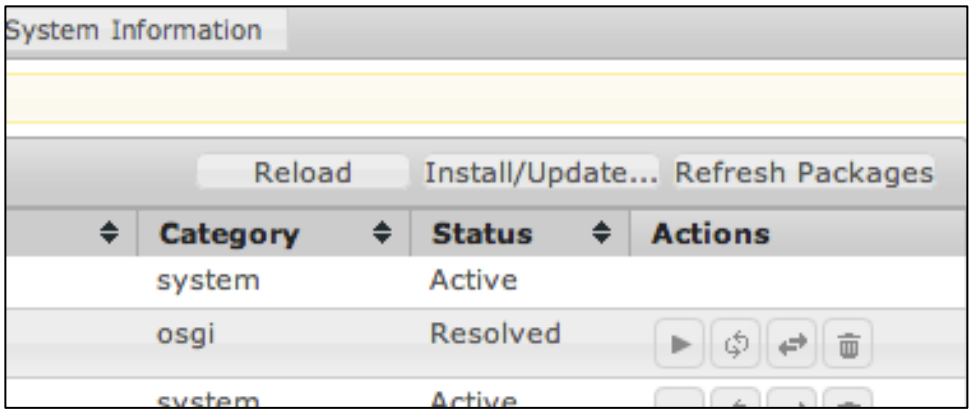
following message in the console window

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.746 s
[INFO] Finished at: 2016-01-15T17:45:08-08:00
[INFO] Final Memory: 22M/216M
[INFO] ------------------------------------------------------------------------
```

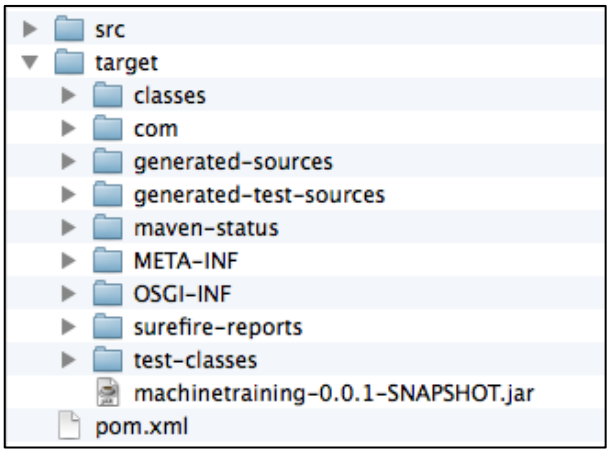
3. Make sure the Machine container is running (refer to Lab 1) and open the web console.

   https://localhost:8443/system/console
   * username: predix
   * password: predix2machine

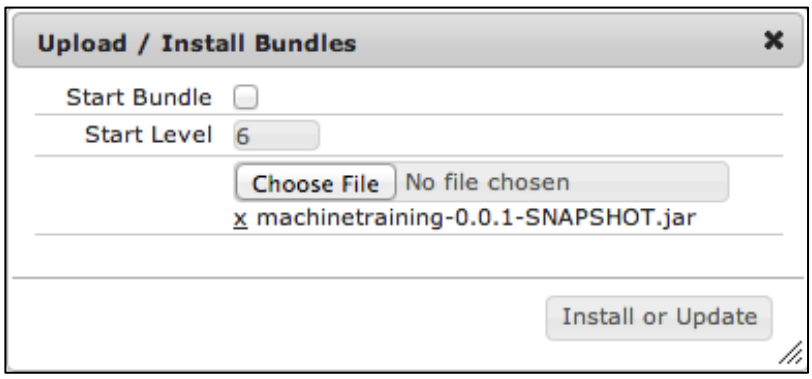
4. Click Install/Update.



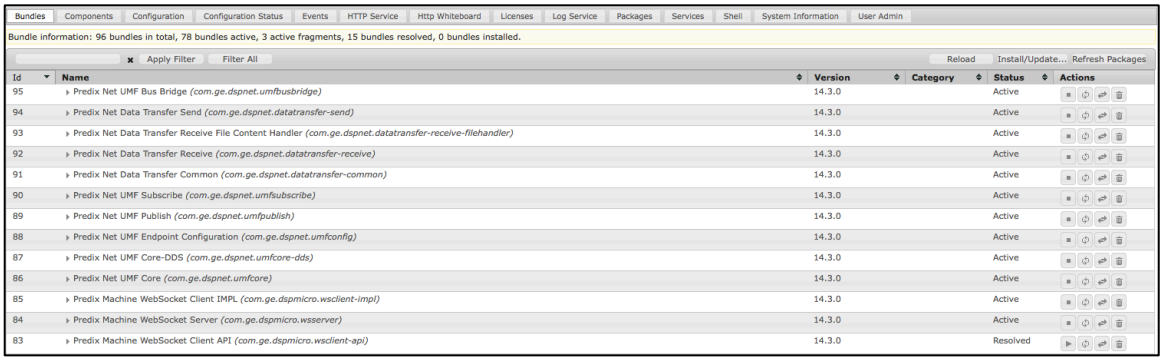
5. Select the bundle by browsing to the project's **target** directory. This should be located in the workspace folder (~/PredixApps/training/machine/workspace). The bundle name is machinetraining-0.0.1-SNAPSHOT.jar.



6. Click Install or Update.



7. Sort the bundle list by descending ID. If you do not see your bundle at the top, click **Reload**. The bundle should be in the **Installed** state.



8. Start the bundle by clicking the play button in the Actions column. Watch the console to see your bundle start. You should see the

following message in the terminal window where you started predix machine.

```
2014-06-19 11:42:19,908[Component Resolve Thread (Bundle
14)]|INFO|com.ge.dspmicro.training.impl.SimpleServiceImpl|71-
machinetraining-
0.0.1.SNAPSHOT this is a test string
```

Lab 4: Consuming another service: Adding Security to the Consumer Bundle

Duration of lab

~ 25 minutes

Objectives

This goal of this lab is to provide hands on experience with:

- Adding a new machine service to your consumer bundle. In this lab you will add a security utility service.
- Using the security service to encrypt a variable in the consumer bundle configuration file.

Step 1: Create a configuration file for the consumer bundle

First you need to create the consumer's configuration file with a non-encrypted password.

1. Go to **<predixmachine>/etc** folder and create a configuration file for the consumer bundle called **com.ge.predix.solsvc.machinetraining.cfg**. You may copy any of the existing .cfg files and remove the all the settings.
2. Add the following 2 properties to the file and save.

```
com.ge.predix.solsvc.machinetraining.uri=https://localhost:8443/testserver/test

com.ge.predix.solsvc.machinetraining.password=secretpassword
```

Step 2: Add security package to POM.xml file

1. In Eclipse, open the POM.XML file for the consumer bundle.
2. Locate the comment <!- - DSP micro API package versions - ->. You will notice several <import.com.ge.dspmicro.<service>.api> statements. These define the range of supported API levels for the package that you are using in your application. Ensure that the following import is listed:

```
<import.com.ge.dspmicro.security.admin.api>"[1.1,2.0)"</i
mport.com.ge.dspmicro.security.admin.api>
```

3. In that same file locate the <Import-Package> tag. If not already there, add the following package at the end of the package list:

```
com.ge.dspmicro.security.admin.api;version=${import.
com.ge.dspmicro.security.admin.api}
```

> Make sure you check the syntax and place a **comma** separating this package from the one above it. Save and close the file.

Step 3: Create a variable for the configuration property

1. Locate the SimpleServiceImple.java file.
2. Add the following static field to your SimpleServiceImpl class. This variable corresponds to the value you added to the configuration file.

```
private static final String PROP_PASSWORD = SERVICE_PID +
".password";
```

Step 4: Add necessary code for consuming a new service

1. In SimpleServiceImple.java add another private field to the class as a handle to the security service.

   ```
   private ISecurityUtils        secUtils;
   ```

   If you get an error press ctrl-shift-o and import `com.ge.dspmicro.security.admin.api`.

2. Create a setter for this field annotated with @Reference. Add the following code after the deactivate() method.

   ```
   @Reference
   public void setSecurityUtils(ISecurityUtils secUtils)
   {
       this.secUtils = secUtils;
   }
   ```

   Type ctrl-shift-O to organize your imports. Select `aQute.bnd.annotation.component` from the list to import the following package into your app:

   `import aQute.bnd.annotation.component.Reference;`.

Step 5: Implement security APIs

1. The activate method is guaranteed to be called when a service starts. This is a good place to encrypt a property. Add the highlighted code to the service.

   ```
   @Activate
   public void activate(ComponentContext ctx)
   {
                       ...
        this.secUtils.encryptConfigProperty(SERVICE_PID,
        PROP_PASSWORD);
                       ...
   }
   ```

2. In order to use the encrypted property, access the value through security utils. Put this somewhere in your activate() method as well. Obviously you would never log a password in a real use case.

```
_logger.info("Original Value : " +
String.valueOf(this.secUtils.getDecryptedConfigProper
rty(SERVICE_PID, PROP_PASSWORD)));
```

Step 6: Build the project and deploy the bundle

1. Build and deploy your bundle. Refer to Lab 2 Step 2 to deploy.
2. To verify that the property has been encrypted, browse to /etc and open up the machinetraining.cfg file. Notice that the password field has been wiped out and an encrypted property has been appended to the file. The configuration file will look like the following.

```
com.ge.predix.solsvc.machinetraining.password =
com.ge.predix.solsvc.machinetraining.password.encrypted =
mTN/VnEVdqNFAOt4Ld1URBSdlJEtZdhTwVbzAdzGCWg=
```

The password is clearly no longer human readable. The console output should also display the correct password.

Lab 5: Generate Gateway Data with Machine Adapter

Duration of lab

~ 25 minutes

Objectives

At the end of this lab, you will be able to: read data from an adapter. You will explore the Machine Gateway, which is one of the core features of Machine services. You will begin by creating a simple adapter that returns random data. You will then implement the interface and register the service with Machine Gateway.

Step 1: Configure the consumer bundle

1. In Eclipse, locate the comment <!- - DSP micro API package versions - -> in the pom.xml file for the consumer bundle. Again, you will notice several <import.com.ge.dspmicro.<service>.api> statements. Ensure that the following import is listed:

   ```
   <import.com.ge.dspmicro.machinegateway.api>"[1.1,2.0)"</import.com.ge.dspmicro.machinegateway.api>
   ```

2. In that same file locate the <Import-Package> tag. If not already there, add the following 3 packages at the end of the package list:

   com.ge.dspmicro.machinegateway.api;version=${import.com.ge.dspmicro.machinegateway.api},

com.ge.dspmicro.machinegateway.api.adapter;version=${import.com.ge.dspmicro.machinegateway.api},

com.ge.dspmicro.machinegateway.types;version=${import.com.ge.dspmicro.machinegateway.api}

Make sure you check the syntax and place a **comma** separating these packages from the ones above.

Step 2: Add necessary code for consuming a new service

1. If not already there, add the private field for Machine Gateway into your SimpleServiceImpl class.

```
private IMachineGateway        gateway;
```

2. Add the setter for the machine gateway field in the SimpleServiceImpl class annotated with @Reference.

```
@Reference
public void setMachineGateway(IMachineGateway gateway)
{
    this.gateway = gateway;
}
```

Step 3: Implement the IMachineAdapter interface

Since this is a lot of code, an implementation has been provided for you as part of the archetype. Take a look at **RandomMachineAdapter.java** . It provides a mock implementation of a machine adapter. When

readData is called from this adapter, it will return a random integer wrapped in the PDataValue class.

In a real use case, this interface should be implemented using a Java library for whatever device technology with which this adapter will interface..

Step 4: Retrieve the adapter through the Machine Gateway API

Adapter type identifies an adapter. A common convention, which is also used by the Random Machine Adapter, is to use the service PID as the adapter type.

1. Add this private field to the SimpleServiceImpl class.

```
private UUID            randomNodeId;
```

2. Now that you have an instance of the machine gateway in your SimpleServiceImpl, insert this code into your activator, after all the code you have put in for the previous labs.

```
for (IMachineAdapter adapter :
      this.gateway.getMachineAdapters())  {
     if ( adapter.getInfo() == null )
     {
         continue;
     }
     if (
RandomMachineAdapter.SERVICE_PID.equals(adapter.getInfo()
.getAdapterType()) )
      {
        this.randomNodeId =
                adapter.getNodes().get(0).getNodeId();
        break;
```

```
        }
}
```

3. Now that you have the nodeId of the single node in the adapter, you can read it directly from the MachineGateway API. There are two ways to read data, either directly from the MachineAdapter or through MachineGateway. In this example you will read from MachineGateway so you do not have to store the reference to the adapter. Add the following code just below the for{} loop.

```
PDataValue value =
     this.gateway.readData(this.randomNodeId);

_logger.info("Reading Random Value" +
     value.getEnvelope());
```

   Import PDataValue from com.ge.dspmicro.machinegateway.types if needed.

4. Build and deploy your bundle. You should see more logging output when you restart the bundle.

Lab 6: Data in Motion: Generate and capture simulated data (DDS Data River)

Duration of lab

~ 25 minutes

Objectives

The overall goal of this lab is to configure a standard Machine container to collect data from Modbus adapter and transfer it using Data River services. At the end of the lab, you will be able to:

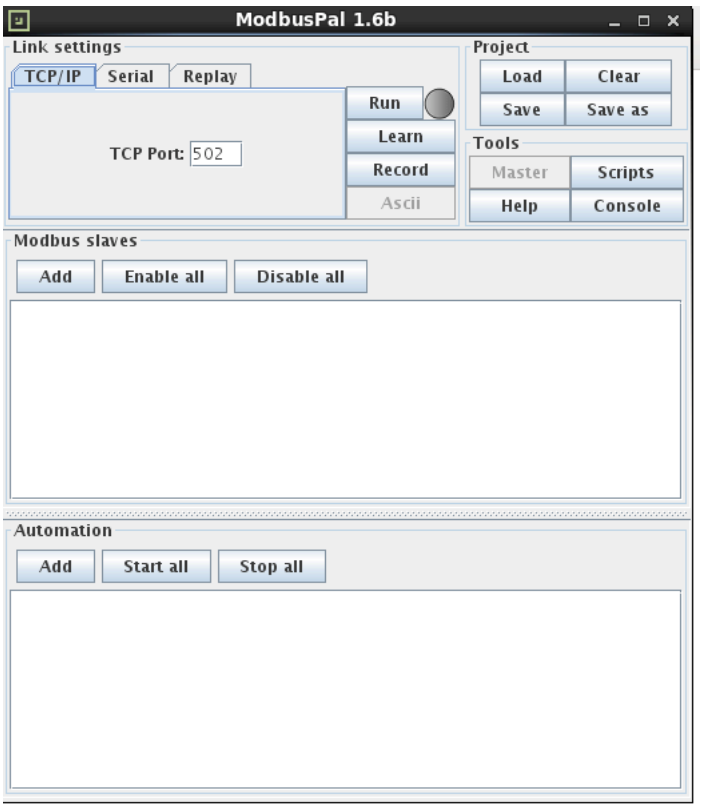
- Configure a Modbus Simulator
- Configure the Modbus adapter in the Machine container
- Configure a spillway for processing data
- Stage generated data through Data Store and Forward
- Configure a DDS Data River and move data locally

If your Predix Debug Machine is running, stop the container. You will restart after reconfiguring the bundles.

Step 1: Download and start the Modbus PLC simulator

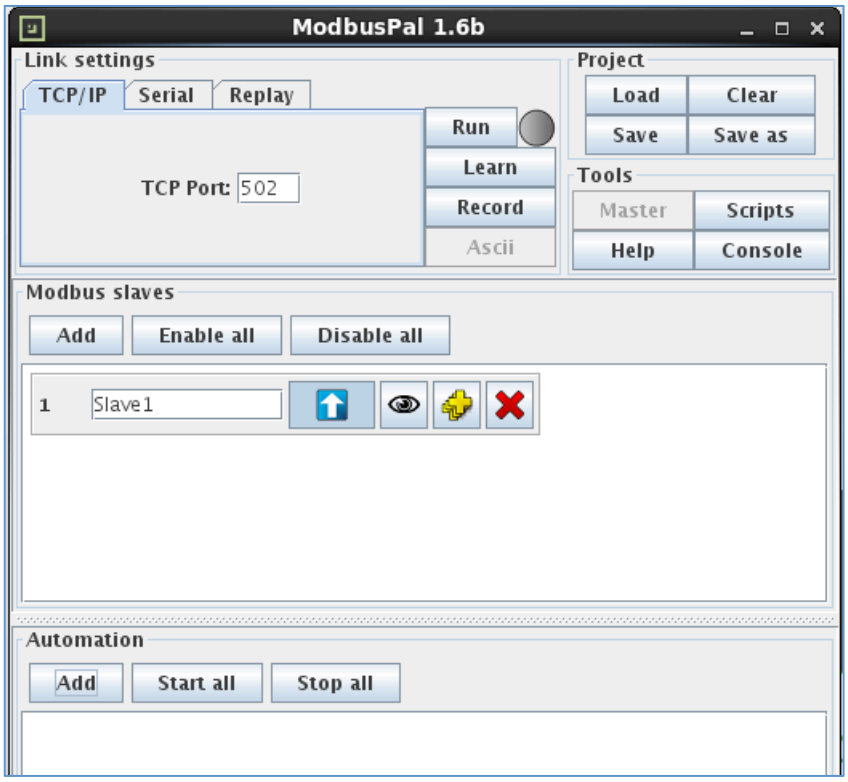
1. Download ModbusPal.jar file from http://sourceforge.net/projects/modbuspal/files/modbus and copy the jar file to ~/PredixApps/training/machine.
2. From a terminal window navigate to ~/PredixApps/training/machine.
3. Start the Modbus simulator with the following command:

```
$ sudo java –jar ModbusPal.jar
```



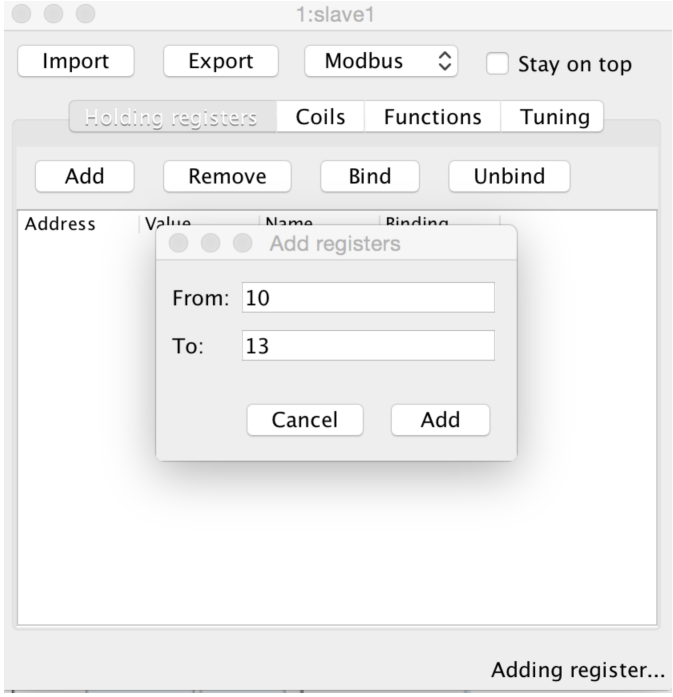
4. In the Modbus slaves section, click on Add to add a slave.

5. Select 1, update the Slave name to “Slave1”, and click the Add button.

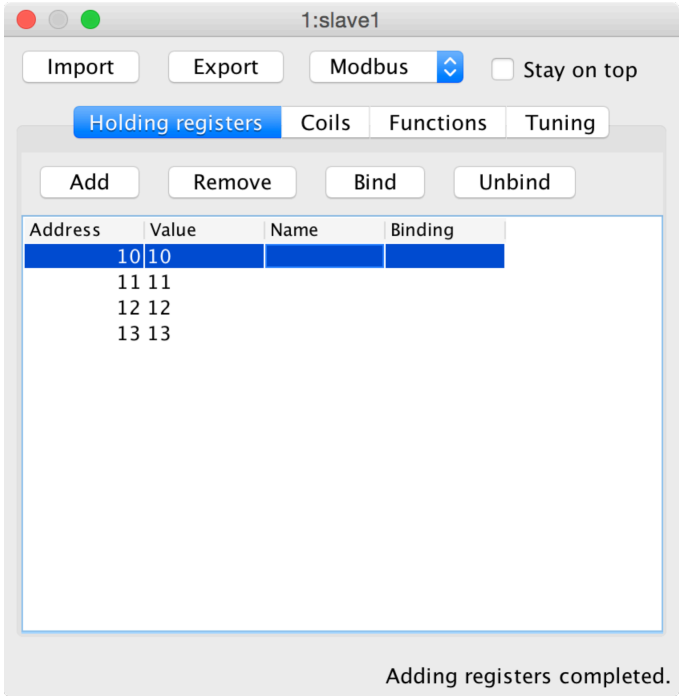


6. Click on the edit button (the button with the eye icon).

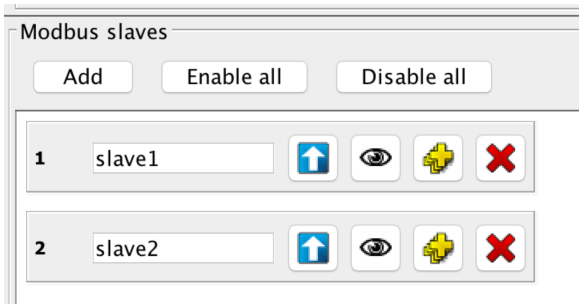
7. In the Holding Registers tab, click on the Add button.



8. Reset the number of registers from 10 to 13 and click on Add. Set the values for the registers to random numbers. The slave configuration should look similar to the picture below:



9. Close the slave editor window.
10. Add a second slave and configure the register range from 20 to 23.



11.Click the Run button to start the simulator

Step 2: Configure the Modbus adapter in Machine services

Predix Machine supports Modbus through the Modbus adapter bundle. This bundle must be configured to work with a given Modbus device.

1. The configuration properties for the Modbus bundle can be found in **etc/com.ge.dspmicro.machineadapter.modbus-0.xml**. Open this file in a text editor. Remove the TCP/IP comments for the channel under **<dataNodeConfigs>**

```
<!-- REMOVE THIS LINE FOR TCP/IP
        <channel protocol="TCP_IP" tcpIpAddress="127.0.0.1"
              tcpIpPort="502">
   . . .
REMOVE THIS LINE -->
```

2. Remove the TCP/IP comments for dataSubscriptionConfigs under **<dataSubscriptionConfigs>**

This will configure the Modbus adapter to listen for values on port 502 at IP address 127.0.0.1. The Modbus PLC simulator software runs at this

location by default.

Step 3: Configure a spillway for DDS

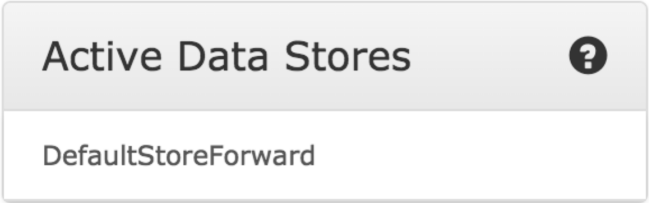
The Data River service allows data to be passed through spillways for processing. In this example, we will set up a simple subscription-based spillway to listen to the Modbus adapter.

1. Open the file **etc/com.ge.dspmicro.hoover.spillway-0.cfg**. Change the com.ge.dspmicro.hoover.spillway.destination property to the following:

   ```
   com.ge.dspmicro.hoover.spillway.destination=Sender
   Service
   ```

Step 4: Begin collecting "data forward" data

1. Start the Machine Container
2. Open Admin Web Console (localhost:8443/system/console)
3. Select Predix→Store and Forward. Select DefaultStoreForward.



Within a minute you should see a numbered list of data objects being captured and stored every 60 seconds. Because there is no connection to the cloud yet, this data is saved. Only the last 10 data objects are shown. You will see errors in the terminal window until the data river is configured.

4. Stop the Machine Container

Step 5: Configure the Data River Sender and Receiver service

The Data River Send service is configured by default to transmit data from all spillways to configured Data River Receiver services.

1. In the **etc/com.ge.dspnet.datariver.send-0.cfg** file update the setting for secure transfer to false.

```
com.ge.dspnet.datariver.send.secure.transfer=false
```

The Data River Receiver service is configured to listen for a Data River Sender service in the same container by default.

2. In the **etc/com.ge.dspnet.datariver.receive-0.cfg** file, update the setting for secure transfer to false.

```
com.ge.dspnet.datariver.receive.secure.transfer=false
```

Step 6: Restart the Machine and Forward Stored Data

1. From the /bin directory restart the machine container

```
$ ./predixmachine clean
```

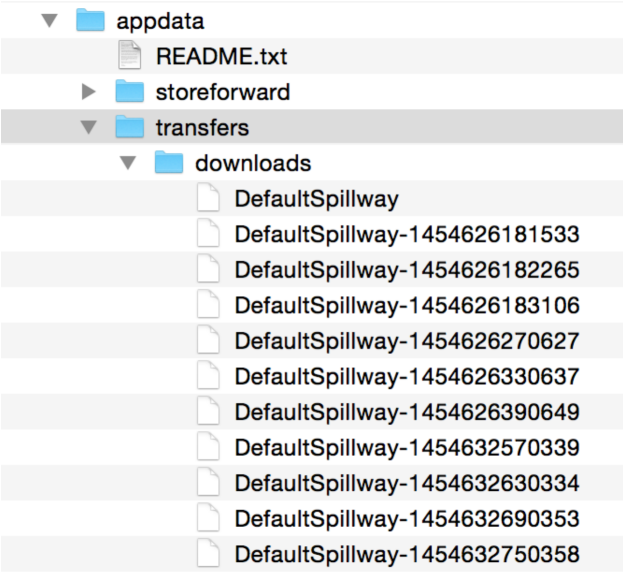
2. Open the Admin Console and go to **Predix->Data Store and Forward** The data under DataStoreForward is now gone (it was sent to data river receiver you just configured).
3. In the terminal window you will see a success message repeated

every few seconds.

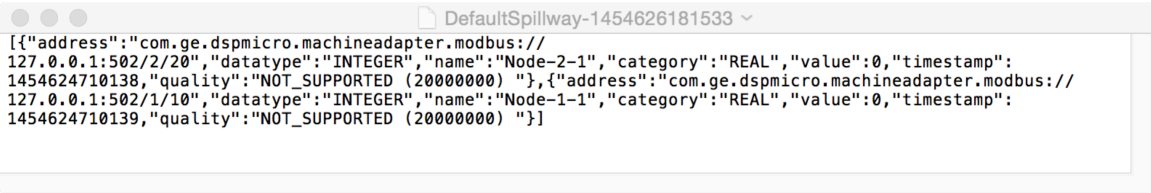
```
fw>$
fw>$2016-02-04 16:38:10,337[Thread-
40]|INFO|com.ge.dspnet.datariver.send.impl.DataSendImpl|83-
com.ge.dspnet.datariver-send-15.3.0|Status Code: -1,100. Request
valid. Transfer ID: efeb4219-4d18-44d0-b0c9-311d75cc09d6.
 2016-02-04 16:38:10,340[pool-9-thread-
1]|INFO|com.ge.dspnet.datariver.receive.impl.DataReceiveCallback|
81-com.ge.dspnet.datariver-receive-15.3.0|Status Code: -2,107.
Opening new transfer session. Transfer ID: efeb4219-4d18-44d0-
b0c9-311d75cc09d6
 2016-02-04 16:38:10,342[pool-11-thread-
1]|INFO|com.ge.dspnet.datariver.send.impl.DataSendImpl|83-
com.ge.dspnet.datariver-send-15.3.0|Status Code: -1,202.
TRANSFER_RESPONSE received. Beginning transfer with Transfer ID:
efeb4219-4d18-44d0-b0c9-311d75cc09d6.
 2016-02-04 16:38:10,353[pool-10-thread-
1]|INFO|com.ge.dspnet.datariver.send.impl.SendContentTask|83-
com.ge.dspnet.datariver-send-15.3.0|Total Bytes Sent = 409.
Transfer ID: efeb4219-4d18-44d0-b0c9-311d75cc09d6.
 2016-02-04 16:38:10,354[pool-8-thread-
1]|INFO|com.ge.dspnet.datariver.receive.filecontenthandler.FileCo
ntentHandler|82-com.ge.dspnet.datariver-receive-filehandler-
15.3.0|
Transfer "efeb4219-4d18-44d0-b0c9-311d75cc09d6" completed.
Total bytes received:409 B
Total time: 0 seconds
 Average speed: 399.4 KiB/s
 Finished at: 2016/02/04 16:38:10,354
 2016-02-04 16:38:10,358[pool-11-thread-
1]|INFO|com.ge.dspnet.datariver.send.impl.DataSendImpl|83-
com.ge.dspnet.datariver-send-15.3.0|Status Code: -1,200.
Successful transfer with ID: efeb4219-4d18-44d0-b0c9-
311d75cc09d6.
```

Because the DDS Data River defaults to localhost, the receiver will store data it receives in files in the **appdata/transfers/downloads** directory inside the container directory.

4. Open this directory. You should see a new file created once per 10 seconds containing values from the Modbus PLC simulator.



The contents of the file should be formatted as JSON and should be similar to the following:




```
[{"address":"com.ge.dspmicro.machineadapter.modbus://
127.0.0.1:502/2/20","datatype":"INTEGER","name":"Node-2-1","category":"REAL","value":0,"timestamp":
1454624710138,"quality":"NOT_SUPPORTED (20000000) "},{"address":"com.ge.dspmicro.machineadapter.modbus://
127.0.0.1:502/1/10","datatype":"INTEGER","name":"Node-1-1","category":"REAL","value":0,"timestamp":
1454624710139,"quality":"NOT_SUPPORTED (20000000) "}]
```

Lab 7: Data in Motion: Connect to the Cloud with HTTP Data River and HTTP Service

Duration of lab

~ 25 minutes

Objectives

The overall goal of this lab is to configure a standard Machine container to collect data from Modbus adapter and transfer it to the cloud using HTTP Data River services. At the end of the lab, you will be able to:

- Configure a connection between machine and Predix Cloud
- Build and deploy an HTTP Data Service in Predix Cloud
- Configure the HTTP Data River in Predix Machine

Step 1: Cloud Foundry Setup

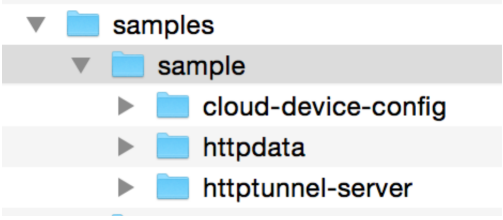
1. Make sure you are on "Internet" and proxies are not set. In a terminal window type **$ env | grep http**. The values for http(s)_proxy and HTTP(S)_PROXY should not be set.
2. Log into Cloud Foundry and locate the UAA instance you created in an earlier class. Make sure you know the **name** and **secret** of the client you added
3. Create an instance of postgres using postgres93-docker

```
$ cf cs <your-name>-postgres-instance free postgres93-docker
```

4. Stop the Predix Machine container

Step 2: Import HTTP Data Service Sample from Predix SDK

1. From Eclipse, select **File->Import->Existing Maven Project** and browse to the sample folder in the samples folder.



   Click Finish. The sample imports as predixmachine-http-data project.

2. Open the project and locate the **manifest.yml** file. Update the name to <your-name>-httpservice. Add the names of the postgres instance and UAA instance you created under services. Save this file.
3. Build the sample app: **Run As->Maven Install**
4. Open a terminal window and go the samples/sample/httpdata folder where the manifest.yml file is. Push the app to the cloud and make sure it starts:

```
$ cf push
$ cf apps
```

5. Once the app is running, open a browser window and enter the application route followed by /vi/welcome. For example http://<app>.run.aws-usw02-pr.ice.predix.io/v1/welcome

   You should see "Welcome to Predix Machine HTTP Data Service".

6. Get the VCAP environment variables for the UAA issueId and the

application route.

```
$ cf env <your-name>-httpservice
```


```
],
"predix-uaa": [
 {
  "credentials": {
   "issuerId": "https://4c76482a-9ffd-4aa6-8b64-31977dc5f8c9.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token",
   "uri": "https://4c76482a-9ffd-4aa6-8b64-31977dc5f8c9.predix-uaa.run.aws-usw02-pr.ice.predix.io",
   "zone": {
    "http-header-name": "X-Identity-Zone-Id",
    "http-header-value": "4c76482a-9ffd-4aa6-8b64-31977dc5f8c9"
   }

"name": "charlie-httpdata",
"space_id": "696d6fd2-ccc9-4b7b-ae23-6c6fffa459f1",
"space_name": "demo",
"uris": [
 "charlie-httpdata.run.aws-usw02-pr.ice.predix.io"
],
```


Step 3: Configure Cloud Foundry Credentials

1. Edit /etc/com.ge.dspmicro.predixcloud.identity.cfg
   - com.ge.dspmicro.predixcloud.identity.uaa.token.url=<UAA issuerid> (for instance https://abca/oauth/token)
   - com.ge.dspmicro.predixcloud.identity.uaa.clientid=<client ID>
   - com.ge.dspmicro.predixcloud.identity.uaa.clientsecret=<secret>

Step 4: Configure spillway for HTTP

2. Edit /etc/com.ge.dspmicro.hoover.spillway-o.cfg
   - com.ge.dspmicro.hoover.spillway.destination=**Http** Sender Service

Step 5: Configure HTTP River

3. Edit com.ge.dspmicro.httpriver.send-0.cfg
   - com.ge.dspmicro.httpriver.send.destination.host=<route of app running in cloud>

Step 6: Restart Predix Machine

4. Go to the /bin directory and start Machine

   $ ./predixmachine clean

   You will see a success message repeated every few seconds.

```
fw>$2016-02-03 15:31:10,786[Thread-
45]|INFO|com.ge.dspmicro.httpriver.send.impl.HttpRiverSendImpl|76
-com.ge.dspmicro.httpriver-send-15.3.0|Transfer completed
successfully.
 2016-02-03 15:31:10,786[Thread-
45]|INFO|com.ge.dspmicro.httpriver.send.impl.HttpRiverSendImpl|76
-com.ge.dspmicro.httpriver-send-15.3.0|[200] transferId:
2adabb8d-2349-4799-bdc8-0257ca6535cc
riverId: cde39710-733b-41cc-8e38-c73bdd093475
riverName: Http Sender Service
contentType: application/octet-stream
contentDisposition: null
contentDescription: null
timestamp: 1454513470699
deviceId: sample-clientid
```

Step 7: Stop ModbusPal and shut down machine

Troubleshooting

Troubleshooting permissions

Are you seeing java.security.AccessControlException?

```
java.security.AccessControlException: access denied
("java.net.somepermission" "localhost" "resolve")
```

The reason for this is because your bundle is attempting to use functionality that is not permitted. During your debugging process as you consume more Machine features, you will need to manually add these permissions. It can sometimes be a tedious process but it allows the container to control which bundles have access to which features.

The best way to add necessary permissions is to add them as the exceptions appear during development. To fix this above one for example, open up your <bundle project>/src/main/resources/OSGI-INF/permissions.perm file and append the access permission shown above.

For example, lets say we had an issue with ("java.net.SocketPermission" "localhost" "resolve"). Open up the permissions.perm file, add this line to the file

```
    (java.net.SocketPermission "*" "resolve")
```

This permission allows bundles with socket access to resolve any host, indicated by the splat (*). Build your bundle and try it again.

For this particular example, you may run into another permission exception("java.net.SocketPermission" "127.0.0.1:8181" "connect,resolve"). This exception is now different from the previous one which means we are making progress. Now modify the permission you added above to be `(java.net.SocketPermission "*" "connect,resolve")`

This should now allow socket operations to resolve a host and connect to the host.

Adding bundle permissions is an iterative and repetitive process. The archetype provided covers most of the required permissions but may need some additional entries. The permissions system helps ensure that bundles only access features they are permitted to access. This helps keep the system secure and avoid unauthorized use.