



# Predix Machine

Documentation



**Proprietary Notice**

The information contained in this publication is believed to be accurate and reliable. However, General Electric Company assumes no responsibilities for any errors, omissions or inaccuracies. Information contained in the publication is subject to change without notice.

No part of this publication may be reproduced in any form, or stored in a database or retrieval system, or transmitted or distributed in any form by any means, electronic, mechanical photocopying, recording or otherwise, without the prior written permission of General Electric Company. Information contained herein is subject to change without notice.

© 2021, General Electric Company. All rights reserved.

**Trademark Notices**

GE, the GE Monogram, and Predix are either registered trademarks or trademarks of General Electric Company.

Microsoft® is a registered trademark of Microsoft Corporation, in the United States and/or other countries.

All other trademarks are the property of their respective owners.

We want to hear from you. If you have any comments, questions, or suggestions about our documentation, send them to the following email address:

[doc@ge.com](mailto:doc@ge.com)

Edge Software and Services Overview.....	6
About Edge Software and Services.....	6
Predix Machine Overview.....	7
What is Predix Machine?.....	7
How Does Predix Machine Fit into the Predix Platform?.....	10
Why Should I use Predix Machine?.....	10
Quick Start: Getting Data from a Device to the Cloud.....	13
Quick Start: Getting Data From a Device to the Cloud.....	13
Get Started with Predix Machine.....	14
Predix Machine SDK Overview.....	14
Requirements.....	19
Verified Platforms.....	22
Downloading the Predix Machine SDK.....	24
Predix Machine SDK Directory Structure.....	28
Installing the Predix Machine SDK.....	28
Generate Predix Machine Containers.....	33
User Permissions and Roles.....	41
Starting Predix Machine.....	43
Run Predix Machine as a Service.....	43
Build and Run Sample Applications.....	45
Accessing Predix Machine Web Console.....	50
Secure Predix Machine.....	52
Securing Predix Machine.....	52
Protecting the Security Folder.....	52
Changing KeyStore and TrustStore Passwords.....	53
Issuing Certificates.....	54
Securing the Technician Console.....	54
Predix Machine VPN Service.....	55
Device Enrollment in the Cloud.....	60
About Predix Cloud Device Enrollment.....	60
Adding a Device to Predix Edge Manager.....	62
Enroll a Device using Predix Edge Technician Console.....	64

Using Predix Edge Technician Console to Enroll Devices with Predix Cloud.....	64
Enroll a Device using the Predix Machine Technician Console.....	65
Predix Machine Technician Console-based Device Enrollment.....	65
Accessing Predix Machine Web Console.....	66
Enrolling a Predix Machine-enabled Device with the Cloud.....	68
Upgrade Predix Machine.....	69
Upgrading Over the Air Using Edge Manager.....	69
Upgrading a Predix Machine Container.....	69
Upgrade Return Codes.....	71
Working with Slow Machines.....	72
Connect Data to the Cloud.....	72
Cloud Services Overview.....	72
Predix Cloud Identity Management Service.....	73
WebSocket River.....	79
Event Hub River Service.....	85
HTTP Data Service.....	91
Command Framework.....	92
Connect Data to the Edge.....	103
Edge Services Overview.....	103
Core Services.....	104
Application Services.....	115
Machine Gateway Services.....	119
Cloud Gateway Services.....	184
Mobile Gateway Services.....	241
Predix Machine Containerization.....	246
Containerization Overview.....	246
Predix Edge SDK.....	248
Data Flow in a Containerized Environment.....	252
Download and Run Containerization Software.....	273
Generate Docker Images.....	279
Troubleshoot Predix Machine.....	283
.....	283

Troubleshoot Eclipse.....	283
Troubleshoot Predix Machine.....	286
Troubleshoot Device Enrollment.....	288
Troubleshoot Synchronization Issues.....	294
Troubleshoot Web Console Errors.....	295
Troubleshoot Container Errors.....	296
Troubleshoot Container Upgrades.....	299
Release Notes.....	300
Predix Machine Release Notes Version 17.2.5.....	300
Predix Machine Release Notes Version 17.2.4.....	300
Predix Machine Version 17.2.3.....	303
Predix Machine Version 17.2.2.....	306
Predix Machine Version 17.2.1.....	309
Predix Machine Version 17.2.0.....	311
Predix Machine Version 17.1.3.....	314
Predix Machine Version 17.1.2.....	317
Predix Machine Version 17.1.1.....	320
Predix Machine Version 17.1.....	323
Predix Machine Data Collection Configuration.....	325
Ingesting Data Using Modbus Adapter.....	325
Ingesting Data Using MQTT Adapter.....	334
<b>Chapter 2. Index.....</b>	<b>a</b>

# Edge Software and Services Overview

## *About Edge Software and Services*

### **Predix edge**

Predix is an edge-to-cloud solution, where the edge is the physical location that allows computing closer to the source of data. Edge computing enables analytics and insights to occur closer to the deployed machines, which are the source of the data.

The edge includes Predix Machine and Predix Connectivity.

- **Predix Machine** is in close contact with the generic asset, providing the capability to exchange data with cloud-based services like big data analytics and asset management. The outcomes of the algorithms from big data analytics and the decision rules from asset management can then be sent back to the machines, optimizing and maximizing performance.

Predix Machine includes the following features

- Predix cloud connectivity
- Machine-to-machine connectivity using industrial protocols like OPC-UA, Modbus, and MQTT.
- Built-in application services like logging, Store and Forward to prevent data loss, and certificate management.
- Docker support
- Device technician console

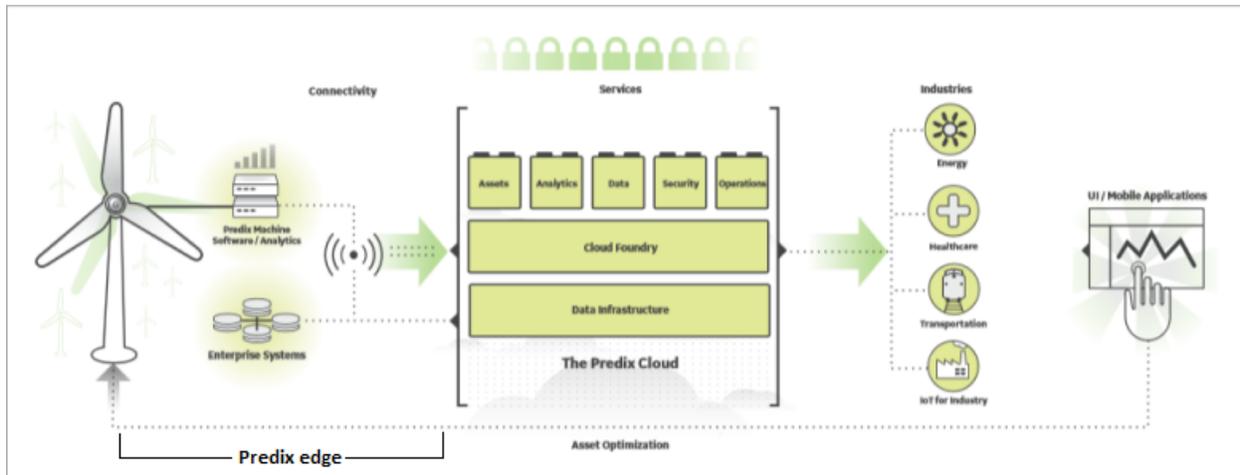
See [Predix Machine Overview \(page 7\)](#) for more information.

- **Predix Connectivity** establishes connectivity between edge devices and the Predix cloud over various access networks, including cellular, fixed line, and satellite communication.

See [Connectivity](#) for more information.

- **Predix-Ready devices** are hardware machines, devices, and software that interoperate with Predix through published APIs and other integration points.

The Predix edge is illustrated in the context of the Predix platform in the following figure:



## Edge Manager

Predix Edge Manager is an interface that is part of Predix cloud, rather than the edge, but provides a single-pane-of-glass view of edge devices and provides insights into device condition and network health. You can manage edge devices and administer apps and configuration files at both a device and fleet level.

See [About Predix Edge Manager \(page 1\)](#) for more information.

## Predix Machine Overview

### *What is Predix Machine?*

Predix Machine is a software platform, which includes a set of technologies, tools, and services that enable the development, deployment, and management of applications, or solutions on embedded hardware that is connected to physical machines.

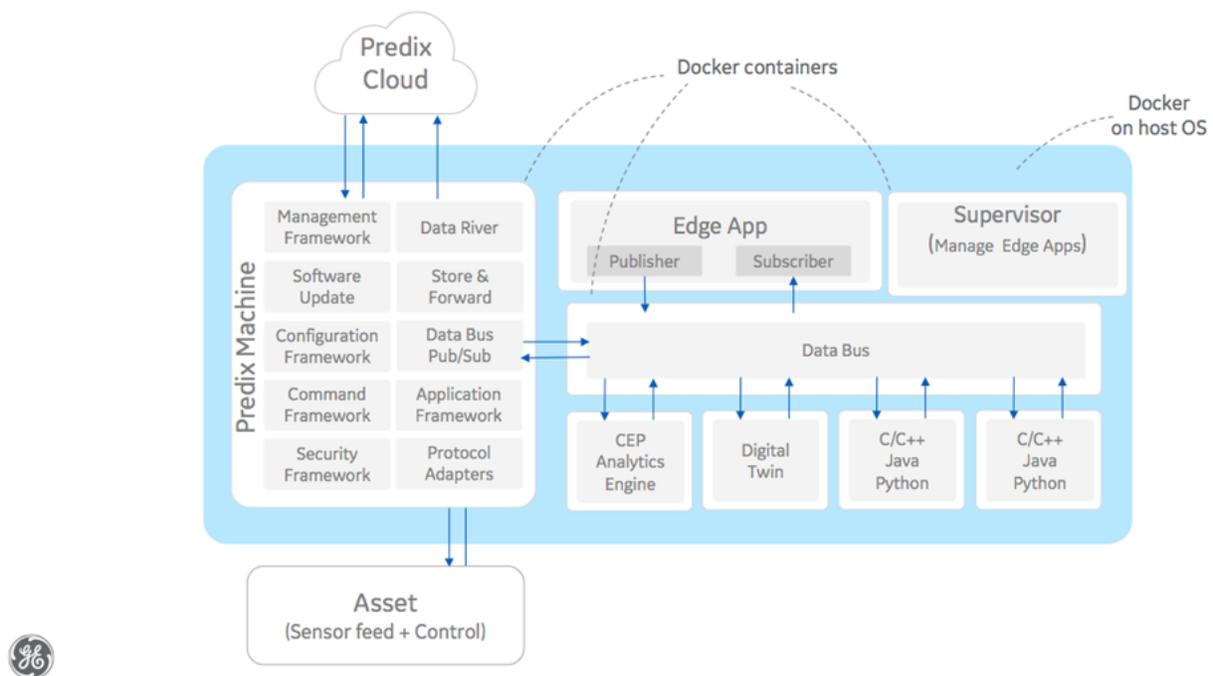
The platform provides Machine-to-Cloud, (M2C) Machine-to-Machine (M2M), and Machine-to-Mobile or –Human (M2H) application connectivity. Predix Machine is capable of running on UNIX-like operating systems, such as Linux or MacOSX. Predix Machine is a Java application with configurable options and customizations. It can also be used in a Docker environment (as shown in the figure below), which allows you to develop entire applications that can be managed by the cloud in any language you want.

Another significant design driver is maintaining the platform while consuming minimal memory and resources. With Java support, Predix Machine can successfully run on devices down to single-board computers like the Raspberry Pi B or BeagleBone Black.

Predix Machine is in close contact with generic assets, providing the capability to exchange data with cloud-based services like big data analytics and asset management. The outcomes of the algorithms from big data analytics and the decision rules from asset management can then be sent back to the machines, optimizing and maximizing performance.

A Predix Machine high-level functional architecture, in a Docker environment, is shown in the following figure.

## Predix Machine Architecture



**Table 1. Predix Machine Services**

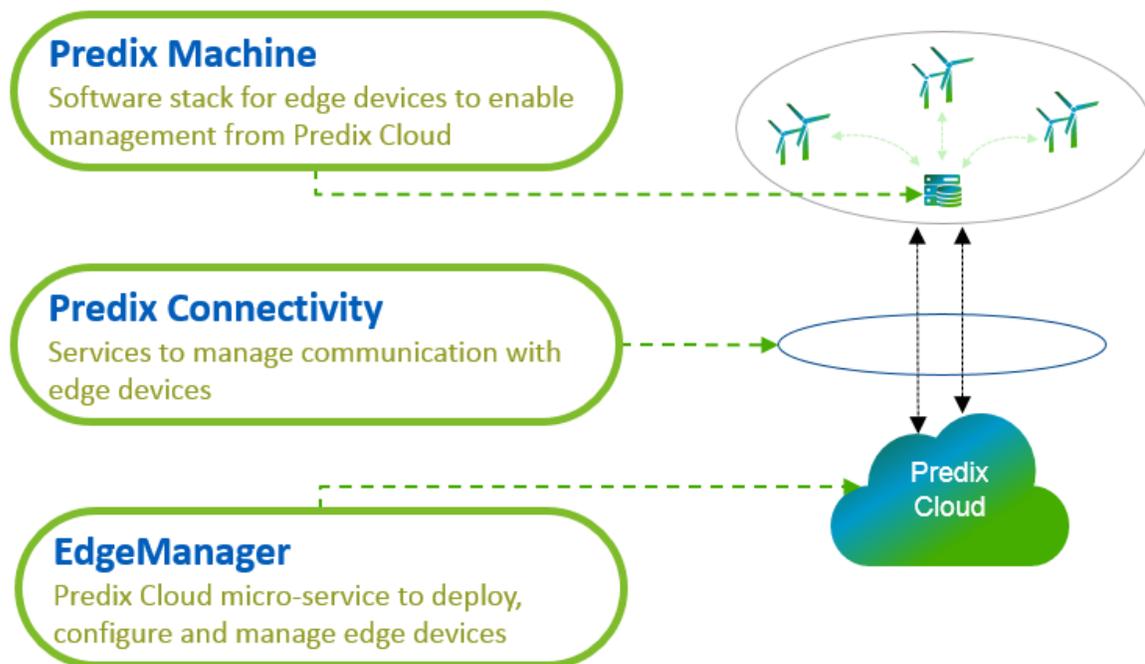
Cloud Services		Edge Services	
Service	Description	Service	Description
<b>Predix Cloud Identity Management</b>	The Predix Cloud Identity Management Service allows you to enroll a Predix Machine device using certificate enrollment or as an OAuth2 client and provides token management after device enrollment.	<b>Core Services</b>	Includes services that support logging and provide security and certificate management.
<b>WebSocket River</b>	Provides connectivity between a Predix Machine-enabled device and the Predix Time Series service in the cloud.	<b>Application Services</b>	Includes services that support user management and the Git repository.
<b>Event Hub River</b>	Enables data transfer from Predix Machine-enabled edge devices to the Predix cloud using HTTP.	<b>Machine Gateway</b>	Includes services that support the machine gateway, which uses industrial protocols like OPC-UA, Modbus, and MQTT and their corresponding adapters.

Cloud Services		Edge Services	
Service	Description	Service	Description
<b>HTTP Data</b>	Predix Machine collects data from various sensors and devices and forwards the collected data to the Predix cloud through the Event Hub River service.	<b>Cloud Gateway</b>	Includes services that provide APIs to build client-side HTTP-compliant applications, allow communication of different network protocols with tunneling, and establish proxy settings.
<b>Command Framework</b>	Enables you to send commands to applications that are running on a device.	<b>Mobile Gateway</b>	Includes the WebSocket Server service, which enables applications to host a WebSocket server endpoint.

## Predix Edge

Predix is an edge-to-cloud solution, where the edge is the physical location that allows computing closer to the source of data. Edge computing enables analytics and insights to occur closer to the deployed machines, which are the source of the data.

# Products for Edge



## EdgeManager

Predix Edge Manager is an interface that is part of Predix cloud, rather than the edge, but provides a single-pane-of-glass view of edge devices and provides insights into device condition and network

health. You can manage edge devices and administer apps and configuration files at both a device and fleet level.

See [About Predix Edge Manager](#) (page ) for more information.

**Related concepts**

[Predix Machine SDK Overview](#) (page 14)

[About Containers](#) (page 246)

[Predix Machine Containerization Overview](#) (page 247)

## How Does Predix Machine Fit into the Predix Platform?

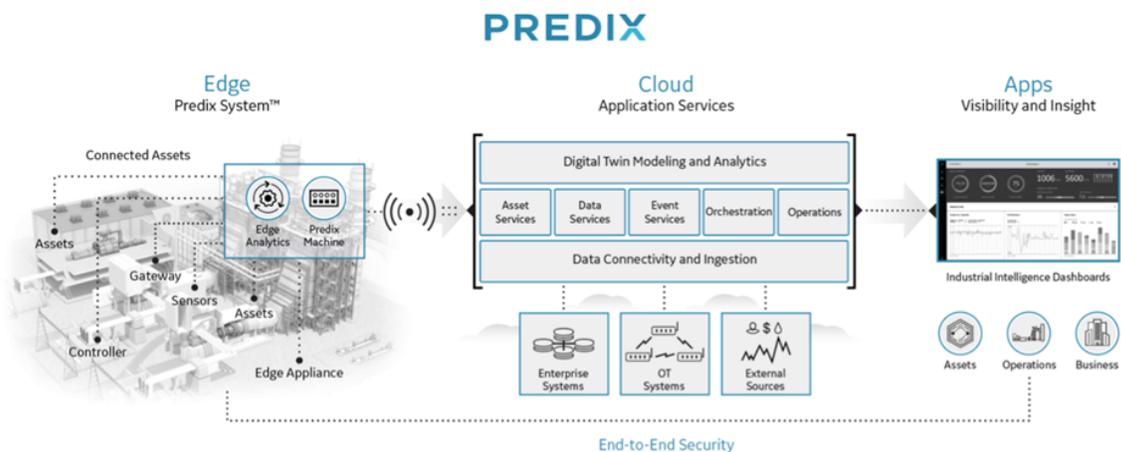
Predix Machine is part of the wider Predix platform. Predix Machine is the runtime environment for Predix Edge, which enables you to securely connect assets to the cloud.

Predix Machine is in close contact with assets and provides the ability to securely exchange data with cloud-based services like big data analytics and asset management.

The outcomes of the algorithms from big data analytics and the decision rules from asset management can then be sent back to the machines, optimizing and maximizing performance for the customer’s benefit.

A representation of the Predix platform is shown in the following figure:

### Edge to Cloud Platform



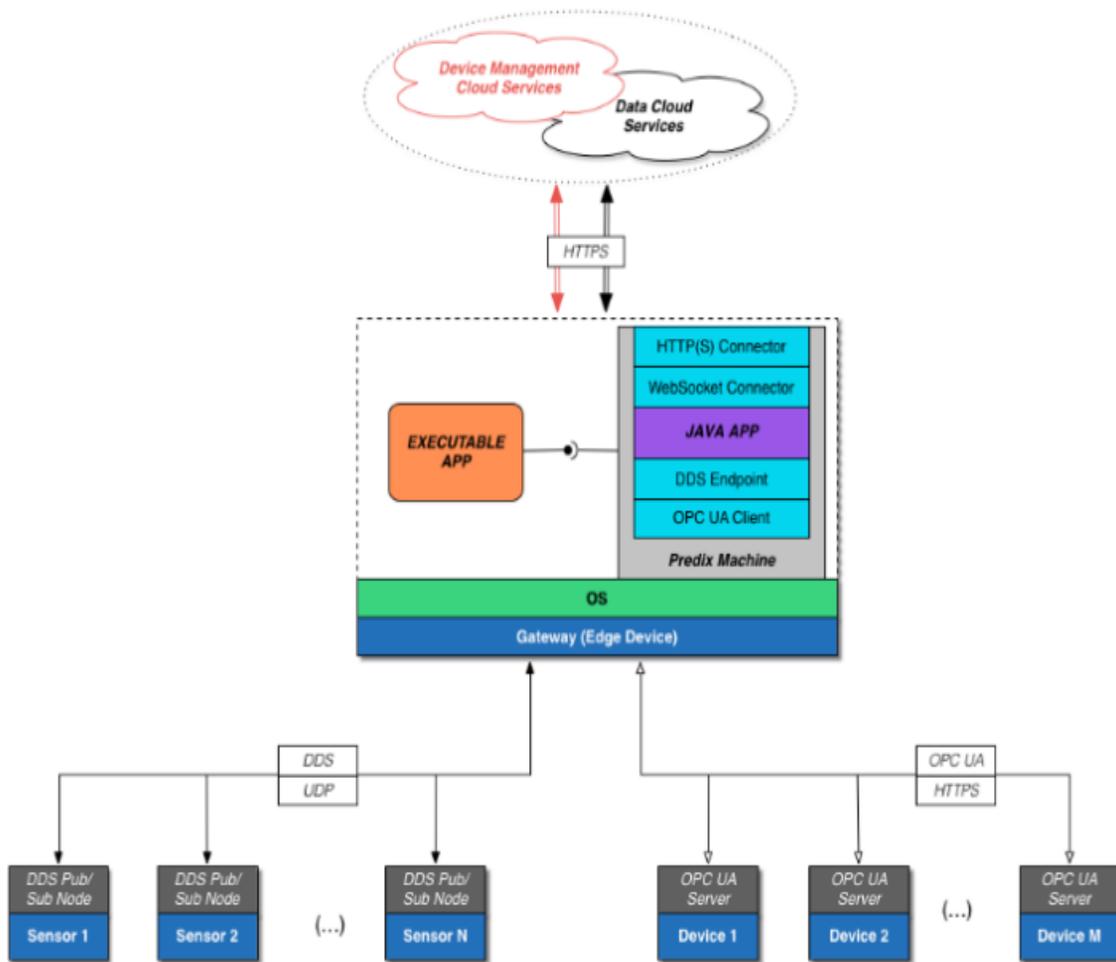
## *Why Should I use Predix Machine?*

Predix Machine is intended to do the following:

- Connect new and legacy assets to the Industrial Internet
- Create edge devices that act as Predix Gateways between assets and the Industrial Internet.
- Develop and deploy applications for the businesses and operators of Predix-enabled assets and gateways.

A sample architecture in which a device acts as a Predix Gateway is shown in the following picture.

Figure: Sample Architecture of a Predix Machine Gateway Use Case



This sample architecture depicts a use case in which two different groups of elements (sensors and devices) are exchanging data with an edge device that is using different protocols. In this sample, Data Distribution Service (DDS) is associated with the sensors and OLE for Process Control Unified Architecture (OPC UA) is associated with the devices. Predix Machine runs on an edge device; because the OPC UA Client and a DDS endpoint are both provided as services, the device itself can work as a data aggregator for devices and sensors. Moreover, Predix Machine provides the HTTP connector that allows the edge device to connect to the cloud and act as a gateway. The connection, which can be secured through TLS, also can be used to allow device management from the cloud. Predix Machine can provide a WebSocket connection to allow inter-process communication (IPC)

with a standalone executable application that is running outside of the Java stack, for example for legacy reasons.

## Quick Start: Getting Data from a Device to the Cloud

### *Quick Start: Getting Data From a Device to the Cloud*

Set up the transfer of HTTP transaction data from a Predix Machine-enabled device to the cloud.

1. Generate a Predix Machine runtime container.

For example, on the command line, enter:

```
GenerateContainers.sh -e /home/17.x.x/SDK/eclipse-jee-mars-SR2-win32-x86-64.zip -c PROV.
```

 **Note:** See [Generating a Predix Machine Runtime Container using Command Line Scripts \(page 33\)](#)

2. Install the HTTP Data service.
3. In the Event Hub River service, set the route host name of the HTTP Data service you deployed.
  - a. Navigate to `<Predix Machine runtime container location>/configuration/machine` and open the `com.ge.dspmicro.eventhubriver.send-0.config` file.
  - b. Set the `com.ge.dspmicro.httpriver.send.destination.host` property to the route host name.
4. Enroll Devices. See [Adding a Device to Predix Edge Manager \(page 67\)](#) and [Enrolling a Predix Machine-enabled Device with the Cloud \(page 68\)](#)
5. Set the connection information for the OPC-UA server or the Modbus subordinate device. If you do not have one, you can download Modbus simulators from <http://modbuspal.sourceforge.net>.
6. Based on the connection you established in the previous step, modify the XML file to remove the following comment lines.
  - Navigate to `<Predix Machine runtime container location>/configuration/machine`. In the `com.ge.dspmicro.machineadapter.modbus-0.xml` file, remove all lines marked `<!-- REMOVE THIS LINE FOR TCP/IP...`
  - Navigate to `<Predix Machine runtime container location>/configuration/machine`. In the `com.ge.dspmicromachineadapter.opcua-0.xml` file, remove all lines marked `<!-- REMOVE THIS LINE...`

 **Note:** For OPC-UA, set the `ServerUri` element to the host name of your OPC-UA server.

7. To activate the OPC-UA configuration:

a. Navigate to `<Predix Machine runtime container location>/configuration/machine` and open the `com.ge.dspmicro.machineadapter.opcua-0.config` file.

b. Set the following property:

```
com.ge.dspmicro.machineadapter.opcua.configFile=configuration/machine/
com.ge.dspmicro.machineadapter.opcua-0.xml.
```

8. Start the Predix Machine runtime container.

After Predix Machine is running, HTTP transaction information is displayed in the console. To verify that data is flowing into the PostgreSQL database, use the service's retrieve URL from a REST web client.

```
https://{Micro-Service-hostname}/v1/retrieve?transferId={copy a transferId
from console}
```

### Related concepts

[Predix Machine Technician Console-based Device Enrollment \(page 65\)](#)

[Device Enrollment Overview \(page \)](#)

### Related tasks

[Starting Predix Machine \(page 43\)](#)

[Generating a Predix Machine Runtime Container using Command Line Scripts \(page 33\)](#)

# Get Started with Predix Machine

## *Predix Machine SDK Overview*

The Predix Machine SDK is an Eclipse-based SDK, which has plugins for generating OSGI containers and scripts for adding Predix Machine to Docker containers.

The Predix Machine SDK allows you to generate your own Predix Machine runtime container by selecting certain feature groups that include all of the necessary bundles for that feature. You can also generate the container by selecting individual bundles. The grouping of the features is based on certain dependencies, as well as related features.

The following capabilities are included in all features:

- Predix Machine Runtime Container
- Felix Dependency Manager
- Declarative Services Support
- Jersey (jaxrs) Bundle
- MetaType-Configuration Management
- Logging
- Security Admin Service: SSL Certificate Management
- User Management Service (Account management)

The following table describes the Predix Machine feature groups.

**Table 2. Predix Machine Features**

Feature Groups	Description	Dependencies
Predix Machine Agent (Containerization)	<p>Allows Docker containers to communicate with each other.</p> <p> <b>Note:</b> This feature is required for development with the Edge SDK. This is one of the Docker containers running on the edge.</p>	<ul style="list-style-type: none"> <li>• Predix Machine Data Bus</li> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Predix Machine Provisioning Support</li> <li>• Predix Management Bus</li> <li>• Predix Connectivity</li> </ul>
Predix Machine Application Services	<ul style="list-style-type: none"> <li>• Git Repository Management Service</li> </ul>	Predix Web Tools
Predix Machine Cloud Gateway	<ul style="list-style-type: none"> <li>• Predix Cloud Identity Management.</li> <li>• Device Details – Updates device details to the cloud.</li> <li>• Command Framework – Sends commands to applications that are running on a device.</li> <li>• Package Handler – Use to configure the location for third-party package downloads.</li> </ul>	Predix Web Tools
Predix Machine Connectivity Management	Open VPN support when generating the container using scripts.	<ul style="list-style-type: none"> <li>• Predix Machine Provisioning Support</li> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> </ul>

Feature Groups	Description	Dependencies
Predix Machine Databus	Communicates data and control plane information between containers.	<ul style="list-style-type: none"> <li>• Predix Machine Gateway</li> <li>• Predix Web Tools</li> </ul>
Predix Machine Device Information	Gives dynamic and static information about the device.	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> </ul>
Predix Machine Event Hub River	Provides connectivity between Predix Machine and the Event Hub service.	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> </ul>
Predix Machine Gateway	<ul style="list-style-type: none"> <li>• OPC-UA Adapter</li> <li>• Modbus Adapter</li> <li>• Healthmonitor Adapter</li> <li>• Hoover</li> </ul>	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Store and Forward</li> <li>• Event Hub River</li> </ul>
Predix Machine HTTP Tunnel	Facilitates communication of different network protocols through HTTP/HTTPS.	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> </ul>
Predix Machine MQTT Support	<p>Publishes messages to a broker or subscribes to a topic to receive messages.</p> <ul style="list-style-type: none"> <li>• MQTT River</li> <li>• MQTT Client</li> </ul>	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> </ul>
Predix Machine Management Bus	Command and configuration support for containers.	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> </ul>
Predix Machine OPC-UA Server	Allows Predix Machine-enabled applications to expose data through the OPC-UA protocol, a common machine-to-machine protocol.	Predix Web Tools

Feature Groups	Description	Dependencies
Predix Machine Provisioning (Device Management and Enrollment)	<ul style="list-style-type: none"> <li>• Remote device management</li> <li>• Provisioning support when generating the container using scripts (PROV)</li> <li>• Bundles to handle ZIP support</li> </ul> <p> <b>Note:</b> See <a href="#">Edge Manager</a> documentation for more information.</p>	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> </ul>
Predix Machine Store and Forward	Forwards data to the cloud and continuously stores data to prevent data loss.	
Predix Machine Technician Console	Use the Predix Machine Technician Console to enroll devices after an administrator or operator has added the device to Predix Edge Manager.	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> </ul>
Predix Machine Web Console	Predix Machine Web Console and Bundle Updates. Includes the Felix HTTP bundle to add REST server ability to the container.   <b>Note:</b> Should only be used for debugging	Predix Web Tools
Predix Machine Web Tools	<ul style="list-style-type: none"> <li>• JSON support with JAX-RS</li> <li>• HTTP services for REST support</li> </ul>	
Predix Machine WebSocket River	WebSocket River provides connectivity between a Predix Machine-enabled device and the Predix Time Series service in the cloud.	<ul style="list-style-type: none"> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Predix WebSockets</li> </ul>
Predix Machine WebSockets	<ul style="list-style-type: none"> <li>• WebSocket Server</li> <li>• WebSocket Client Service</li> </ul>	

The following table describes which feature groups are included in the Predix Machine image.

**Table 3. Predix Machine Images**

Image	Included Feature Groups
Predix Machine Agent	<ul style="list-style-type: none"> <li>• Predix Machine Agent</li> <li>• Predix Machine Data Bus</li> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Predix Machine Provisioning</li> <li>• Predix Management Bus</li> <li>• Predix Connectivity</li> <li>• Predix Machine Device Information</li> <li>• Predix Machine Store and Forward</li> <li>• Predix Machine Web Console</li> </ul>
Predix Machine Agent Debug	All feature groups except for OPC-UA Server.
Predix Machine Debug	All feature groups except for OPC-UA Server.
Predix Machine Basic	No feature groups are included.
Predix Machine Connectivity	<ul style="list-style-type: none"> <li>• Predix Machine Connectivity</li> <li>• Predix Machine Provisioning</li> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Predix Machine Web Console</li> <li>• Predix Machine Device Information</li> </ul>
Predix Machine Provision	<ul style="list-style-type: none"> <li>• Predix Machine Provisioning</li> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Predix Machine Web Console</li> <li>• Predix Machine Device Information</li> </ul>
Predix Machine Default	<ul style="list-style-type: none"> <li>• Predix Machine Provisioning</li> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Predix Machine Web Console</li> <li>• Predix Machine Event Hub River</li> <li>• Predix Machine Gateway</li> <li>• Predix Machine MQTT Support</li> <li>• Predix Machine Store and Forward</li> <li>• Predix Machine WebSocket River</li> <li>• Predix Machine Device Information</li> </ul>
Predix Machine Tech	<ul style="list-style-type: none"> <li>• Predix Machine Technician Console</li> <li>• Predix Machine Provisioning</li> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Predix Machine Web Console</li> <li>• Predix Machine Store and Forward</li> </ul>

Image	Included Feature Groups
Predix Machine Tunnel	<ul style="list-style-type: none"> <li>• Predix Machine Provisioning</li> <li>• Predix Web Tools</li> <li>• Predix Cloud Gateway</li> <li>• Predix Machine Web Console</li> <li>• Predix Machine Device Information</li> <li>• Predix Machine HTTP Tunnel</li> </ul>

### Related concepts

[Data Bus Overview \(page 255\)](#)

[Management Bus \(page 265\)](#)

## Requirements

### Java JDK or JRE 8

Make sure the JRE includes the keytool in the path.

### Environment Variables

The following environment variable must be set for use by Predix Machine scripts:

`JAVA_HOME` – Java Virtual Machine location used with Predix Machine.

 **Note:** Do not use a trailing backslash or add quotes to the `JAVA_HOME` environment variable.

### Software Packages

The following software packages are required to use the Predix Machine SDK.

- Eclipse ( [Mars](#), [Neon](#), or [Oxygen](#) 64-bit versions) with the Plug-in Development Environment (PDE), for example Eclipse IDE for Java EE Developers.

 **Note:** If you are using a different version of Eclipse for your development environment, you can install and use another Eclipse version to use for the Predix Machine SDK.

- Predix Machine SDK. See [Downloading the Predix Machine SDK \(page 24\)](#).
- Maven version 3.1 or above. To verify your Maven installation and version, on a command line, enter: `mvn -version`. You can download Maven [here](#).

## Required Linux Utilities

Because certain devices may run minimal or limited versions of various operating systems, some utility executables that are used in various scripts to start and run the Predix Machine container may not be installed. The following utility executables are required for running Predix Machine in a Linux environment.

Utility	Description	Utility	Description
<b>awk</b>	Executes pattern-matching operations on data	<b>kill</b>	Sends a signal to a process
<b>cat</b>	Catenates	<b>nohup</b>	Prevents commands from aborting if you exit
<b>chmod</b>	Changes permissions of files or directories	<b>printf</b>	Prints a formatted string
<b>command</b>	Verifies if a command exists	<b>ps</b>	Displays process status
<b>date</b>	Prints or changes time and date	<b>readlink</b>	Prints value of symbolic link or file name
<b>dirname</b>	Strips last part of a filename	<b>sed</b>	Filters and transforms text
<b>echo</b>	Repeats typed text sent to a peripheral or	<b>sleep</b>	Delays or pauses
<b>find</b>	Searches text in a file	<b>sudo</b>	Allows you to execute commands as another user
<b>getopts</b>	Parses command line arguments	<b>systemctl</b>	systemd utility that controls the systemd system and service manager
<b>grep</b>	Processes text line by line and prints lines that match specified patterns	<b>tr</b>	Stops processor to wait for further instruction
<b>head</b>	Outputs first part of files	<b>trap</b>	Translates sets of characters
<b>jar</b>	Manipulates Java Archive (JAR) files	<b>uname</b>	Prints system information
<b>java</b>	Starts a Java application	<b>unzip</b>	Used to extract compressed files
<b>keytool</b>	Creates private keys		Required as part of the Java runtime

## Memory Requirements

Predix Machine does not provide memory management-related directives to the Java Virtual Machine (JVM), allowing the *Ergonomics* feature in the JVM to make intelligent choices that it can tune dynamically. The JVM makes these choices based on the class of the server Predix Machine is installed on, which in turn is determined by the total available memory, the number of CPUs, and platform architecture (32-bit or 64-bit).

In the absence of explicit command line parameters specifying memory allocation, the JVM determines the minimum and maximum heap sizes at start-up, and ensures that the usage stays between these limits, growing and shrinking the committed heap allocation as necessary. For example, for Java 8, set the minimum heap size to 1/64 of available physical memory, and the maximum heap size to 1/4 of available physical memory up to 1 GB, for a 32-bit system with two or

more CPUs and 2 or more GB of RAM. The default maximum heap size can be up to 32 GB on a 64-bit system with 128 GB RAM or more.

Therefore, on a 64-bit Linux server with 64 GB RAM and 16 CPUs, Predix Machine (or any Java process that does not explicitly specify heap parameters) will be given a minimum heap size of 1GB and maximum heap size of 16 GB. On the other hand, Predix Machine running on a smaller device such as a Raspberry PI with 434 MB RAM will be given a minimum heap size of about 7 MB and maximum heap size of about 110 MB. Predix Machine has been found to operate well using the defaults on a variety of systems, including Raspberry PI.

Unless the situation demands otherwise, it is best practice to leave the heap configuration and tuning to the JVM. However, it is possible that the operating characteristics of application bundles running under Predix Machine may require more heap space than what is allocated by default. Heap space also depends on the features you selected in the Predix Machine container. Additionally, the heap usage may need to be reduced due to other applications running on the system.

 **Note:** By default, Predix Machine starts with a minimum memory requirement of 75 MB. Running with a max set to less than 125 MB is not supported.

## Memory Footprint for Predix Machine Features

The following table shows the memory footprint of each of the Predix Machine Features.

Feature	Memory usage while container is running (MB)
Predix Cloud Gateway	57.3
Predix Connectivity Management with ODE	46.4
Predix Connectivity Management without ODE	46.8
Predix Machine Data Bus	29.1
Predix Device Information	58.4
Predix Event Hub River	47.6
Predix HTTP Tunnel	59
Predix Machine Gateway	36.5
Predix Machine Management Bus	26.7
Predix MQTT Support	34.5
Predix OPC-UA Server	188
Predix Provisioning	161.7
Predix Store and Forward	106.2
Predix Technician Console	73.5

<b>Feature</b>	<b>Memory usage while container is running (MB)</b>
Predix Web Console	72.5
Predix Web Tools	58.1
Predix WebSocket River	47.4
Predix WebSockets	41.6

## Memory Footprint for Docker Containers

The following table shows the memory footprint of each of the Predix Machine Docker containers on Ubuntu 14.04.

<b>Container</b>	<b>Memory usage while container is running</b>	<b>Image Size</b>
Bootstrap Loader	~1MB	13.63MB
Predix Machine	~200MB	186.90MB
Mosquitto	~20MB	15.42MB
C++ SDK Sample Subscriber	~1MB	10.12MB
C++ SDK Sample Publisher	~1MB	10.12MB

## Verified Platforms

Predix Machine is verified to run on the following platforms.

<b>System/Brand</b>	<b>Operating System</b>	<b>Example Specification</b>
Apple MacBook	MacOS X 10.9.4  <b>Note:</b> For development environment only.	<ul style="list-style-type: none"> <li>• Intel Core i7 2.3GHz</li> <li>• 16GB RAM</li> </ul>
Linux Server	GNU/Linux 3.8.13-35.1.2.el6uek.x86_64	<ul style="list-style-type: none"> <li>• Intel Xeon<sup>®</sup> CPU X5650 @ 2.67 GHz</li> <li>• 12GB RAM</li> <li>• 6Core</li> </ul>

System/Brand	Operating System	Example Specification
Linux VM	GNU/Linux 2.6.32-400.29.2.el6uek.x86_64	<ul style="list-style-type: none"> <li>• Intel Xeon CPU X7550 @ 2.00GHz</li> <li>• 8GB RAM</li> </ul>
Intel Galileo		<ul style="list-style-type: none"> <li>• 400MHz 32-bit Intel Pentium ISA-compatible</li> <li>• 512MB on-die SRAM, 256</li> </ul>
Raspberry Pi	Linux ARM6, 3.10.25 Raspbian	<ul style="list-style-type: none"> <li>• ARMv6-compatible processor rev 7 (v6l)</li> <li>• 448180KB RAM</li> </ul>
Intel Kontron M2M	GNU/Linux 2.6.34.10-WR4.3.0.0_standard	<ul style="list-style-type: none"> <li>• Intel Atom<sup>®</sup> CPU E640 @ 1.00 GHz</li> <li>• 1 GB RAM</li> </ul>
Beagle Bone		<ul style="list-style-type: none"> <li>• AM335x 720MHz ARM Cortex-A8</li> <li>• 256MB DDR2 RAM</li> </ul>
HP ProLiant DL320 Server	Centos 6.7 and JDK 1.7	<ul style="list-style-type: none"> <li>• Intel Xeon 3.60 GHz</li> <li>• 32 GB RAM</li> </ul>
Qualcomm DragonBoard 410	Ubuntu-based Linux, Oracle JDK 1.7	<ul style="list-style-type: none"> <li>• ARM Cortex A53 1.2 GHz</li> <li>• 1 GB RAM</li> </ul>
GEIP MFA	Pengutronix Linux, Oracle Embedded JDK 1.7	<ul style="list-style-type: none"> <li>• ARM 600 MHz</li> <li>• 512MB RAM</li> </ul>
GEIP PAX Rxi	Win7 Pro, Oracle JDK 1.7	<ul style="list-style-type: none"> <li>• VIA Eden X2 U4200 1.2 GHz</li> <li>• 4 GB RAM</li> </ul>
Intel MoonIsland on Advantech UTX 3115	Wind River Linux, JDK 1.7	<ul style="list-style-type: none"> <li>• Intel Atom Dual Core E3826 1.46 GHz</li> <li>• 2 GB RAM</li> </ul>

System/Brand	Operating System	Example Specification
Cisco829	VM Running on Linux, Oracle JDK/ Open JDK 1.7	<ul style="list-style-type: none"> <li>• Intel Atom Dual-Core Rangeley</li> <li>• CPU 1250 MHz</li> <li>• 2GB RAM</li> </ul>

 **Note:** Windows is not a supported platform with Predix Machine 17.2.x and later.

## Downloading the Predix Machine SDK

You must have a Predix US-West account to access the download site.

To begin using the SDK, you must first download the SDK package.

This is a test paragraph.

1. Access the Predix Machine SDK download at <https://artifactory.predix.io/artifactory/webapp/#/artifacts/browse/tree/General/PREDIX-EXT/predix-machine-package/predixmachinesdk/17.2.5/predixmachinesdk-17.2.5.zip>.
2. Enter your Predix account credentials when prompted for a user name and password.
3. Download the `PredixMachineSDK-17.2.x.zip` file.
4. Unzip and extract all of the files in the ZIP file.

Generate a Predix Machine Provisioning container. (*page* )

### Related concepts

Accessing Artifactory Downloads (*page* )

### Related information

(*page* )

## Predix Machine Downloads

Download Predix Machine software packages.

### Prerequisites

Before downloading Predix Machine software packages, you must:

- Meet the software, memory and platform requirements as described in [Requirements \(page 19\)](#).

- Have a Predix account. If you do not have a Predix account, see [am\\_registering\\_for\\_predix\\_account.ditamap#IZDc2MmY1YjEtNjQ1NC00YTikLThlZDctYmM0NjBjODU1OQ](#)

## Predix Machine SDK

Package	Description
<a href="#">Predix Machine SDK 17.2.1</a>	See the <a href="#">Release Notes for Predix Machine 17.2.1 (page 309)</a> .
<a href="#">Predix Machine SDK 17.2.0</a>	See the <a href="#">Release Notes for Predix Machine 17.2.0 (page 311)</a> .
<a href="#">Predix Machine SDK 17.1.3</a>	See the <a href="#">Release Notes for Predix Machine 17.1.3 (page 314)</a> .

For installation instructions, see [Installing the Predix Machine SDK \(page 28\)](#).

## Predix Machine Edge SDK

Use the edge SDK to develop applications that run in Docker containers and communicate with the Docker container based Predix Machine through the Data Bus.

Package	Description
<a href="#">predixmachine-edgesdk-cpp/</a>	<p>Use the C++ edge SDK without a runtime, for C++ debug development environment, or for Java development.</p> <p> <b>Note:</b></p> <p>Artifacts are provided to support different runtime architectures (ARM and x86). Select the download that makes sense from the following options:</p> <ul style="list-style-type: none"> <li>• armhf-arm7 – Raspberry PI3 Architecture</li> <li>• x86_64 – 64-bit x86 architecture</li> </ul> <p>For the C++ edge SDK without a runtime, choose one of the following:</p> <ul style="list-style-type: none"> <li>• <code>predixmachine-edgesdk-cpp-alpine-linux-armhf-release-shared/</code></li> <li>• <code>predixmachine-edgesdk-cpp-alpine-linux-x86_64-release-shared/</code></li> </ul> <p>For debug development environment for C++ edge SDK, choose one of the following folders:</p> <ul style="list-style-type: none"> <li>• <code>predixmachine-edgesdk-cpp-rootfs-alpine-armhf/</code></li> <li>• <code>predixmachine-edgesdk-cpp-rootfs-alpine-x86_64/</code></li> </ul>
<a href="#">predixmachine-edgesdk-java/</a>	Java edge SDK.

## Predix Machine Docker Containers

Docker container components allow you to use Predix Machine in a containerization environment. To use Predix Machine in a complete containerization environment, you must download and install the following Docker container components, based on your environment:

- Predix Machine Agent for Docker/containerized environment
- Bootstrap Loader
- MQTT Broker

Package	Description
<a href="#">predixmachine-agent-armhf/</a>	
<a href="#">predixmachine-agent-armhf-jre7/</a>	
<a href="#">predixmachine-agent-armhf-jre8/</a>	
<a href="#">predixmachine-agent-armhf-jre8-openvpn/</a>	
<a href="#">predixmachine-agent-debug-x86_64-jre8/</a>	
<a href="#">predixmachine-agent-debug-x86_64-jre8-openvpn/</a>	
<a href="#">predixmachine-agent-x86_64/</a>	
<a href="#">predixmachine-agent-x86_64-jre7/</a>	
<a href="#">predixmachine-agent-x86_64-jre8/</a>	
<a href="#">predixmachine-agent-x86_64-jre8-openvpn/</a>	
<a href="#">predixmachine-alpine-armhf/</a>	
<a href="#">predixmachine-alpine-x86_64/</a>	
<a href="#">predixmachine-bootstrap-alpine-armhf/</a>	
<a href="#">predixmachine-bootstrap-alpine-x86_64/</a>	
<a href="#">predixmachine-bp-cpp-armhf/</a>	
<a href="#">predixmachine-bp-cpp-x86_64/</a>	
<a href="#">predixmachine-edgesdk-cpp-base-alpine-armhf/</a>	
<a href="#">predixmachine-edgesdk-cpp-base-alpine-x86_64/</a>	
<a href="#">predixmachine-mosquitto-armhf/</a>	
<a href="#">predixmachine-mosquitto-x86_64/</a>	
<a href="#">predixmachine-openjdk-jre-armhf/</a>	
<a href="#">predixmachine-openjdk-jre-x86_64/</a>	
<a href="#">predixmachine-openvpn-armhf/</a>	

Package	Description
<a href="#">predixmachine-openvpn-x86_64/</a>	
<a href="#">predixmachine-python2-armhf/</a>	
<a href="#">predixmachine-python2-x86_64/</a>	
<a href="#">predixmachine-python3-armhf/</a>	
<a href="#">predixmachine-python3-x86_64/</a>	

## Predix Machine Docker Quick Start App

The Docker Quickstart app provides an application that is running within a Docker container. You can download the Docker Quickstart app from <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/docker-quickstart/>.

### Related tasks

[Downloading the Predix Machine SDK \(page 24\)](#)

[Downloading the Predix Machine edge SDK \(page 248\)](#)

[Downloading Docker Images \(page 273\)](#)

## Licensing

You receive a non-production license key when you download the Predix Machine SDK. To obtain a production license key, contact Predix Machine product management.

The following table lists the non-production license key expiration dates for Predix Machine SDK versions.

Predix Machine SDK Version	Expiration Date
17.2.5	1/31/2020
17.2.4	1/31/2019
17.1.3	1/31/2019
17.1	1/31/2018
16.4	1/31/2018
16.3	1/1/2018
16.2	1/1/2017
16.1	1/1/2017

## *Predix Machine SDK Directory Structure*

When extracted, the downloaded SDK file creates the following directory structure:

Directory	Description
docs	<p>Contains the following documentation:</p> <ul style="list-style-type: none"> <li>• <code>Container_docs.zip</code> – SDK documentation.</li> <li>• <code>apidocs.zip</code> – Javadoc APIs.</li> <li>• <code>predixmachine-bom-&lt;Predix_Machine_version&gt;.txt</code> – lists the components, and their versions, used in the Predix Machine SDK.</li> </ul>
eclipse-plugins	<p>Indicates the location that you will point the Eclipse installation to. Includes the following folders:</p> <ul style="list-style-type: none"> <li>• <code>features</code></li> <li>• <code>plugins</code></li> </ul>
license	<p>Contains the license files.</p>
samples	<p>Contains <code>sample-apps.zip</code> and <code>sample-cloud-apps.zip</code> files for sample applications.</p>
utilities	<p>Includes the following folders:</p> <ul style="list-style-type: none"> <li>• <code>CompareContainer</code> – Contains a program that will compare two machine folders and build a download package for upgrading from one to the other.</li> <li>• <code>containers</code> – Contains scripts used for generating containers.</li> </ul>
InstallationGuide.pdf	<p>The <i>Predix Machine Software Development Kit Installation Guide</i>.</p>

## *Installing the Predix Machine SDK*

To install the Predix Machine SDK, follow these steps.

1. Open Eclipse.

The **Welcome to Eclipse** page appears.

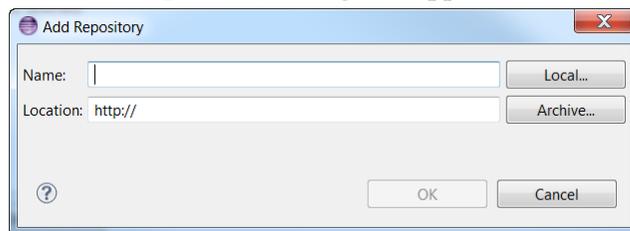
2. On the **Help** menu, select **Install New Software**.

The **Available Software** window appears.

 **Note:** If you just downloaded the latest version of Eclipse, you should clear the **Contact all updates sites during install to find required software** checkbox to prevent Eclipse from searching for updates for all installed packages. If you have an older version of Eclipse, you can select the checkbox to perform this procedure, but it may take a long time.

3. Click the **Add** button.

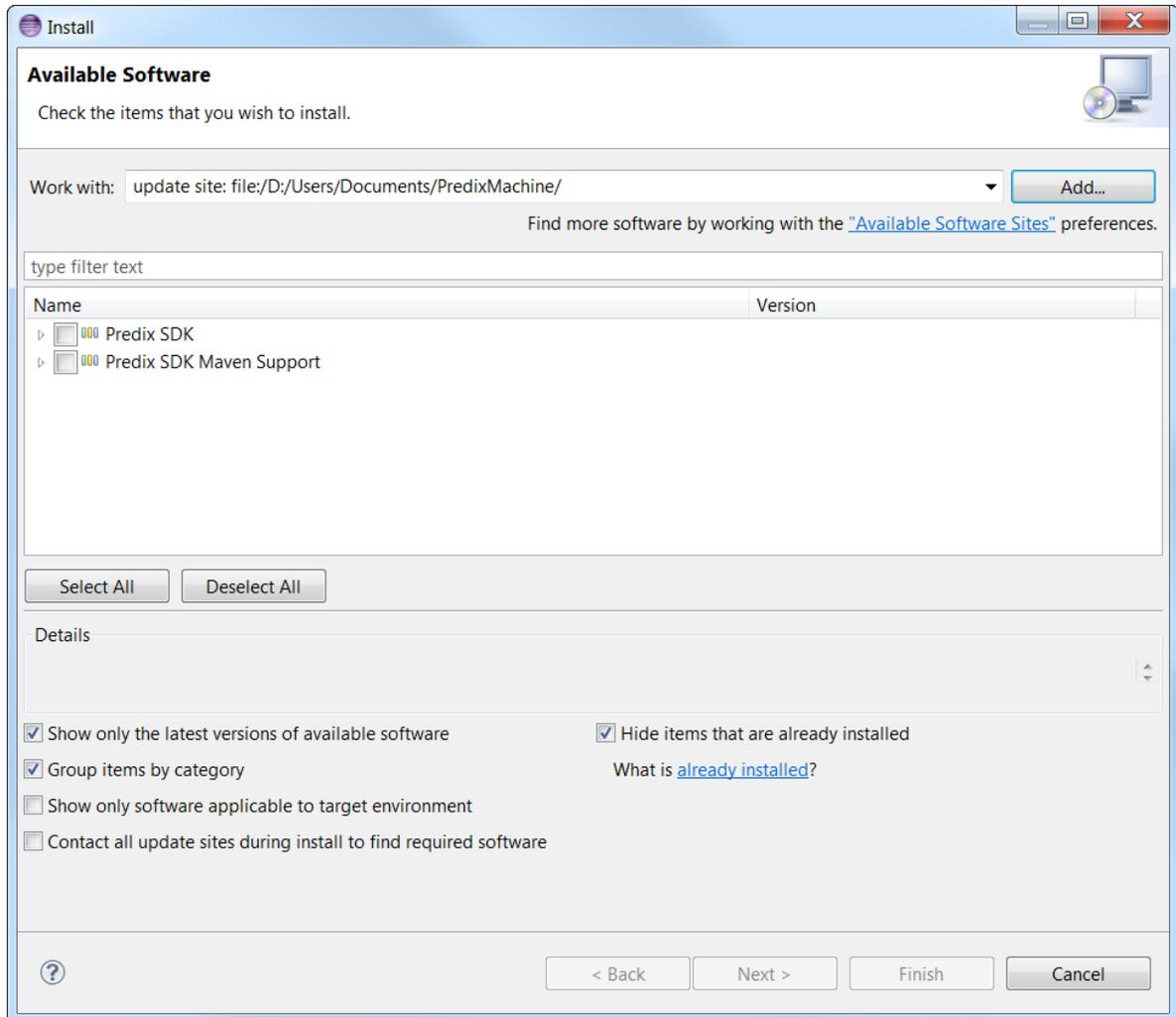
The **Add Repository** dialog box appears.



4. Click the **Local** button and then browse to the installation location where you unzipped the Predix Machine SDK files, select the **eclipse-plugins** folder, and click **OK**.

In the Add Repository dialog box, click **OK**.

The Predix Machine SDK and Predix Machine SDK Maven Support options now appear in the list of available software.



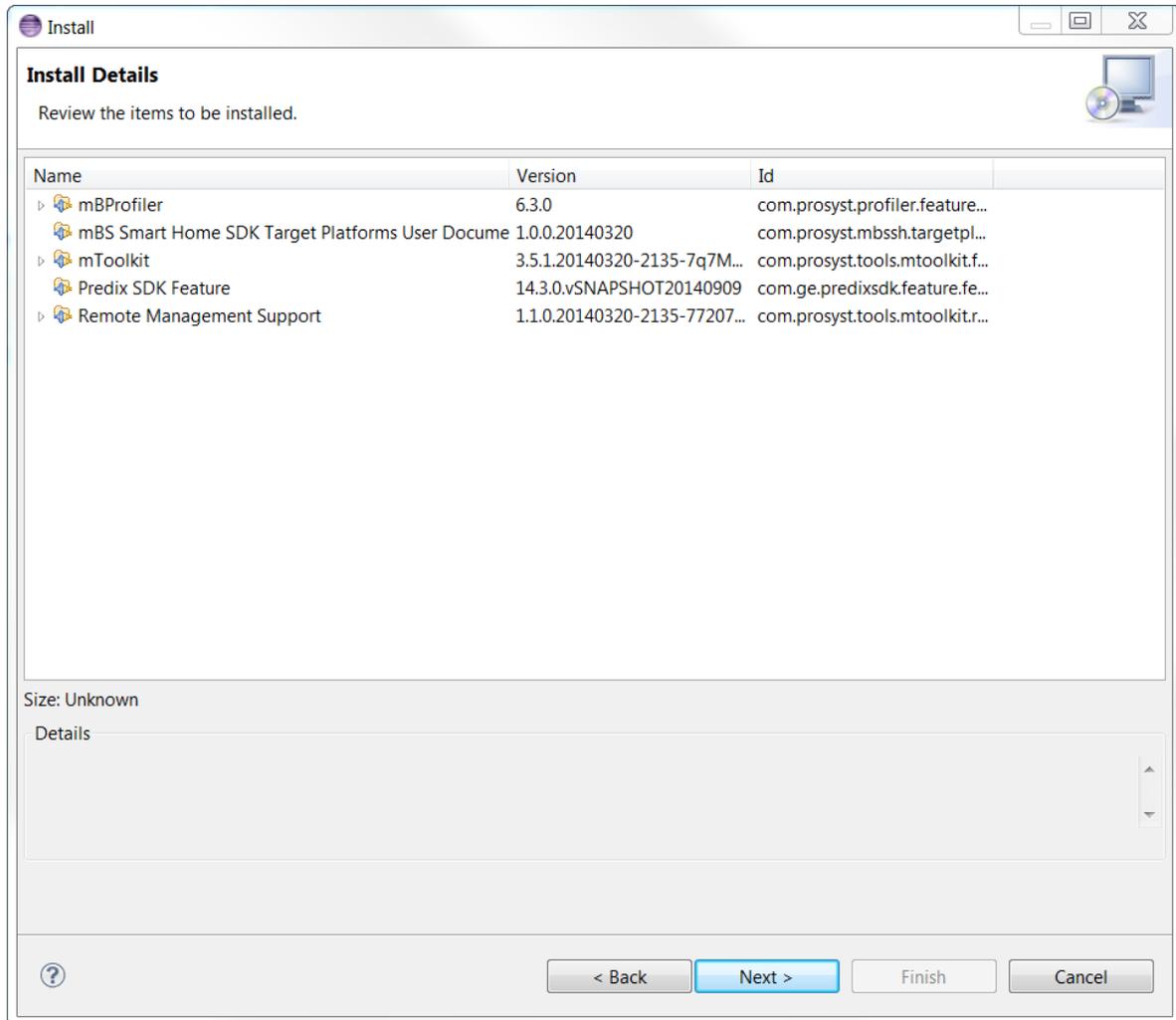
5. Select the following installation options:

- **Predix Machine SDK**
- **Predix Machine SDK Maven Support**

**Note:** Only select **Predix Machine SDK Maven Support** if you have installed the Maven Integration (m2eclipse) plugins. If M2E is not installed, the installation cannot continue until you install M2E or the Eclipse IDE for Java JEE Developers. This is available in the Eclipse Marketplace.

6. In the **Details** section, select the options you want to customize your installation. You can use the Default selections. Click the **Next** button.

The **Install Details** window appears.



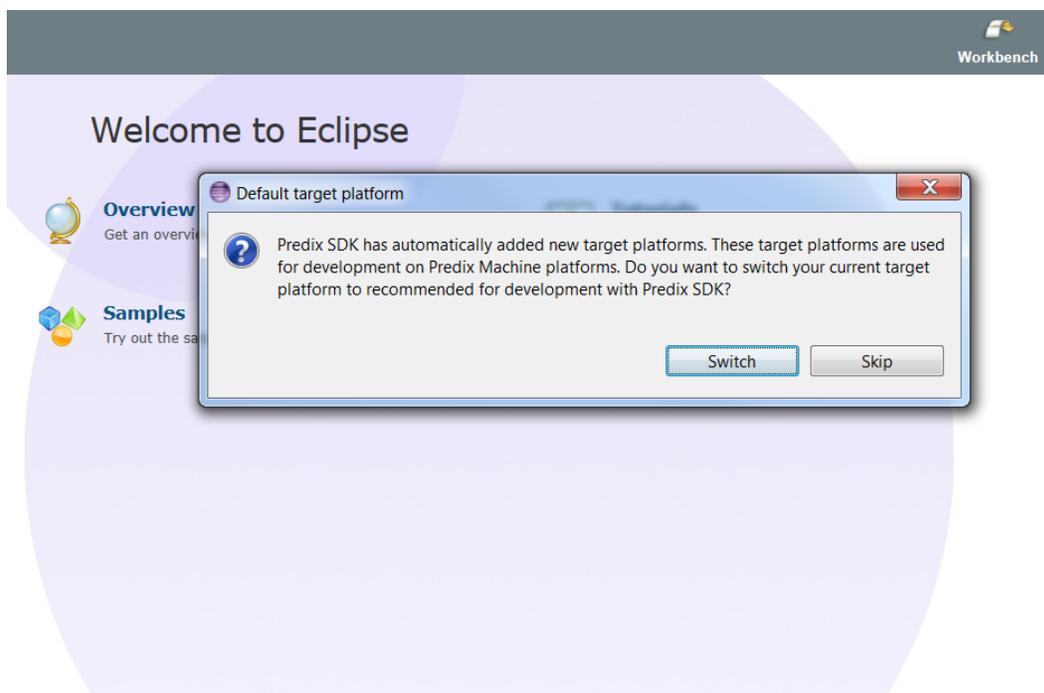
7. On the **Install Details** window, click **Next**.  
The **Review Licenses** window appears.
  8. Review the terms of the license agreements, then choose the **I accept the terms of the license agreements** option, and click **Finish**.  
The **Installing Software** progress bar appears.
-  **Note:** If a Security Warning appears, click **OK** to continue.
9. When the software installation is complete, click **Yes** to restart Eclipse.  
If you have not created a workspace for Eclipse, the **Select a Workspace** dialog box opens.  
Select a workspace location, and click **OK**.

## Starting Eclipse

When you start Eclipse after installing the Predix Machine SDK, Eclipse configures your environment to use the SDK. The default PDE configuration uses OSGi runtime components.

1. Start Eclipse.
2. If, during start-up, you are prompted to switch the Eclipse Plug-in Development Environment (PDE), click **Switch**.

 **Note:** If you do not switch, you can switch later.



## Bundle Migration

When upgrading Predix Machine versions, bundles may be renamed, refactored, or removed. For an easy upgrade, you can remove all bundles from your image and replace them entirely with the new bundles. You can add new bundles for specific features by selecting the corresponding **Bundle type** in the **Add Bundles** dialog box. See [Generating a Predix Machine Container Using Eclipse \(page 35\)](#) for instructions.

## *Generate Predix Machine Containers*

You can generate Predix Machine runtime containers by using either the Predix Machine SDK in Eclipse or a command line script. You can also create Predix Machine as a Docker image or convert an existing Predix Machine container to a Docker image.

### **Predix Machine Runtime Container Types**

The following types of Predix Machine runtime containers can be generated:

- **Predix Machine default container:** If you do not specify the container type, the default container is created.
- **Predix Machine Agent:** Provides predefined Predix Machine with agent feature for Docker support.
- **Predix Machine Agent Debug:** Provides Debug Predix Machine with agent feature for Docker support.
- **Predix Machine Debug:** Provides the Predix Machine Web Console.
- **Predix Machine Provision:** Provides provisioning support.
- **Predix Machine Custom:** A custom Predix Machine container using a custom image you created in Eclipse.
- **Predix Machine Tech:** Technician Console image.
- **Predix Machine Connectivity:** Provides connectivity support.
- **Predix Machine Tunnel:** Provides HTTP tunneling support.

The methods by which you specify the container type differ depending on whether you use the Predix Machine SDK in Eclipse, or if you use a command line script.

#### **Related tasks**

[Generating a Predix Machine Runtime Container using Command Line Scripts \(page 33\)](#)

[Generating a Predix Machine Runtime Container Using Eclipse \(page 35\)](#)

## *Generating a Predix Machine Runtime Container using Command Line Scripts*

- Download Predix Machine from <https://artifactory.predix.io/artifactory/webapp/#/artifacts/browse/tree/General/PREDIX-EXT/predix-machine-package/predixmachinesdk/17.2.5/predixmachinesdk-17.2.5.zip>.
- Download Eclipse with PDE runtime plug-ins; for example Eclipse IDE for Java EE Developers. This download should remain in the .zip or tar.gz format.
- Ensure that you have Maven installed. On a command line interface, type `mvn -version`. Your version should be 3.1 or above.

Use the following commands in the command line to generate a Predix Machine runtime container:

- `-e <ECLIPSE_PATH>`: Path of downloaded Eclipse ZIP/TAR file.
- `-c <CONTAINER_TYPE>`: Type of Predix Machine container to create.

 **Note:** For information on generating Predix Machine as a Docker image, see [Generating a Predix Machine as a Docker Image using Command Line Scripts \(page 279\)](#).

You can generate the following types of containers:

- AGENT: Predix Machine with agent feature for Docker support. This container is required for running in the Dockerized model to use with the edge SDK.
- AGENT\_DEBUG: Predix Machine Debug with agent feature for Docker support.
- PROV: Predix Machine Provision (includes only the JAR bundles that support provisioning).
- DEBUG: Predix Machine Debug with Predix Machine Web Console.
- TECH: A Technician Console image.
- CONN: Predix Machine Connectivity image.
- TUNNEL: A Predix Machine container with HTTP tunneling.
- CUSTOM `<image file full path>`: A Predix Machine container using a custom image you created in Eclipse.
- [not specified]: Predix Machine default container.

 **Note:** Each of these container types maps to the **Workspace Image** selection in the **Image Description** dialog box in the Predix SDK in Eclipse.

1. Open a terminal window.
2. In the command line, navigate to the `<Predix Machine SDK download location>/predixmachinesdk-17.2.x/utilities/containers` folder.
3. Run the following command:

```
GenerateContainers.sh -e <full path and name of downloaded Eclipse.tar.gz file> -c <type of container>
```

For example:

```
GenerateContainers.sh -e /home/17.x.x/SDK/eclipse-jee-mars-SR2-win32-x86-64.zip -c PROV
```

The script creates the Predix Machine runtime container in the `<Predix Machine SDK download location>/predixmachinesdk-17.2.x/utilities/containers` folder.

 **Note:** If you receive the following console error, you can ignore it:

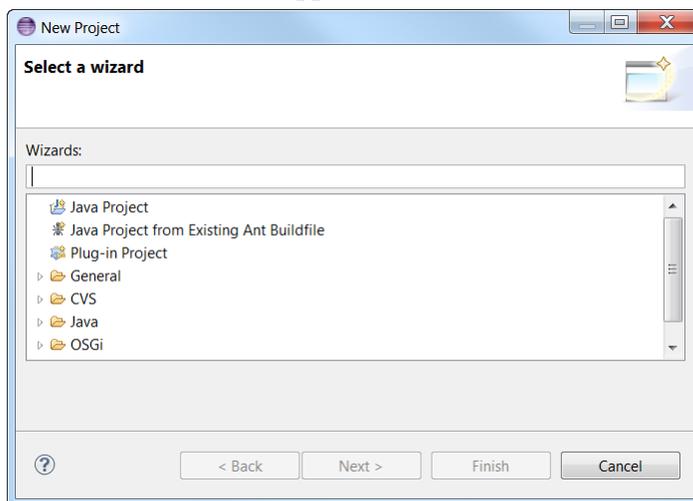
```
java.lang.ClassCastException:
org.eclipse.osgi.internal.framework.EquinoxConfiguration$1 cannot be cast
to java.lang.String at
org.eclipse.m2e.logback.configuration.LogHelper.logJavaProperties(LogHelper.java:26)
```

```
at
org.eclipse.m2e.logback.configuration.LogPlugin.loadConfiguration(LogPlugin.java:189)
at
org.eclipse.m2e.logback.configuration.LogPlugin.configureLogback(LogPlugin.java:144)
at org.eclipse.m2e.logback.configuration.LogPlugin.access
$2(LogPlugin.java:107) at org.eclipse.m2e.logback.configuration.LogPlugin
$1.run(LogPlugin.java:62) at
java.util.TimerThread.mainLoop(Timer.java:555) at
java.util.TimerThread.run(Timer.java:505)
```

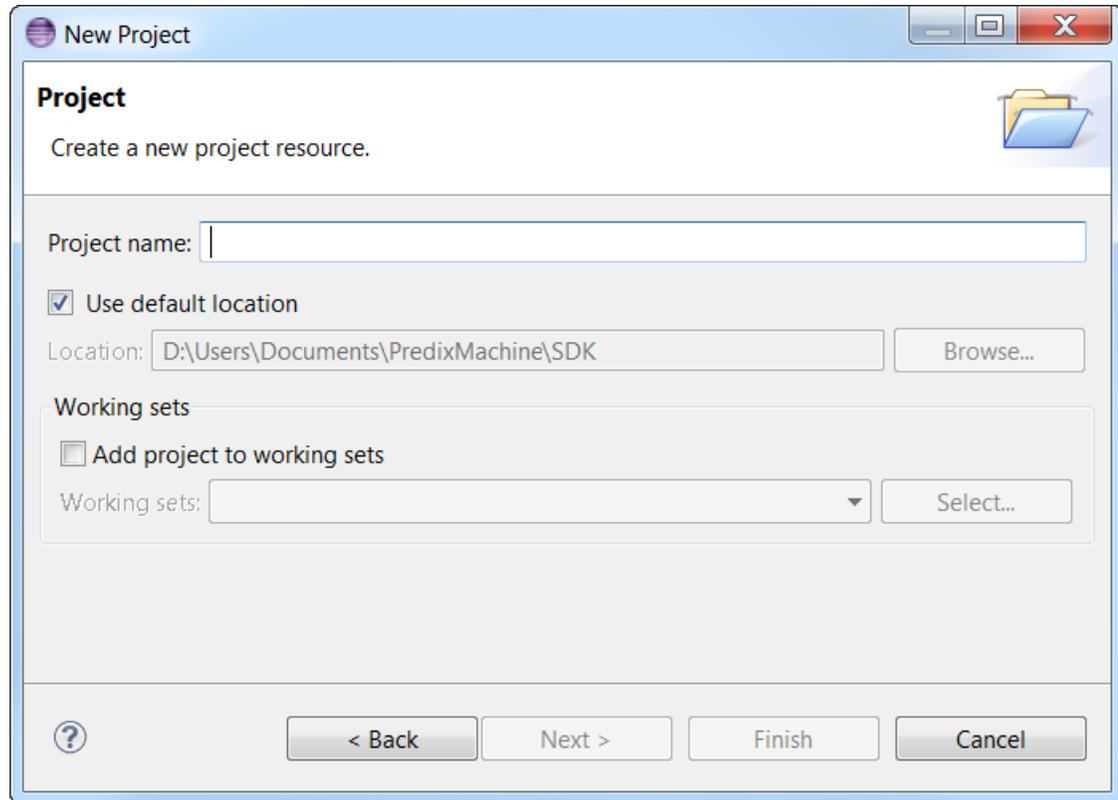
## *Generating a Predix Machine Runtime Container Using Eclipse*

Follow these steps to generate a Predix Machine runtime container using Eclipse.

1. In Eclipse, select **File > New > Project** to create a project in which to create files and folders. The **New Project** box appears.



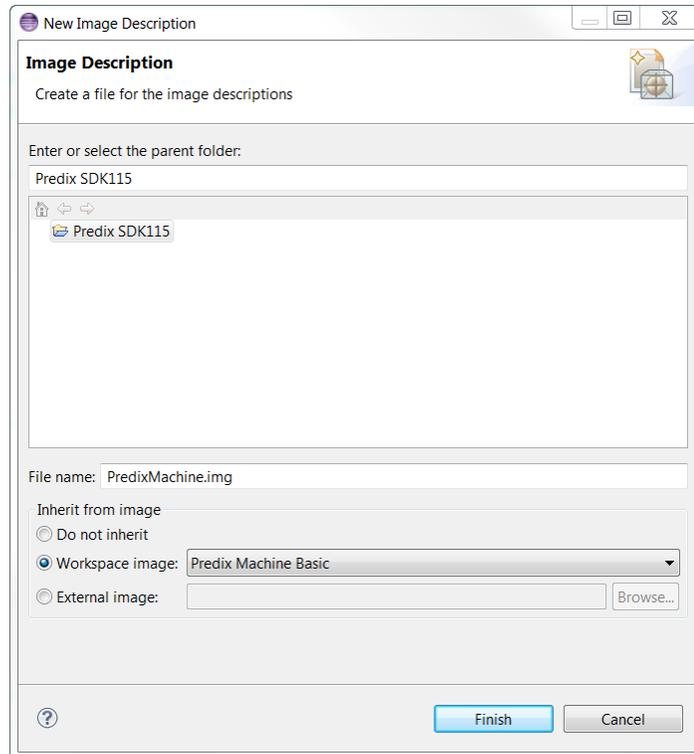
- a. In the New Project dialog, select **General > Project** and click **Next**. The **Project** page appears in the **New Project** wizard.



- b. In the **Project** page of the New Project wizard, assign a **Project name** to your project and click **Finish**.  
Your project appears in the **Navigator** pane.

2. Create a Predix Machine SDK image file:

- a. Right-click in the **Navigator** panel, and choose **New > Image Description**.  
The **New Image Description** window appears.



- b. In the **File name** box, type a name for the file.
- c. In the **Inherit from image** section, choose the container type from the drop-down list, and click **Finish**.

For information about the different container types, see [Predix Machine SDK Overview \(page 14\)](#).

The image is created.

### 3. Add bundles to the container:

- a. In the **Bundles** section, click the **Add** button.  
The **Add bundles** window appears.
- b. In the **Bundle type** list, select **Predix Features Group** to add bundles necessary for the various features.  
A list of each of the Predix features appears. See [Predix Machine SDK Overview \(page 14\)](#) to see the list of Predix Machine bundles that are part of each Predix Machine feature.

**Note:** The Predix Machine SDK will prompt you to update image editor files when you open the image files. During the update, it will update older versions to current versions.

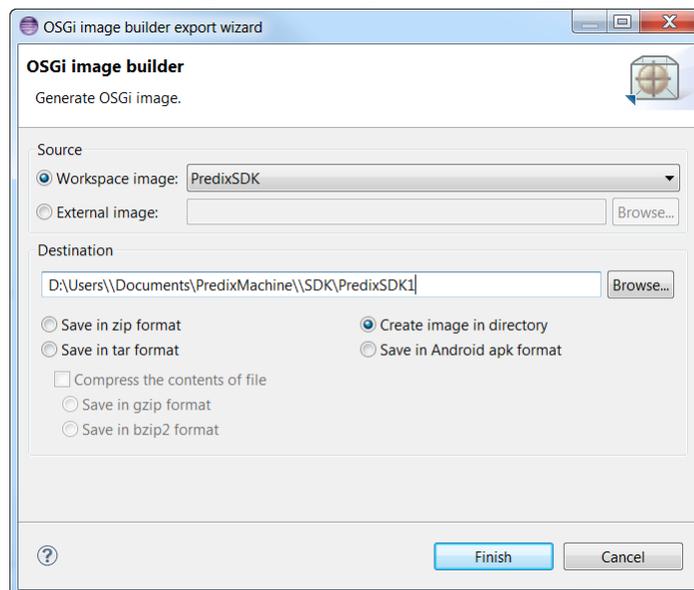
- c. Select the features to include in the container and click **OK**.

**Note:** If you are adding a Maven bundle to the image, Eclipse will create an absolute path for the image. Edit the image file as text to change the absolute path to a relative path. For example:

```
<element type="com.prosyst.tools.builder.osgiBundle"
  location=maven:rel:../../../../../../gut/PredixMachine/master/machine-ge/ge-httpclient/pom.xml
```

#### 4. Generate the container:

- a. In the **Overview** section of your project, click the **Export** image link. The **OSGi image builder export wizard** appears.



- b. In the **Workspace image** list, select the image you created in step 2.
- c. In the **Destination** box, browse to the target location for the container and click **Finish**. The wizard builds the container and stores it in the target location.

**Note:** You can also use the **Run**, **Debug**, and **Profile** options instead of **Export**. These enable you to test services that you may want to add.

Profiling sessions are limited to between 5 and 10 minutes.

When using Eclipse on Linux, the profiler does not work correctly unless `LD_LIBRARY_PATH` is set before launching Eclipse. This must be set to include `<workspace root>/metadata/.plugins/com.prosyst.tools.mbsemulator/images/<image name>/osgi/lib/`

`mbprofiler-agent/runtimes/linux-x86_64-generic` where `<workspace root>` is the location of the Eclipse workspace in use and `<image name>` is the name of the image file you created.

5. Start the generated container:
  - a. On your computer, navigate to the `<exported location>/bin/` folder.
  - b. Run the `start_predixmachine.sh` shell script.

## *Generating a Predix Machine Container for Development on Windows*

Windows, as a container, is supported in the run/debug under Eclipse only.

You can generate a Predix Machine container on Windows, but Predix Machine 17.2.x and later does not support running the Predix Machine container in a production environment. So, for example, you can generate a provisional container in Eclipse on Windows.

The SDK environment under Windows is supported for building Predix Machine containers, but in order to connect and enroll with the cloud you must run the `bin/start_predixmachine.sh` script, which starts the yeti process, which is required for device enrollment. You can use the Docker solution on Windows for testing with the cloud.

1. In Eclipse, generate the Predix Machine container as described in [Generating a Predix Machine Runtime Container Using Eclipse \(page 35\)](#).
2. In Platform Settings, select **Windows/x86\_64**.
3. Start the generated container:
  - a. On your computer, navigate to the `<Location_of_exported_image>/machine/bin/predix` folder.
  - b. Double-click the `start_container.bat` script.

## *Predix Machine Directory Structure*

The generated Predix Machine container has the following directory structure:

Directory	Description
appdata	<p>As a best practice, services should have read and write access to only this folder. Contains files that user has uploaded through the Predix Machine Web Console/Technician Console. Contains the following subdirectories:</p> <ul style="list-style-type: none"> <li>• <code>GitRepositories</code></li> </ul> <p>Contains the Git bundle used for creating and replicating Git repositories. All Git repositories should be stored in this directory.</p>

Directory	Description
bin	Contains the <code>start_predixmachine</code> and <code>stop_predixmachine</code> scripts for starting and stopping Predix Machine.
configuration	Configuration files for bundles. Containers the following subdirectories: <ul style="list-style-type: none"><li>• machine Configuration files for Predix Machine bundles and any custom bundles</li><li>• yeti The <code>yeti.cfg</code> file containing the connection cycle count parameters read by the <code>start_yetiscript</code>.</li></ul>
installations	Yeti process monitors this folder for software to install. Software must be a ZIP file with an <code>install.bat/install.sh</code> file inside.
license	End user license agreement and a list of open source software.
logs	Runtime logs. Contains the following subdirectory: <ul style="list-style-type: none"><li>• machine Runtime logs for Predix Machine</li><li>• installations Yeti installation logs</li></ul>

Directory	Description
machine	<p>Container directory which includes the following subdirectories:</p> <ul style="list-style-type: none"> <li>• bin/predix/ <ul style="list-style-type: none"> <li>Start container and stop container scripts</li> </ul> </li> <li>• bin/vms <ul style="list-style-type: none"> <li>The solution.ini and all all other ini files</li> </ul> </li> <li>• bundles <ul style="list-style-type: none"> <li>Samples should be copied here.</li> </ul> </li> <li>• configs <ul style="list-style-type: none"> <li>OSGi XML file contained in bundle com.prosyst.mbs.osgi.metatyp.bundle</li> </ul> </li> <li>• install <ul style="list-style-type: none"> <li>Install scripts</li> </ul> </li> <li>• lib/framework/*.jar <ul style="list-style-type: none"> <li>Contains runtime libraries.</li> </ul> </li> </ul>
security	<p>Security policies and keystores. In a deployment environment, access to this folder should be restricted to users with administration rights.</p>
yeti	<p>Contains <code>service_installation</code> folder with Predix Machine as a service start and install scripts. Also includes files and scripts that help Yeti act as a watchdog.</p> <p> <b>Note:</b> For Provisioning containers only.</p> <p> <b>Important:</b> Do not move any files that are in this folder.</p>

## User Permissions and Roles

To use Predix Machine, you must have a Predix account to access the Predix Machine SDK download site.

To use Predix Machine, you must meet the technical requirements outlined in the *Requirements* section. See [Requirements \(page 19\)](#).

Once you have downloaded the Predix Machine SDK, certain users/roles and permissions are recommended to secure the process of creating and running a Predix Machine runtime container, and provisioning devices to the cloud through Edge Manager.

The following table lists the various users/roles and settings that are used and includes default or recommended user names for certain roles.

User/ Role	User Name (if applicable)	Description/Functions	Related Documentation
Predix.io/ Cloud Foundry user	(your Predix.io log in)	Allows user to download the Predix Machine SDK.	<p>(page )</p> <p><a href="#">Downloading the Predix Machine SDK (page 24)</a></p> <p><a href="#">Configure User Proxy and Repository settings (page 286)</a></p>
Predix Machine administrator	Initially, <code>predix</code> . Must change password on first login.	Creates users, configures web console plug-ins, changes Predix Machine Web Console log in. Creates <code>predixuser</code> secure user for protecting <code>Security</code> folder.	<p><a href="#">Managing Users (page 115)</a></p> <p><a href="#">Configuring Predix Machine Web Console Plug-ins (page 51)</a></p> <p><a href="#">Management Service (Users) (page 115)</a></p>
Secure user	<code>predixuser</code>	<p>User who is not an administrator or root user.</p> <p>Secures <code>Security</code> folder. Only user who has read/write access to <code>Security</code> folder.</p> <p>Runs Predix Machine containers.</p> <p> <b>Important:</b> As a best practice, this user with limited administrative privileges should run the container. A root/administrator should not run Predix Machine.</p>	<p><a href="#">Securing Predix Machine (page 52)</a></p>
Edge Manager Administrator		Administrative access and permissions. Creates groups, imports devices, creates users, and assigns roles to users, and changes user passwords.	<p>Creating Users and Assigning User Roles (page )</p> <p>About Predix Edge Manager Groups (page )</p>
Edge Manager Operator	(assigned by Edge Manager administrator)	Accesses Device Manager and Repository in Edge Manager.	<p>Device Manager Overview (page )</p> <p>Uploading Software and Configuration Packages to the Predix Edge Manager Repository (page )</p>
Edge Manager Technician	(assigned by Edge Manager administrator)	<p>This role is assigned by the Edge Manager administrator. Technicians can enroll devices in the cloud using the Predix Machine Technician Console.</p> <p>The Technician role is assigned to the individual responsible for performing the enrollment activity in the Web Console for each device.</p>	<p><a href="#">Enrolling a Predix Machine-enabled Device with the Cloud (page 68)</a></p>

User/ Role	User Name (if applicable)	Description/Functions	Related Documentation
Edge Manager Viewer	(assigned by Edge Manager administrator)	The viewer role has read-only access to most Edge Manager pages but has limited ability to perform actions.	

## Starting Predix Machine

You can start Predix Machine so that it starts the runtime container and any other Predix Machine services that you have installed, such as Yeti.

 **Note:** To run Predix Machine as a service, see [Run Predix Machine as a Service \(page 43\)](#).

1. Navigate to `<Predix Machine runtime container location>/bin`.
2. Use the following command to start Predix Machine:

```
start_predixmachine.sh
```

## Run Predix Machine as a Service

You can install Predix Machine as a service so that you do not have to start the Predix Machine provisioning container manually, and so that it can run with minimal user interaction. The service monitors the container and installs updates when they are pushed to Predix Machine.

 **Note:** You can only run Predix Machine as a service when you generate a provisioning (-PROV) container type.

## Installing the Predix Machine Linux Service

You can install the Predix Machine service in a Linux environment.

You must generate a provisioning (-PROV) container type to use this service.

The service installation setup file is in the `<Predix Machine runtime container location>bin/service_installation` directory.

 **Note:** Do not move the container after installing the service, as the setup uses the absolute path when finding `start` and `stop` scripts. If you move the container, the service cannot start Predix Machine.

1. Navigate to `<Predix Machine runtime container location>bin/service_installation`.
2. Call the `setup.sh` script to start the installer.

If you want to uninstall the service, call the `<Predix Machine runtime container location>bin/service_installation/uninstall.sh` script

## *Starting the Predix Machine Linux Service*

Set the `PREDIXMACHINELOCK` environment variable to point to `/tmp` or `/var/run/predixmachine` so that the "lock" file that is created in the `<PredixMachine_Home>/security` folder is automatically cleared when the machine is restarted or not shut down properly.

You can start the Predix Machine service on Linux systems using `systemd` version 225. `Systemd` is an init system that has been installed as the default init system on many distributions since 2015. You can also run the service on Raspberry Pi using `systemctl` version 215.

1. (Optional) Rescan `systemd` for new or changed unit by using the following command:

```
systemctl --user daemon-reload
```

You specify that this is a user service and not a system service.

2. Start the service by using the following command:

```
systemctl --user start predixmachine
```

You can omit the `.service` filetype at the end of the service configuration filename.

3. (Optional) If you want to re-enable the service at startup, use the following command:

```
systemctl --user enable predixmachine
```

This creates a hyperlink from the service in the `.config/systemd/user` directory to the `.config/systemd/user/default.target.wants/predixmachine.service` directory. Do not create this hyperlink on your own. This is where the system will search to start services at boot.

4. (Optional) If you want to disable the service at startup, use the following command:

```
systemctl --user disable predixmachine
```

This removes the hyperlink and disables the service at startup.

## *Build and Run Sample Applications*

Three sets of sample applications are provided to illustrate how to use Predix features: one for the Predix Machine runtime container, one for cloud applications, and one for edge applications.

To view container or cloud samples, navigate to: <Predix Machine SDK installation location>/samples and extract the files from `sample-apps.zip`, `sample-cloud-apps.zip`.

The `sample-apps.zip` file includes:

<b>Sample</b>	<b>Description</b>
<code>sample-basicmachineadapter</code>	Implements the <code>IMachineAdapter</code> interface and shows how to develop a basic machine adapter.
<code>sample-commandhandler</code>	Shows how to handle various commands issued from Edge Manager
<code>sample-configuration</code>	Shows how to use Meta Mapping for files with configuration files.
<code>sample-container</code>	Provides a POM file to bundle up contents into a working container.
<code>sample-custompolling</code>	Provides an example of how to poll the cloud for updates and tasks on demand.
<code>sample-databus (OSGI)</code>	Shows how to use the the Databus to publish a message and subscribe to a topic from inside the OSGI container. For information about publishing and subscribing from a Docker container, see the Edge SDK samples.
<code>sample-gitrepository</code>	Uses the local repository created from the <code>gitrepository</code> service and shows how to remove files, remove changes, update local repository, and incorporate changes to the remote repository.
<code>sample-healthmachineadapter</code>	Shows how to subscribe to the health adapter and print the data to the log.
<code>sample-hoover</code>	Shows a Processor implementation, which will process the data per the configuration on the spillway.
<code>sample-httpclient</code>	Shows how to use the HTTP Client service to make a rest call.
<code>sample-mqttclient</code>	Shows how to use the MQTT Client to publish a message or subscribe to a topic.
<code>sample-mqttmachineadapter</code>	Shows how to subscribe to MQTT messages from the MQTT machine adapter.
<code>sample-security</code>	Shows how to use the SecurityAdmin service to encrypt and decrypt a string containing sensitive information.

Sample	Description
sample-storeforwardclient	Demonstrates how to use the StoreForward service to pass data through a persistence queue after receiving it from the source, and before sending it to its destination.
sample-subscriptionmachineadapter	Implements the ISubscriptionMachineAdapter interface and shows how to develop the subscription functionality of machine adapter.
sample-websocketclient	Shows how to use the WebSocket client service to connect to a local and external WebSocket endpoint.
sample-websocketriver	Shows how to use the WebSocket River service to connect to an external WebSocket endpoint to send data.
sample-websocketserver	Shows how to use the WebSocket Server service connect to a client WebSocket endpoint.

 **Note:** See [Building Samples from a Command Line \(page 46\)](#) and [Running Samples in Eclipse \(page 48\)](#) or [Building Predix Machine SDK Samples Using Eclipse IDE \(page 49\)](#).

The sample-cloud-apps.zip file includes:

Sample	Description
httptunnel-server	Shows how to start the HTTP Tunnel Server using a script.

 **Note:** Read the associated readme.txt files included with each cloud sample for instructions on how to use the samples.

## *Building Samples from a Command Line*

1. Generate an API key.

a. Sign into <https://artifactory.predix.io>.

 **Note:** Use your predix.io account user name and password to login.

b. In the upper right of the screen, click your user name.

c. Enter your predix.io password.

d. Click **Unlock**.

e. Click the gear icon to generate the key.

The API Key field is populated with the generated API key. The API key values are masked.

f. Click the clipboard icon on the right of the API Key field to copy the key to your clipboard.

2. In the `<user directory>.m2/settings.xml` file:

- a. Configure the proxy settings for Maven based on your own site needs. Ask your network administrator if you have questions about your requirements.
- b. Add a server entry with the following information. (You will use the API key that you copied in Step 1.)

```
<server>
  <id>artifactory.external</id>
  <username>{your predix cloud login}</username>
  <password>{encrypted password - API key}</password>
</server>
```

 **Note:** If you are a GE employee on a GE network, do not include artifactory.external, see Step D.

c. You may also have to set proxy settings for the HTTPS protocol.

```
<proxy>
  <id>optional</id>
  <active>true</active>
  <protocol>http</protocol>
  <username>proxyuser</username>
  <password>proxypass</password>
  <host>proxy.host.com</host>
  <port>80</port>
  <nonProxyHosts>*.host.com|localhost</nonProxyHosts>
</proxy>
```

d. If you are a GE employee, add the following server and repository entries.

```
<server>
  <id>artifactory.repo</id>
  <username>{sso}</username>
  <password>{encrypted password}</password>
</server>
```

```
<repository>
  <id>artifactory.repo</id>
  <name>artifactory.repo or name of your choosing</name>
  <url>https://devcloud.swcoe.ge.com/artifactory/repo</url>
</repository>
<repository>
  <id>artifactory.external</id>
  <name>artifactory.external or name of your choosing.</name>
```

```
<url>https://artifactory.predix.io/artifactory/repo</url>
</repository>
```

3. To build the samples from the command line:
  - a. Navigate to <Predix Machine SDK installation location>/samples and unzip either the sample-apps.zip or sample-cloud-apps.zip files.

4. To build samples:

- a. Navigate to <Predix Machine SDK installation location>/samples/sample-apps/sample/<sample-name>.
- b. Run the following command:

```
mvn clean install
```

 **Note:** Some samples require items to be pushed to Artifactory. (MQTT Client and Cloud Samples, for example.) You must also push a 3rd-party JAR file into your local m2 to build. Refer to the readme.txt files for instructions.

## *Running Samples in Eclipse*

Create a new container using Eclipse. See [Generating a Predix Machine Runtime Container Using Eclipse \(page 35\)](#).

You can run samples using the Predix Machine SDK.

1. Access Eclipse and open your Predix Machine image.
2. In the **Bundles** section, click the **Add** button.  
The **Add bundles** window appears.
3. In the **Bundle type** list, select **Predix Samples Group**.  
A list of each of the Predix samples appears.
4. Select the samples you want to run and click **OK**.
5. Click **Run**.

## *Running Samples from Generated Containers*

You can run samples from a container that was generated using scripts.

Before running any samples from a container that was generated using scripts, make sure the project is built.

 **Note:** Before starting, ensure that the `solutions.ini` file is either copied to or created in the `machine/bin/vms` directory.

1. Copy the JAR file from `<SDK installation location>/samples/sample-apps/sample/<sample-name>` to `<Predix Machine installation location>/machine/bundles/`.
2. Copy any configuration files needed by the sample from the `<Predix Machine SDK installation location>/samples/sample-apps/sample/configuration/machine` directory to the `<Predix Machine runtime container location>/configuration/machine` directory.
3. Modify the `solution.ini` file by adding a `<bundle>` tag for each sample. For example:

```
<bundle>
  <name>com.ge.dspmicro.{sample-name}-{version}.jar</name>
</bundle>
```

You can also copy and paste the existing `solution.ini` file for all samples and modify that file. The `solution.ini` file is located in `<SDK installation location>/samples/sample-apps/sample/machine/bin/vms`.

4. To run the container:
  - a. Navigate to `<Predix Machine runtime container location>/bin`.
  - b. Run the following command: `start_predixmachine.sh`.

## *Building Predix Machine SDK Samples Using Eclipse IDE*

Use the Eclipse IDE in which you installed the Predix Machine SDK to build Java samples.

1. Generate an API key.
  - a. Log in to <https://artifactory.predix.io> using your predix.io user name and password.
  - b. In the upper right of the screen, click your user name.
  - c. Enter your predix.io password.
  - d. Click **Unlock**.
  - e. Click the gear icon to generate the key.  
The API Key field is populated with the generated API key. The API key values are masked.

- f. Click the clipboard icon on the right of the API Key field to copy the key to your clipboard.
2. In the `<user directory>.m2/settings.xml` file:
    - a. Configure the proxy settings for Maven based on your own site needs. Ask your network administrator if you have questions about your requirements.
    - b. Add server entry with this information. (You will use the API key that you copied in Step 1f.)

```

<server>
  <id>artifactory.external</id>
  <username>predix.io login</username>
  <password>{encrypted password - API key}</password>
</server>

```

3. Launch Eclipse and create your own workspace.
4. Import samples by selecting: **File > Import > Maven > Existing Maven Projects**. Click **Next**.
5. Browse and select `<Predix Machine SDK installation location>/samples/sample-cloud-apps/sample` or `<Predix Machine SDK installation location>/samples/sample-apps/sample` as the root directory.
6. Click **Finish** to import all samples into your workspace.
7. Select the sample root directory and then select **Run > Run As > Maven Install**.

## Accessing Predix Machine Web Console

You can access and manage the bundles in your Predix Machine environment using the Predix Machine Web Console.

Start a Predix Machine provisioning (PROV) container.

 **Note:** You can only access the Predix Machine Web Console for Predix Machine DEBUG, PROV, OR TECH container types; you must specify one of these types when you generate the container from the Predix SDK.

1. Open a web browser and navigate to:

```
https://localhost:8443/system/console/
```

 **Note:** Since the Web Console uses a self-signed certificate, the browser will warn that the connection is not private. You can proceed.

A login prompt is displayed.

2. Enter your user name and password. If this is your first time logging into the Predix Machine Web console, use the following defaults:

**Username:** predix

**Password:** predix2machine



**Note:** Your password must:

- Be at least eight characters long and not more than 15 characters long
- Contain at least two uppercase letters
- Contain at least one lowercase letter
- Contain at least two numbers
- Contain at least one special character
- Not contain the user name
- Not contain spaces

The Predix Machine Web Console opens, using an Apache Felix web interface. For more information on using the Apache Web Console, see <http://felix.apache.org/documentation/subprojects/apache-felix-web-console.html>.

3. If you use a proxy server, set HTTP Client proxies.
  - a. In the Predix Machine Web Console, click **OSGi > Configuration**.
  - b. Click the Apache HttpClient OSGi bundle in the Name column. For example, **CM\_GENERATED\_PID.0**.
  - c. Select the **proxy.enabled.name** check box.
  - d. In the **proxy.host.name** box, enter the proxy host.
  - e. In the **proxy.port.name** box, enter the proxy port number.
  - f. Complete the **proxy.password.name** and **proxy.exceptions.name** box, and click **Save**.

## *Configuring Predix Machine Web Console Plug-ins*

As the administrator, you can use the `org.apache.felix.webconsole.internal.servlet.OsgiManager.cfg` file to configure plug-ins that appear as menu items in the **Predix Machine Web Console > > OSGi** menu to limit the operations that the technician can perform.

**Important:** Do not use the **Apache Felix OSGi Management Console** page in the Predix Machine Web Console (**Predix Machine Web Console > > OSGi > Configuration > Apache Felix OSGi Management Console**) to configure plug-in/menu options. If you make changes on that page, the menu items may not appear correctly.

1. In a Predix Machine Technician container, navigate to `<Predix Machine runtime container location>/configuration/machine`.

- Open the `org.apache.felix.webconsole.internal.servlet.OsgiManager.cfg` file.
- Use any of the following values to configure the `plugins` property. To configure more than one plug-in, use comma-separated values.

Plug-in	Value
Configuration Manager	<code>org.apache.felix.webconsole.internal.configuration.ConfigManager</code>
Log Services	<code>org.apache.felix.webconsole.internal.compendium.LogServlet</code>
Bundles	<code>org.apache.felix.webconsole.internal.core.BundlesServlet</code>
Services	<code>org.apache.felix.webconsole.internal.core.ServicesServlet</code>
Licenses	<code>org.apache.felix.webconsole.internal.misc.LicenseServlet</code>
VM Stat	<code>org.apache.felix.webconsole.internal.system.VMStatPlugin</code>

The following example shows the `plugin` value populated with the Configuration Manager and Log Services plug-ins.

```
plugins=org.apache.felix.webconsole.internal.configuration.ConfigManager,org.apache.felix.webconsole.internal.compendium.LogServlet
```

## Secure Predix Machine

### *Securing Predix Machine*

Because Predix Machine supports a variety of deployments, providing a comprehensive guide to securing it is impractical. At a minimum, solutions should perform these recommended steps.

- [Protect the <Predix Machine runtime container location>/security folder. \(page 52\)](#)
- [Change KeyStore and TrustStore passwords \(page 53\).](#)
- [Issue certificates \(page 54\).](#)
- [Secure the Technician Console \(page 54\)](#), and upon first login, change the password.

### *Protecting the Security Folder*

The Predix Machine `security` folder must be protected so that only a single user with specific privileges can read the contents of the folder.

The `<Predix Machine runtime container location>/security` folder of Predix Machine contains sensitive information such as passwords and KeyStores with private keys. At a minimum, this folder needs to be completely protected so that only a single user with minimal privileges can read the contents of the folder. This ensures that no other users in the system can read the contents of the security folder. Make sure that the user that has access to the `security` folder is restricted from deeper access to the system.

The following is an example of how you might protect the `Security` folder.

1. Create a user called `predixuser` as part of the General Users group in your respective operating system. This user should not be an administrator or root user.
2. Change all controls (read, write, execute) of the `<Predix Machine runtime container location>/security` folder to be accessible only to the new user.
3. Run Predix Machine as a user with limited privileges.

 **Important:** Avoid running Predix Machine as root/administrator. If you must run as root/administrator, ensure that there is limited access to the root/administrator account.

[Changing KeyStore and TrustStore Passwords \(page 53\)](#)

## *Changing KeyStore and TrustStore Passwords*

To secure Predix Machine, you should change KeyStore and TrustStore passwords.

- Many of the KeyStores used by the Security Admin service have generated passwords and should be considered secure. KeyStores generated for client and server TLS, digital signatures, and secret keys have generated passwords. The passwords are available in the Security Admin configuration file at `<Predix Machine runtime container location>/security/com.ge.dspmicro.securityadmin.cfg`.

Some passwords are not generated and should be changed.

- TrustStore passwords are not generated and should be changed immediately.
1. Change KeyStores for OPC-UA client and OPC-UA server services immediately because the passwords in their respective configuration files are difficult to recover.

 **Note:** See [Configuring the OPC-UA Adapter \(page 168\)](#) and [Configuring the OPC-UA Server \(page 175\)](#).

2. Change TrustStore passwords in the `<installation location>/security` folder.

 **Note:** See [Configuring the Client KeyStore/TrustStore \(page 109\)](#) and [Configuring the Server KeyStore/TrustStore \(page 110\)](#).

[Issuing Certificates \(page 54\)](#)

## Issuing Certificates

Predix Machine generates several self-signed certificates for its various keystores. These can be used to establish a TLS connection, but browsers do not trust these certificates because they are not signed by a trusted root certificate authority (CA).

If a properly signed certificate is required, you must obtain a certificate issued by a CA.

1. Obtain a certificate issued by a CA.
2. Import the certificate into the server SSLContext KeyStore at `<installation location>/security/tls_server_keystore.jks` by [configuring the server KeyStore/TrustStore \(page 110\)](#).

At first login, change the password to the Predix Machine Technician Console.

## Securing the Technician Console

The Technician Console provides administrative control over Predix Machine. You should secure the console when setting up a new Predix Machine instance.

1. Use a CA signed certificate.
  - a. Obtain a signed certificate.
  - b. Import the certificate into the `<Predix Machine runtime container location>/security/tls_server_keystore.jks` file. See [Importing a New Private Key and CA Signed Certificate \(page 108\)](#)
  - c. Configure the server SSLContext keystore. The Predix Machine Technician Console uses the certificate provided in the Security Admin keystore to establish TLS.
2. Change the Predix Machine Technician Console password.
  - a. Select **Predix > User Administration**.
  - b. Select the default user.
  - c. Enter a new password.
3. After set up of Predix Machine is complete, disable the Technician Console to prevent access. You can disable and enable using the Technician Console: Disable command in Edge Manager. See [Executing Commands \(page \)](#).

If access is required in the future, you must re-enable it.

## *Predix Machine VPN Service*

Predix Machine provides a service for managing a VPN client to provide a secure connection to desired server destinations.

You must have a prepared environment with a server and client that are ready to communicate before you can use the Predix Machine VPN service.

Predix Machine provides its certificate and key to be used in the HTTP communication between the VPN client and server by using the VPN's management facility. This allows Predix Machine to retain its private key and provide an API to sign requests as needed for the VPN communication.

 **Note:** As the request signing is performed by Predix Machine, you must engage Predix Connectivity's resources to design a workable configuration scheme that takes into account the certificate authorities and destination server setup.

### **Predix Connectivity Use Case**

An example use case is Predix Machine runs on a field device that is enrolled in Predix Edge Manager via certificate enrollment. VPN servers are set up to recognize certificates that are given to Predix Machine by the certificate authorities that Edge Manager uses for device enrollment. They also set up the VPN servers to trust the same certificate authority. Thus, when VPN clients request for information to be signed by Predix Machine, the private key and certificate used are already trusted by the server.

Other use cases can vary depending on needs.

### **Functionality**

1. Configure VPN directory settings in the `com.ge.dspmicro.predix.connectivity.openvpn.config` file.
2. Enroll devices.
3. The Predix Connectivity bundle provides a public certificate for the VPN client to present to the server after device enrollment.
4. The certificate for the VPN client must be supplied before the VPN client is started, otherwise, it terminates with an error. Once the certificate is available the VPN client can be started.
5. It then opens a TCP communication socket to go into wait mode until a management client is available to manage it.
6. If Predix Machine is running and the connectivity VPN management configuration switch is enabled, it acts as the management client, answering the VPN client's signing requests to enable a secure connection.

Evidence of a successful VPN connection is seen in the existence of a TUN/TAP device in the network interface.

## Limitations

- The current VPN management is only available for the Linux environment.
- The configuration of VPN is platform dependent, and also may be OS version dependent.
- Versions of VPN prior to 2.3.12 have bugs that render the management interface unusable in the manner described.

## Configuring VPN

Configure the settings for the VPN directory.

1. Navigate to `<Predix Machine runtime container location>/configuration/machine`.
2. Open the `com.ge.dspmicro.predix.connectivity.openvpn.config` file and set the values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.predix.connectivity.openvpn.agent.enabled</code>	This setting enables you to manage VPN connections on devices through Edge Manager. Set to <code>true</code> to enable VPN management on Edge Manager. When set to <code>False</code> , disables VPN management.	Boolean	<code>false</code>	No
<code>com.ge.dspmicro.predix.connectivity.openvpn.dir</code>	OpenVPN client configuration directory. This is where all information being provided to VPN is kept.	String	<code>/etc/openvpn</code>	No

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.predix.connectivity.openvpn.client.hostname</code>	OpenVPN client management interface hostname. For security reasons, this should always be localhost.	String	localhost	No
<code>com.ge.dspmicro.predix.connectivity.openvpn.client.port</code>	VPN client management interface port	Integer	7509	Yes
<code>com.ge.dspmicro.predix.connectivity.openvpn.remote.hostname</code>	The remote host name. Currently not used, as already configured in VPN client	String		No
<code>com.ge.dspmicro.predix.connectivity.openvpn.remote.port</code>	The remote host name. Currently not used, as already configured in VPN client	Integer	1194	No
<code>com.ge.dspmicro.predix.connectivity.openvpn.log.verbosity</code>	Set the log verbosity for the VPN client.	Integer	1	No

Property	Description	Type	Default Value	Required
com.ge.dspmicro.predix.connectivity.openvpn.datepattern	<p>A Date Pattern string for specifying a logging schedule:</p> <ul style="list-style-type: none"> <li>• MONTHLY – for rolling at midnight on first day of the month</li> <li>• DAILY – for rolling daily at midnight</li> <li>• WEEKLY – for rolling at midnight on first day of week</li> <li>• DAY_NIGHT – for rolling daily at midnight and midday</li> <li>• HOURLY – for rolling every hour on the hour</li> <li>• MINUTE – for rolling every minute on the minute</li> </ul>	String	MONTHLY	No
com.ge.dspmicro.predix.connectivity.openvpn.backuplogs	Number of backup logs to retain.	Integer	10	No
com.ge.dspmicro.predix.connectivity.openvpn.logdirectory	The absolute path for vpn logs directory. VPN must be independently configured to log to this location, to a file named vpn.log. The file name must not be added to the value for this setting.	String	"/etc/openvpn"	No

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.predix.connectivity.openvpn.coll.threshold</code>	The size threshold which the log file should exceed before it will be rolled, in kilobytes. A value of 1 means 1kb.	Long	1024	No
<code>com.ge.dspmicro.predix.connectivity.openvpn.enableondemandevents</code>	<p>Enable to allow the cloud to send on-demand events to devices.</p> <p>If enabled, Event listener listens for events on the VPN Network over port 7510.</p>	Boolean	False	No

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

## Get VPN Log Example

The `ILogHandler` service interface is designed to work with the Predix Machine and Predix Edge Manager log discovery and retrieval system.

The `ILogHandler` service interface makes VPN logs available to the `GetLog` and `GetAvailableLogs` commands that are already available in Predix Machine.

- `GetLog` – uploads a user-specified log file to Predix Edge Manager.
- `GetAvailableLogs` – retrieves the file names of logs you give to the `GetLog` command.

Example:

```
public String[] getLogFileNames();
```

```
/**
 * @param logFileName - the filename for which to get content for. It
 * should be a name that was given by getLogFileNames().
 * @return - an InputStream from which log content may be read.
 *
 * It is up to the implementation to handle file permission issues for
 * log files that are stored in restricted portions of
 * the device storage, as well as the reading of files that are perhaps
 * being written to as they are read.
 */
public InputStream getLogContent(String logFileName);
}
```

## Device Enrollment in the Cloud

### *About Predix Cloud Device Enrollment*

For cloud enrollment, devices must be added to Predix Edge Manager by an administrator or operator before enrolling the device with the technician console. Enroll devices with Predix Edge Technician Console for devices running Predix Edge Agent, or Predix Machine Technician Console for devices running Predix Machine.

When the device is initially added to Predix Edge Manager, it has no identity associated with the Predix cloud until an identity is created on the cloud through certificate enrollment and associated with the device using Predix cloud authentication.

Certificate-based device authentication and enrollment allows a device to enroll itself with Predix Edge Manager at startup and obtain a certificate signed by a GE root authority so that no device-specific credentials are required. Once a device is configured with the Edge Manager URL, device ID, and shared secret, it can communicate with the cloud environment at startup and obtain its own certificate and credentials.

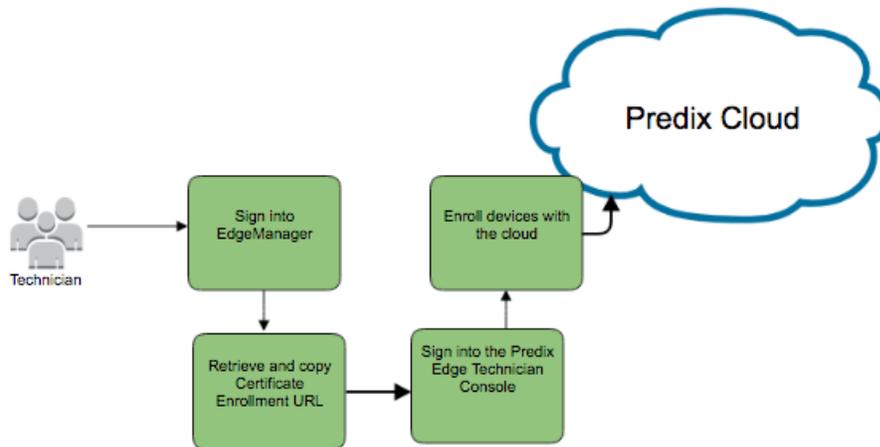
### **Administrator Tasks**

1. The administrator creates the technician user with the Technician role in Predix Edge Manager, and provides the technician with Predix Edge Manager login credentials.
2. The administrator or operator adds devices to Predix Edge Manager and enters a shared secret for the device.

### **Technician Tasks**

Task	Description
1. Login to Predix Edge Manager and change password.	The administrator provides initial sign-in credentials and the URL to access Predix Edge Manager to the technician. When the technician logs in for the first time, they are prompted to change their password.
2. Go to <b>Settings</b> .	The technician is directed to the <b>Settings &gt; Enrollment</b> page and makes note of the appropriate certificate enrollment URL.
3. Sign into the local technician console.	Sign into the technician console.  For Predix Machine, see <a href="#">Accessing Predix Machine Web Console (page 50)</a> . For Predix Edge Agent, see <a href="#">Using Predix Edge Technician Console to Enroll Devices with Predix Cloud (page 64)</a> .
4. Finish enrollment process.	The technician finishes enrolling the device with either Predix Edge Technician Console or the Predix Machine Technician Console. This creates an identity for the device in the cloud.

Figure: Technician Workflow for Predix Edge Technician Console



**Related concepts**

[Predix Cloud Identity Management Service \(page 73\)](#)

**Related tasks**

[Adding a Device to Predix Edge Manager \(page 62\)](#)

[Using Predix Edge Technician Console to Enroll Devices with Predix Cloud \(page 64\)](#)

## Adding a Device to Predix Edge Manager

When you add a device to Predix Edge Manager, information that is specific to the device is added so that when you enroll the device with Predix Machine or Predix Edge Agent, the device can be verified through the security certificate.

Before a device that has Predix Machine or Predix Edge Agent installed can be enrolled and brought online, you must add the device to Predix Edge Manager. This procedure is for adding a single device to Predix Edge Manager. To add multiple devices, see [Importing a Device List \(page 63\)](#).

1. Sign into Predix Edge Manager.

2. In the left navigation pane, select  **Device Manager > Devices**.

3. In the Device Manager page, select **Action > Add**.

4. In the **Add a Device** dialog box, enter the information for the device:

- **Device Name** – the name of the device should be unique and descriptive, and can consist of upper and lower case characters and numbers.
- **Device ID** – used to identify the device with Predix Edge or Machine. The device ID must be unique in a Predix Edge Manager tenant. While the Device ID is typically a serial number, another option is using the MAC address of the WAN interface, which is auto-populated on the Predix Cloud Enrollment page in the local technician console.

 **Note:** The Device ID can consist of lower-case characters and numbers, however, any upper-case characters entered during device creation will be converted to lower-case.

 **Note:** The device ID must follow these conventions:

- Must be a minimum of 3 characters.
- Must not exceed 63 characters.
- Must start with an alphanumeric character (0-9 or a-z).
- The remaining characters can be any combination of alphanumeric, underscore (\_), or hyphen (-).
- Do not use colons (:).
- The Device ID is case-insensitive, but is always stored as lower-case. If you enter upper-case characters in Predix Edge Manager, they are converted to lower-case.

 **Note:** Write down or copy your Device ID for use when enrolling the device with Predix Machine or Predix Edge later.

- (Optional) **Group** – Select the target group for the device.
- (Optional) **Technician** – Select a technician to whom to assign the device.
- **Device Model** – Select the device model from the drop-down list.

- (Optional) **Manufacturer Installed BOM** – A manufacturer BOM lists packages installed before the device is shipped to the user (the packages are not installed through Edge Manager).
  - a. Click **Choose BOM**.
  - b. Select a BOM from the list, and click **Confirm**.

 **Note:** Once a manufacturer BOM is installed, it cannot be modified. Any BOMs deployed at a future date are compared against the initial manufacturer installed BOM, and any packages that are already installed as part of the initial manufacturer BOM are skipped.

- (Optional) **Description** – Add a description for the device.
- **Shared Secret** – Enter the **Shared Secret**. The shared secret provides an initial form of authentication for a device that otherwise does not have an existing identity when you enroll it with Predix Edge. Certificate-based device authentication and enrollment allows a device to enroll itself to Predix Edge Manager at startup and obtain a certificate signed by a root authority.
- **Confirm Secret** – Re-enter the shared secret.
- (Optional) Click **Next** to assign a service to the device.
- Click **Finish** to add the device.

5. If you clicked **Next** in the previous step, in the **Assign Service** dialog box, select the service, or services, to assign to the device.
  - (Optional) Click **Next** to add location details for the device.
  - Click **Finish** to add the device.

6. (Optional) If you clicked **Next** in the previous step, in the **Location** dialog box, enter location details for the device.

 **Note:** The **Elevation** value must be in meters.

- Click **Next** to add custom attributes for the device.
- Click **Finish** to add the device.

7. (Optional) If you clicked **Next** in the previous step, in the **Custom Attributes** dialog box, enter custom attributes as key/value pairs, then click **Finish**.  
Key/value custom attributes can be used to add more details about a device, for example, `Region:West`.  
Click **+** to add more attributes, and **X** to delete attributes.

8. Click **Finish**.

You receive a confirmation that the device has been successfully added. The device list automatically refreshes and displays the device you added. This may take a moment.

Once you have added the devices to Predix Edge Manager and assigned the technician, the technician can enroll them with Predix Machine or Predix Edge Agent. The technician needs to know the following information in order to enroll the devices:

- Device ID
- Certificate enrollment URL (found on the Settings page)
- Shared secret

### Related concepts

About Predix Edge Manager Groups (*page* )  
Edge Manager Predix Cloud Service Configuration (*page* )

### Related tasks

Viewing Devices in a Specific Group (*page* )  
[Enrolling a Predix Machine-enabled Device with the Cloud \(\*page 68\*\)](#)  
[Using Predix Edge Technician Console to Enroll Devices with Predix Cloud \(\*page 64\*\)](#)  
Viewing the Device Summary (*page* )  
Importing a Device List (*page* )

## Enroll a Device using Predix Edge Technician Console

### *Using Predix Edge Technician Console to Enroll Devices with Predix Cloud*

You must install Predix Edge Technician Console (*page* ).

For devices running Predix Edge, with connectivity to Predix cloud, you can use the Predix Edge Technician Console to configure the device with the Predix Edge Manager certificate enrollment URL, device ID, and shared secret, so it can communicate with the cloud environment at startup and obtain its own certificate and credentials.

1. Sign into Predix Edge Technician Console.
2. In the **Device Status** page, click **Enroll**.
3. In the **Enroll Device** dialog box, enter the following information:
  - **Device ID** – Identifies the device with Predix Edge OS. The device ID you enter must match the device ID assigned when the device was added to Edge Manager by the administrator.
  - **Shared Secret** – Enter the shared secret that was entered with the device was added to Predix Edge Manager.
  - **Certificate Enrollment URL** – URL of the Predix Edge Manager tenant. You can find the correct certificate enrollment URL in the Predix Edge Manager **Settings** page.

#### 4. Click **Enroll**.

A green banner displays at the top of the Device Status screen confirming enrollment was successful and the device status displays "enrolled."

In Edge Manager, the device status displays "online" (this may take a moment).

## Enroll a Device using the Predix Machine Technician Console

### *Predix Machine Technician Console-based Device Enrollment*

A technician can use the Predix Machine Technician Console to enroll devices after an administrator or operator has added the device to Predix Edge Manager.

The flow for enrolling devices using the Predix Machine Technician Console is as follows:

#### **Administrator Tasks**

1. The administrator creates the technician user with the Technician role in Predix Edge Manager, and provides the technician with login credentials.
2. The administrator or operator adds devices to Predix Edge Manager and enters a shared secret for the device.

#### **Technician Tasks**

<b>Task</b>	<b>Description</b>
1. Login to Predix Edge Manager and change password.	The administrator provides initial sign-in credentials and the URL to access Predix Edge Manager to the technician. When the technician logs in for the first time, they are prompted to change their password.
2. Go to Settings.	The technician is directed to the <b>Settings &gt; Enrollment</b> page and makes note of the appropriate certificate enrollment URL.
3. Sign into the Machine Technician Console.	Sign into the Predix Machine Web Console and go to <b>Technician Console &gt; Predix Cloud Enrollment</b> .  See <a href="#">Accessing Predix Machine Web Console (page 50)</a> .

Task	Description
4. Finish enrollment process.	The technician finishes enrolling the device with Predix Machine. This creates an identity for the device in the cloud.  See <a href="#">Enrolling a Predix Machine-enabled Device with the Cloud (page 68)</a> .

Figure: Technician Device Enrollment Flow

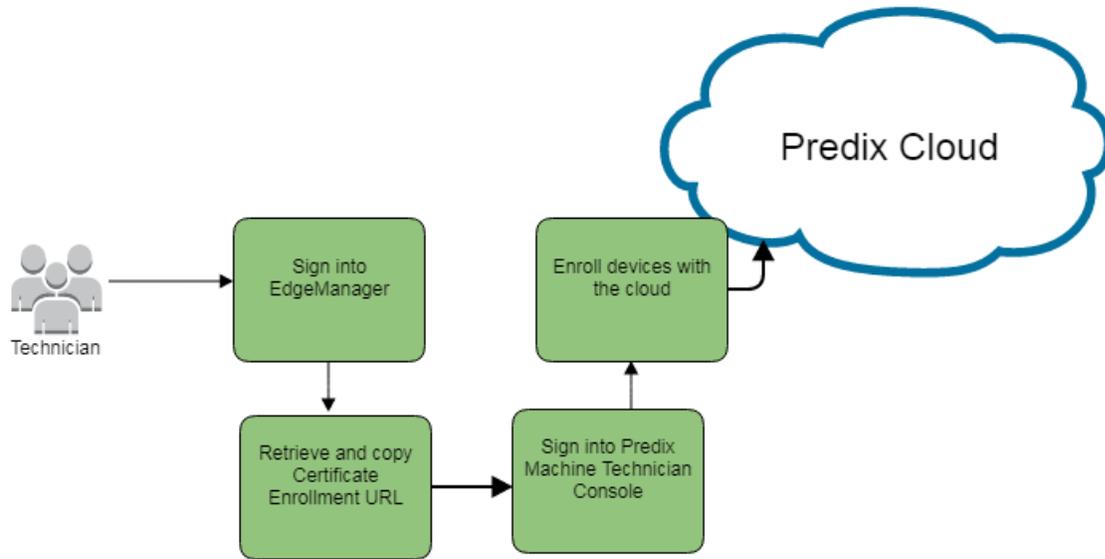
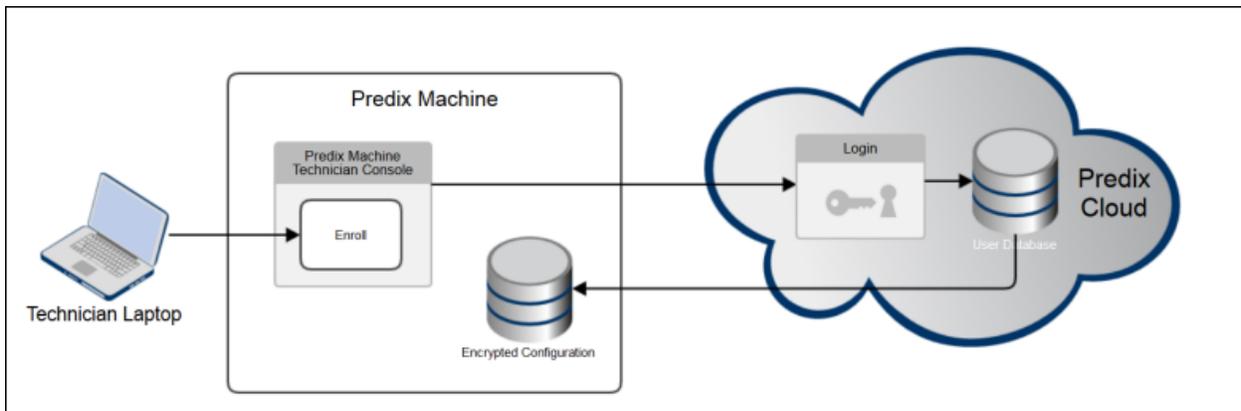


Figure: Predix Machine Device Enrollment Process



## Accessing Predix Machine Web Console

You can access and manage the bundles in your Predix Machine environment using the Predix Machine Web Console.

Start a Predix Machine provisioning (PROV) container.

 **Note:** You can only access the Predix Machine Web Console for Predix Machine DEBUG, PROV, OR TECH container types; you must specify one of these types when you generate the container from the Predix SDK.

1. Open a web browser and navigate to:

```
https://localhost:8443/system/console/
```

 **Note:** Since the Web Console uses a self-signed certificate, the browser will warn that the connection is not private. You can proceed.

A login prompt is displayed.

2. Enter your user name and password. If this is your first time logging into the Predix Machine Web console, use the following defaults:

**Username:** `predix`

**Password:** `predix2machine`

 **Note:** Your password must:

- Be at least eight characters long and not more than 15 characters long
- Contain at least two uppercase letters
- Contain at least one lowercase letter
- Contain at least two numbers
- Contain at least one special character
- Not contain the user name
- Not contain spaces

The Predix Machine Web Console opens, using an Apache Felix web interface. For more information on using the Apache Web Console, see <http://felix.apache.org/documentation/subprojects/apache-felix-web-console.html>.

3. If you use a proxy server, set HTTP Client proxies.
  - a. In the Predix Machine Web Console, click **OSGi > Configuration**.
  - b. Click the Apache HttpClient OSGI bundle in the Name column. For example, **CM\_GENERATED\_PID.0**.
  - c. Select the **proxy.enabled.name** check box.
  - d. In the **proxy.host.name** box, enter the proxy host.
  - e. In the **proxy.port.name** box, enter the proxy port number.
  - f. Complete the **proxy.password.name** and **proxy.exceptions.name** box, and click **Save**.

## *Enrolling a Predix Machine-enabled Device with the Cloud*

Technicians use the Predix Machine Technician Console to enroll a Predix Machine-enabled device in the Predix cloud.

Administrators or operators must add a Predix Machine-enabled device to Predix Predix Edge Manager before the technician enrolls the device with Predix Machine. See [Adding a Device to Predix Edge Manager \(page 67\)](#).

1. Sign into the Predix Machine Web Console.
2. Click **Technician Console > Predix Cloud Enrollment**.
3. Enter the following information:
  - **Device ID** – Device ID from Predix Edge Manager device summary. This matches the device ID you entered when you added the device to Predix Edge Manager.
    -  **Note:** The device ID must follow these conventions:
      - Must be a minimum of 3 characters.
      - Must not exceed 63 characters.
      - Must start with an alphanumeric character (0-9 or a-z).
      - The remaining characters can be any combination of alphanumeric, underscore (\_), or hyphen (-).
      - Do not use colons (:).
      - The Device ID is case-insensitive, but is always stored as lower-case. If you enter upper-case characters in Predix Edge Manager, they are converted to lower-case.
  - **Certificate Enrollment URL** – URL of your Predix Edge Manager tenant.
  - **Shared Secret** – Provides an initial form of authentication for a device that otherwise does not have an existing identity.
4. Click **Enroll Device**.

The **Enroll Device** button is disabled while the device generates a certificate and sends the corresponding Certificate Signing Request (CSR) to the Predix cloud for signing. This process can take up to 30 seconds to complete before an enrollment success message is displayed.

When certificate enrollment is complete, Predix Machine restarts itself before it appears in Predix Edge Manager with an **Online** status.

### **Related tasks**

[Viewing the Device Summary \(page 67\)](#)

# Upgrade Predix Machine

## *Upgrading Over the Air Using Edge Manager*

You can use Predix Edge Manager to upgrade Predix Machine over the air.

- You must have access to Predix Edge Manager and the Predix Machine Technician Console using a Predix Machine provisioning (-PROV) container.

1. Unzip the `PredixMachineSDK-17.2.x.zip`.  
You see a configuration and machine folder.
2. Zip up configuration folder as `configuration.zip` and machine as `machine.zip`.  
`zip -r -X <archive_name>.zip <directory_name>`
3. Sign in to Predix Edge Manager.
4. Upload `configuration.zip` as type "configuration," and `machine.zip` as type "system" or "application" to Predix Edge Manager.
5. Deploy the configuration package first to the device.
6. Deploy the software/application package to the device.

## *Upgrading a Predix Machine Container*

Schedule Predix Machine container upgrades, or use the upgrade script.

- You must have access to Edge Manager and the Predix Machine Technician Console using a Predix Machine provisioning (-PROV) container.

If your `<Predix Machine runtime container location>` contains a `/yeti` folder, you can schedule upgrades to the container in Edge Manager. Otherwise, you can upgrade a Predix Machine runtime container by using `<Predix Machine runtime container location>/bin/start_predixmachine.sh` script to start a container. This starts a watchdog to keep the container running and acts as an installer to upgrade applications. For example, if you want to upgrade an existing container, use the following procedure.

Starting with Predix Machine 17.2.0, a program is available in `<Predix Machine SDK download location>/predixmachinesdk-17.2.x/utilities/CompareContainer` that will compare two machine folders and build a download package, with

updated JAR files, for upgrading from one to the other, or for adding components to an existing container.

1. Create a package that includes the configuration or applications to upgrade.
  - a. Create a container that includes the solution you want to install on the device.
  - b. Create an `install` directory that includes an `install.sh` (Linux or Mac).  
For example, `<Application or configuration>/install/install.sh`.

 **Note:** If you do not include an install script, the package upload will fail.

2. Inside the Predix Machine directory, which includes the configuration files or applications you want to update, create a ZIP package. For example, to update configuration files, zip the `<Predix Machine installation location>/configuration` folder.
  - In OS X, right-click on the folder and click **Compress <directory name>**.
  - In Windows, right-click on the folder and click **Send to > Compressed (zipped) folder**.
  - In Ubuntu, compress and give the package a `.zip` extension.

 **Warning:** Always test your package installation on a local device before deploying it remotely. If the installation fails on a remote device, it may become unreachable and require field service.

3. In Edge Manager, upload software to the repository. See [Uploading Software and Configuration Packages to the Predix Edge Manager Repository \(page 42\)](#).
4. Start Predix Machine using the `<Predix Machine runtime container location>/bin/start_predixmachine.sh` script. See [Start Predix Machine \(page 43\)](#).

Predix Machine console will be displayed in the terminal while running this script, but will not be accessible for server commands

5. If the device is not previously enrolled, enroll and deploy the device. See [Adding a Device to Predix Edge Manager \(page 44\)](#).
6. Deploy device configuration. See [Viewing and Deploying Device Configurations to a Single Device \(page 45\)](#).
7. Deploy the application that corresponds to the configurations you deployed. See [Viewing and Deploying Device Software to a Single Device \(page 46\)](#).

Packages can be deployed at any time to the device. If the device is unreachable, the packages will be installed when the device reconnects.

8. To shut down the container, use the `<Predix Machine runtime container location>/bin/stop_predixmachine.sh` script.

The console will display checks as the Predix Machine is shutdown:

```
Yeti is shutting down.
Shutdown signal sent successfully..
Watchdog has shutdown.
Shutdown complete.
Yeti has shutdown.
```

The `stop_predixmachine` script will check once a second for 3 minutes for the stop notification before throwing an error to the terminal. See the `$PREDIXHOME/logs/machine/start_predixmachine.log` or `$PREDIXHOME/logs/machine/machine.log` for more information on the failure.

## *Upgrade Return Codes*

Find out what Predix Machine upgrade return codes mean.

A return code of 0 indicates that your Predix Machine upgrade was successful. Other return codes indicate a problem during the upgrade. Return codes between 20-49 indicate a problem with the package, or with Yeti. Codes greater than 50 indicate a problem with the install script or procedure.

Common return codes are listed in the following table.

<b>Error Code</b>	<b>Description</b>
21	No signature file found for the associated ZIP file. Package origin could not be verified.
22	Package origin was not verified to be from the Predix cloud. Installation failed.
23	Unable to extract archive using JAR or unzip utilities. Cannot perform upgrade.
24	Incorrect ZIP format. Applications should be a single folder with the <code>packagename/install/install.sh</code> structure.
51	An error occurred while running the install script.
52	Predix Machine did not reconnect to the cloud after the installation of the package.
53	An error occurred while running the installation script. The installation script did not produce a JSON result to verify its completion.
54	Previous backup directory could not be removed.
55	Previous directory could not be copied to <code>backup directory.old</code> .
56	Update could not be moved to the container data directory.

Error Code	Description
100	The installation is in progress. Yeti does not check for the JSON file created by the installation script and continues with the other installations. Use this only if another process creates the status JSON file, such as after a system restart.

## *Working with Slow Machines*

When a config is pushed to a slow machine the yeti process often times out before the new config can be installed and rolls back the installation. The `yeti.cfg` file allows you to configure the timeout value allowing configs to install on slow machines. The default is 600 seconds in 10 second increments.

1. Navigate to `machines/yeti`.
2. Open `yeti.cfg` file.
3. Set the `rollbackWaitDuration` value to a positive integer value. The default is 60.  
`rollbackWaitDuration=60`.

The following is an example of the `yeti.cfg` file.

```
# [Optional] The number of 10 second wait cycles after installation for
# Yeti to wait for the
# container to reconnect to the cloud. After this time has expired, Yeti
# rollback the previous installation (only compatible with default
# install.sh scripts)
# Minimum suggested value is 60. This should be increased on slower
# devices that may take
# more time to start the process.
# e.g. rollbackWaitDuration=60 will wait 60*10=600 seconds before rolling
# back.
rollbackWaitDuration=60
```

## Connect Data to the Cloud

### *Cloud Services Overview*

#### *About Connecting Data to the Cloud*

Predix Machine includes services that allow you to connect your Predix Machine-enabled devices to the cloud and send data from those devices to the cloud.

These services include:

**Predix Cloud Identity Management**

Enables Predix Machine device enrollment with the cloud using certificate enrollment or as an OAuth2 client, and provides token management after device enrollment.

**WebSocket River**

Provides connectivity between Predix Machine-enabled device and the Predix Time Series in the cloud.

**Event Hub River**

Enables data transfer from Predix Machine-enabled devices to the Predix cloud using HTTPS.

**Command Framework**

Provides the ability to send commands to applications that are running on a device.

**HTTP Data Service**

Receives data from the Event Hub River service and persists it to a relational database in the Predix cloud.

## *Predix Cloud Identity Management Service*

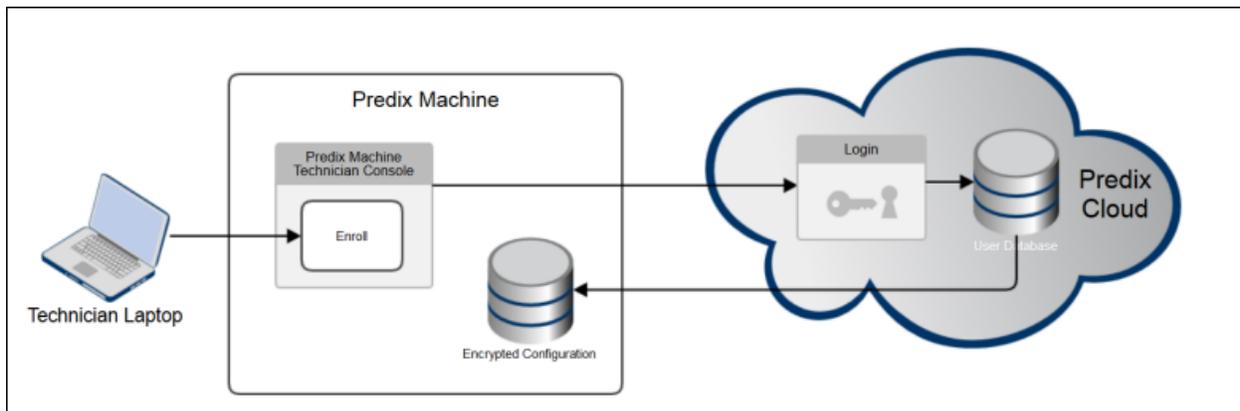
### *Predix Cloud Identity Management Service*

The Predix Cloud Identity Management Service allows you to enroll a Predix Machine device using certificate enrollment, which transparently uses OAuth2 access tokens to send data to protected endpoints.

**Device Enrollment**

A Predix Machine-enabled device initially has no identity associated with the Predix cloud. Similarly, the Predix cloud has no knowledge of the device. The Predix cloud Identity Management service allows you to create an identity on the cloud and associate it with the device using Predix cloud credentials. No device-specific credentials are needed. The sequence of steps for this flow is shown below:

Figure: Device Enrollment Process



## Enrollment Prerequisites

Some solutions that consume Predix Machine may require the following prerequisites because many manufacturing processes have various methods to obtain these values and are non standard.

- Predix Machine provides the `ICertificateEnrollParameters` interface, a service interface that provides required values for automatic enrollment.
- The solution must implement an OSGi service that implements the `ICertificateEnrollParameters` interface. A default implementation is provided to pull these values from environment variables or an INI file.

```
public interface ICertificateEnrollParameters
{
    /**
     * @return - String - the serial number of the device
     */
    public String getSerialNumber();
    /**
     * @return - String - the URL to for the edge manager server
     */
    public String getEdgeManagerURL();
    /**
     * @return shared secret required for enrollment
     */
    public String getSharedSecret();
}
```

- Before provisioning the device to the cloud, the following settings must be provided or generated:
  - The solution must provide an external application to call the Edge Manager API for adding a device.
  - An operator must generate a `client_id` and `client_secret` with the correct scopes for this application so that the application can access the Edge Manager APIs.
  - Along with other attribute values, the application must provide the `serial number` and `shared secret` implemented by the interface in the call to Edge Manager.

 **Note:** To view all attribute values for the `/addDevice` API, see the <https://www.predix.io/api>.

- The application must have a way to obtain this shared secret.

This can either pull the shared secret from the device or generate the secret and push it to the device before enrollment. Because these processes can vary by solution, the way to perform this is left open.

## Consumer Configuration

A Maven dependency and an OSGi import are required to consume the certificate-based device authentication and enrollment service if your solution uses its own implementation of the `ICertificateEnrollParameters` interface.

 **Note:** If you are using the default implementation of the `ICertificateEnrollParameters` interface, the following dependency and import are not required.

- The following Maven dependency is required to consume the service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>device-api</artifactId>
  <version>{Predix Machine version}</version>
</dependency>
```

- The following OSGi import is required to consume the bundle:

```
Import-Package: com.ge.dspmicro.device.api;version="[1.0,2]"
```

## *Obtaining the Predix Cloud Identity Management Service*

This example shows how to inject the Predix Cloud Identity Management service.

To obtain the Predix Cloud Identity Management service, you must add a reference to it so that OSGI will call your service with the service reference ("inject" the service).

Inject the `cloudIdentityService` using declarative services.

This example shows how to inject the service:

```
import com.ge.dspmicro.device.api.clientidentity;
...
@Reference
public void setService(IClientIdentity cloudIdentityService)
```

```
{
  //set service here
}
```

## Using the Predix Cloud Identity Management Service APIs

The Cloud Identity Management Service provides an `IClientIdentity` API to establish an identity so that Predix Machine can obtain tokens from the UAA Service.

Use the `IClientIdentity` REST endpoint to send a request to the enrollment service and receive authorization credentials.

Method	URL	Request Data	Response
POST	/enroll/enrollCert	<ul style="list-style-type: none"> <li>• HTTP Basic Auth</li> <li>• Headers: Basic base64(username:password)</li> <li>• Body: JSON payload containing the deviceId, edgeManagerUrl, and sharedSecret</li> </ul>	<ul style="list-style-type: none"> <li>• 200 Enrollment successful</li> <li>• 400 Bad request</li> <li>• 401 Unauthorized</li> </ul>

The following shows an example request:

```
URL: /enroll/enrollCert
Method: POST
Headers:
  Authorization: Basic base64(username:password)
Payload:
{
  "deviceId": "mydeviceId",
  "edgeManagerUrl": "<EdgeManager_URL>",
  "sharedSecret": "<Shared_Secret>"
}
```

### Related information

- (page )
- (page )

## Configuring the Predix Cloud Identity Management Service

You can configure your implementation of the Predix cloud Identity Management Service to use certificate authorization for the device.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is property=`<type>["<value1>","<value2>"]`. For example, array of integer property=`I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to, and open, the file `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.predixcloud.identity.config`.
2. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.predixcloud.identity.config.jwt.enabled</code>	Use a JSON Web Token (JWT) Bearer Token as a means of requesting an OAuth2 access token. Requires <code>clientid</code> and a device certificate obtained through certificate enrollment.	String	N/A	Yes
<code>com.ge.dspmicro.predixcloud.identity.config.enrollment.url</code>	The Predix cloud enrollment URL. This URL is called once when creating credentials during enrollment.	URL String	N/A	Yes
<code>com.ge.dspmicro.predixcloud.identity.config.uaa.url</code>	The Predix cloud User Account and Authentication (UAA) endpoint for getting a token using <code>client_credentials</code> grant type.	URL String	N/A	Yes
<code>com.ge.dspmicro.predixcloud.identity.config.deviceid</code>	The client ID assigned to the device. If this property has a value, this device is enrolled. If it is blank, the device is not enrolled.	String	N/A	No
<code>com.ge.dspmicro.predixcloud.identity.config.deviceidentity</code>	<b>Predix device identity</b> Must contain only lower case letters (a-z), numbers (0-9), and : - _ . Must begin with a letter or number.	String	N/A	No
<code>com.ge.dspmicro.predixcloud.identity.config.internaldeviceid</code>	<b>Predix cloud internal device ID</b> . This is filled in by the enrollment process.	String		Yes
<code>com.ge.dspmicro.predixcloud.identity.config.tenantid</code>	The ID of the tenant. A tenant is an entity that is provisioned in the Predix platform with a UAA instance, hosted in its own org and space. This is filled in by the enrollment process.	String		No

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.predixcloud.enroll.deviceMac</code>	Predix device MAC address used to pin certificate to device. Acceptable formats are 6 bytes of case insensitive hex, with each byte separated by ':', '-' or none. For example: xx:xx:xx:xx:xx:xx or xx-xx-xx-xx-xx-xx or XxXxXxXxXxXx	String		Yes
<code>com.ge.dspmicro.predixcloud.enroll.url</code>	Used for uploading configuration commands from the device.  If not set, the device ID is appended automatically to the end of the URL.  When the device is enrolled this value is set automatically.	String		No
<code>com.ge.dspmicro.predixcloud.enroll.validateUrl</code>	The cloud service for validating installation packages.  This value is automatically set when the device is enrolled.	String		No
<code>com.ge.dspmicro.predixcloud.enroll.sharedSecret</code>	The shared secret assigned to the device when using certificate enrollment.	String		Yes
<code>com.ge.dspmicro.predixcloud.enroll.encryptedSharedSecret</code>	The encrypted shared secret assigned to the device when using certificate enrollment.	String		Yes

## Token Management

When you use the HTTP Client and WebSocket River, OAuth access tokens are managed automatically when the `ITokenRequester` service interface is registered upon device enrollment. If the device is not enrolled, the service is not available.

For example, if a service requires a UAA access token, it can use the `ITokenRequester` to request the token without understanding the grant type the system is configured for.

## Dependencies

A Maven dependency and an OSGi import are required to consume the service.

- The following Maven dependency is required to consume the service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>httpclient</artifactId>
  <version>{Predix Machine version}</version>
```

```
</dependency>
```

- The following OSGi import is required in the consuming bundle:

```
Import-Package: com.ge.dspmicro.httpClient.api;version="[1.3,2)"
```

## Obtaining the Token Management Service

Add a reference to the Token Management service.

Inject the `ITokenRequester` service using declarative services.

The following examples shows how to inject the service.

```
import com.ge.dspmicro.httpClient.api.ITokenRequester;

...
@Reference
public void setService(ITokenRequester service)
{
    //set service here
}
```

## *WebSocket River*

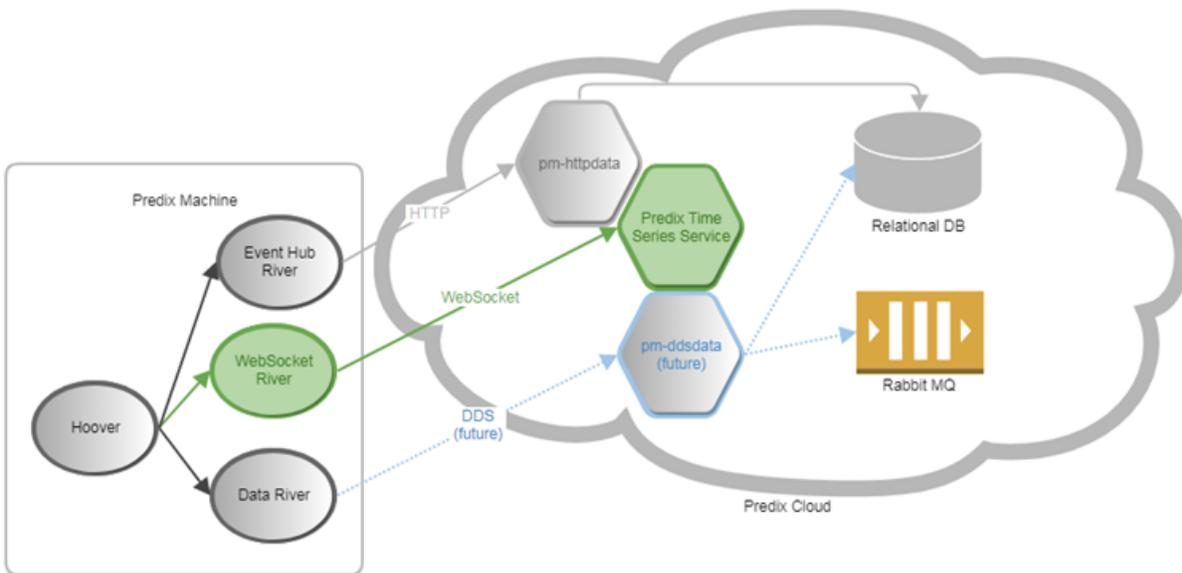
### *WebSocket River*

The WebSocket River provides connectivity between a Predix Machine-enabled device and the Predix Time Series service in the cloud.

 **Note:** See Time Series Service ([page](#) ).

The following figure illustrates the use case between the WebSocket River and the Predix Time Series service.

Figure: WebSocket River Use Case



## WebSocket River to Time Series Service Data Transfer

The WebSocket River establishes a connection when you first attempt a data transfer and keeps that channel open for as long as possible. Each data transfer verifies that the WebSocket connection is open. If the connection has been closed, the service opens a new connection. The `send` operation is thread-safe, as only one active WebSocket connection exists at any given time.

Requests from the WebSocket Client to the WebSocket Server are asynchronous. To make sure that all requests are received by the Predix Time Series service, which acts as the WebSocket Server in this case, a management thread keeps a schedule of timeouts for all requests.

## Data Format

The Time Series service requires data to be structured in a specific format as shown in (page ). To accommodate the Time Series format requirements, note the following Time Series format shown in the following example:

```
{
  "name": "sensor_tag_1234",
  "datapoints": [
    [
      1435854353541,
      4
    ],
    [
      1435854356541,
      2
    ]
  ]
}
```

```

    "attributes": {
      "asset_id": "UA231231",
      "site_id": "LOC12"
    }
  }
}

```

WebSocket River's `send()` method accepts data in this format or in the format of a serialized `PDataValue` object. A sample for serializing a `PDataValue` object to the required format is shown below:

```

ByteArrayOutputStream out = new ByteArrayOutputStream();
out.write(' ');
out.write(pDataValue.toBytes());
out.write(' ');

WebSocketRiverRequest request = new WebSocketRiverRequest();
request.setData(pDataInputStream);
transferId = wsRiverSend.send(request);

```

If you are using WebSocket River with Spillway and the provided Machine Adapters (such as OPC-UA, Modbus, or HealthMonitor), this conversion will happen automatically. You do not need to perform any transformation in this case.

## Limitations

Be aware of the following limitations when using the WebSocket River:

- The WebSocket River is designed to be used with the Predix Time Series service, so you can send data to the Predix Time Series service only.

 **Note:** To send data to a WebSocket endpoint other than the Time Series service, you can use the WebSocket Client Service to enable your application to establish a WebSocket connection with a WebSocket server. See [WebSocket Client Service \(page 238\)](#).

- You can pass data to the WebSocket River only in the format specified in the Data Format section.
- The order in which requests are acknowledged is not guaranteed because communication is asynchronous.

## Supported Numeric Data Types

Type	Contains	Size	Range
int	Signed integer	32 bits	-2147483648 to 2147483647

Type	Contains	Size	Range
long	Signed integer	64 bits	-9223372036854775808 to 9223372036854775807
float	IEEE 754 floating point	32 bits	$\pm 1.4E-45$ to $\pm 3.4028235E+38$
double	IEEE 754 floating point	64 bits	$\pm 4.9E-324$ to $\pm 1.7976931348623157E+308$

 **Note:** Boolean values are converted to numeric data types. Boolean value of `True` is converted to `1` and `False` is converted to `0`.

## Dependencies

Maven dependencies and an OSGi import package are required to consume the service:

- The following Maven dependency is required in the `pom.xml` file:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>websocketriver-send</artifactId>
  <version>{version number}</version>
</dependency>
```

- The following OSGi import is required to consume the bundle:

```
Import-Package:
  com.ge.dspmicro.websocketriver.send.api;version="[1.0,2)"
```

## Obtaining the WebSocket River Send Service

Add a reference in your service to the WebSocket River Send service.

Inject the WebSocket River Send service using declarative services.

The following example shows how to inject the service.

```
import com.ge.dspmicro.websocketriver.send.api.IWebsocketSend;

...

@Reference
public void setService(IWebsocketSend service)
{
    //set service here
}
```

## Using the WebSocket River Send API

The WebSocket River Send APIs are used to implement the send service on the local machine.

Review the WebSocket River Send APIs to understand how to implement the WebSocket River.

1. Navigate to and extract all files in the following file, <SDK installation location>/docs/apidocs.zip
2. Navigate to and open the Javadoc API, <SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.websocketriver.send.api.

## Configuring the WebSocket River Send Service

The WebSocket River Send service links the machine configured with a Websocket River to the time series cloud.

The WebSocket River service configuration in Predix Machine must be compatible with the configuration of the Time Series service running in the cloud.

- The UAA token URL in the Predix Machine Cloud Identity Management Service configuration must be a trusted issuer for the Time Series service instance in the cloud.
- The header name and value for the Predix Zone ID must match the configuration of the Time Series service instance in the cloud. These values can be retrieved by running a `cf env` command for the application bound to the Time Series service. Sample values are shown below.

```
"ingest": {
  "uri": "wss://ingestion_url",
  "zone-http-header-name": "Predix-Zone-Id",
  "zone-http-header-value": "9378e3db-e683-46a2-97c2-ccd11d75869d",
  "zone-token-scope": ["timeseries.zones.9378e3db-e683-46a2-97c2-ccd11d75869d.user", "timeseries.zones.9378e3db-e683-46a2-97c2-ccd11d75869d.ingest"]
}
```

- The client that is assigned to the Predix Machine instance must have the authorities listed in the zone-token-scope section of the previous example. In this example, these authorities are

```
timeseries.zones.9378e3db-e683-46a2-97c2-ccd11d75869d.user and
timeseries.zones.9378e3db-e683-46a2-97c2-ccd11d75869d.ingest
```

You can configure your implementation of WebSocket River.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to and open the following file: `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.websocketriver.send-[n].config`.
2. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.websocketriver.send.river.name</code>	A human-readable name for the river. This property links the river to other services such as Spillway.	String	"WS\ Sender\ Service"	Yes
<code>com.ge.dspmicro.websocketriver.send.destination</code>	The full URL of the WebSocket server to which data should be transmitted. It must begin with <code>ws://</code> or <code>wss://</code>	String	N/A	Yes
<code>com.ge.dspmicro.websocketriver.send.header.zone</code>	The name of the header where the zone ID should be inserted..	String	N/A	No
<code>com.ge.dspmicro.websocketriver.header.zone.value</code>	The zone ID to be inserted into the header of the initial WebSocket request.	String	N/A	No
<code>com.ge.dspmicro.websocketriver.send.timeout</code>	The timeout in milliseconds to wait for a response before assuming a transfer has failed.	Integer	"10000"	Yes
<code>com.ge.dspmicro.websocketriver.send.compression</code>	Whether or not to use Gzip compression when transmitting data.	Boolean	False	No
<code>com.ge.dspmicro.websocketriver.send.memory.limit</code>	The maximum memory to allocate for send buffers for the River.	Integer	L"0"	No

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.websocketriver.send.tag.augment</code>	Set to true if tags sent to Time Series should be augmented using a pattern.	Boolean	False	No
<code>com.ge.dspmicro.websocketriver.send.tag.pattern</code>	The pattern to use for modifying tags.  Only two variables are supported for substitution — <code>\${deviceId}</code> (device id) and <code>\${tagName}</code> (the original tag name).  Apart from the variables, the pattern can contain alphanumeric characters, '.', '-', '_'	String		No
<code>com.ge.dspmicro.websocketriver.send.data.format</code>		String	PDATAVALUE	No

## *Event Hub River Service*

### *Event Hub River Service*

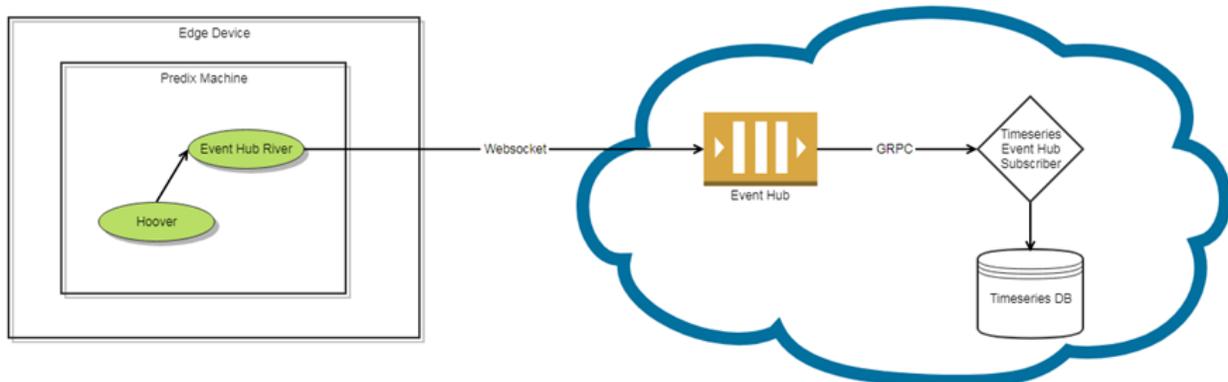
The Event Hub River service is a plugin service in Predix Machine that implements the `IRiver` interface to allow Event Hub to ingest data from the Machine Gateway component in Predix Machine.

Event Hub River provides connectivity between Predix Machine and the Event Hub service. Services that store or process data in the cloud must have custom subscriber applications that read the data provided in Event Hub and convert the format to the service's format for storage and processing.

 **Note:** If the device is sending time series data only, it is recommended that you use the WebSocket River service, which is optimized to handle time series data by formatting and compressing for time series at the device. Event Hub should be used if the device is sending other types of data (data not solely intended for time series), or if you want to perform processing on the time series data in the Predix cloud before injecting the data into Time Series.

Event Hub River service is available as an OSGi bundle in the Predix Machine JVM runtime, as well as a containerized Docker application that subscribes to the Predix Machine Data Bus and allows Event Hub to ingest the data subscriptions.

Event Hub River provides a single secure, reliable, fault-tolerant communication channel to the cloud so that data can move seamlessly among different cloud services regardless of the development language of choice.



This OSGi component conforms to the `IRiver` interface provided by Predix Machine, through which its data transfer API is invoked when there is data to transfer. The component then makes a WebSocket call to the configured Event Hub cloud service. The component then makes a GRPC call, using the Event Hub SDK, to the configured Event Hub cloud service.

The stand-alone containerized application subscribes to the Mosquitto MQTT broker that services the Predix Machine Data Bus, from where it has access to the data subscriptions to send.

The Event Hub cloud service offers synchronous and asynchronous transfer modes. Only the asynchronous mode is used in the Event Hub River service. Consequently, it has a Request Manager to track the timeout or acknowledgement of each data send request with `IRiverStatusCallback` objects.

## Example Use Case

- An edge device transmits data sources from Machine Gateway adapters to a cloud service like Predix Time Series by sending it through Event Hub River to the Event Hub service. A Time Series subscriber service (provided by a solution builder or cloud service provider) in the cloud subscribes to the Event Hub service and pushes the data to the Time Series service.
- Containerized Edge Analytics applications running on the edge device publish data onto the Predix Machine Data Bus, and the containerized Predix Machine Event Hub River subscribes to the data and publishes it to the Event Hub service for a cloud-service-oriented subscriber (for example, Time Series) to transmit it to the desired cloud endpoint.

## Data Format

Since the final destination of the data in the cloud is unknown, the data format that is sent is Predix Machine protobuf-based `EdgeDataList`, which has a list of `EdgeData` protobuf objects. Event Hub subscriber apps must use the `EdgeDataList` protobuf model to de-serialize the incoming

data. The following example shows the `EdgeDataList` and `EdgeData` formats (as shown in the `edge_data.proto` file):

```
import "google/protobuf/timestamp.proto";

package com.ge.predixmachine.protobuf;

option java_multiple_files = true;
option java_package = "com.ge.predixmachine.datamodel.datacomm";
option java_generate_equals_and_hash = true;

//
// Represents the data being passed around within edge device using Predix
// Machine Databus.
// EdgeData can be data collected through sensors, video clips, etc.
//
// Developer note: Tags with values in the range 1 through 15 take one byte
// to encode,
// including the identifying number and the field's type. Tags in the range
// 16 through
// 2047 take two bytes. So we should reserve the tags 1 through 15 for very
// frequently
// occurring message elements.
//
message EdgeData
{
    // Raw data
    repeated Data body = 1;

    // Data source ID - UUID
    string node_id = 2;

    // A user-friendly name of data source
    string node_name = 3;

    // URI of data source
    string node_uri = 4;

    // Timestamp when the data is created
    google.protobuf.Timestamp timestamp = 5;

    // Quality rating of data
    DataQuality quality = 6;

    // Category of data
    DataCategory category = 7;

    // Key-value pair for further identifying the data.
    map<string, string> custom = 100;
}

// Represents a list of EdgeData
message EdgeDataList
```

```
{
  repeated EdgeData edge_data = 1;
}

// Represents the actual data
message Data
{
  oneof data
  {
    double double_data = 1;
    float float_data = 2;
    int32 int_data = 3;
    int64 long_data = 4;
    bool boolean_data = 5;
    string string_data = 6;
    bytes binary_data = 7;
  }
}

// Quality of data content.
enum DataQuality
{
  // Default
  // Data is good
  GOOD = 0;
  // Data is bad
  BAD = 1;
  // Cannot be certain of data quality
  UNCERTAIN = 2;
  // Data quality is not supported
  NOT_SUPPORTED = 3;
}

// Categories to classify content.
enum DataCategory
{
  // Default
  // This Category only allows you to see real data being transferred.
  REAL = 0;
  // This Category only allows you to see test data being transferred.
  TEST = 1;
  // This Category only allows you to send ping message to test if a
  // service is listening to requests.
  PING = 2;
  // This category allows you to be notified on any status changes for
  // all transfers.
  ALL = 3;
}
```

## Limitations

The order in which requests will be acknowledged is not guaranteed because communication of responses is asynchronous.

## *Event Hub River Service Consumer Configuration*

### Maven Dependencies

The following Maven dependencies are required to consume this service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>eventhubriver-send</artifactId>
  <version>17.2.0</version>
</dependency>
```

### OSGI Imports

The following OSGi imports are required in the consuming bundle:

```
Import-Package: com.ge.dspmicro.eventhubriver.send.api;version="[1.0,2)",
com.ge.dspmicro.river.api;version="[1.0,2)"
```

## *Obtaining the Event Hub River Service*

Add a reference to the Event Hub River service in your service.

Inject the `IRiver` service using declarative services.

You must specify the target of the `IRiver` service to inject to match the river you intend to use. This target should match the `eventhubriver.send.river.name` property set in the `com.ge.dspmicro.eventhubriver.send-0.config` file.

The following example shows how to inject the `IRiver` service:

```
import com.ge.dspmicro.river.api.IRiverSend;
...
@Reference(target = "(com.ge.dspmicro.eventhubriver.send.river.name=Event
Hub Sender Service)")
public void setService(IRiverSend service)
{
  //set service here
}
```

## Configuring the Event Hub River Service

The configuration for the Event Hub River service must be compatible with the Event Hub service instance provisioned in the cloud.

You must have an Event Hub service instance provisioned in the cloud. See (page       ).

The VCAP\_SERVICES environment variables show the URI and Zone ID to use for accessing the Event Hub service, for example:

```
"predix-event-hub": [
  {
    "credentials": {
      "publish": {
        "protocol_details": [
          {
            "protocol": "grpc",
            "uri": "event-hub-aws-usw02.data-services.predix.io:443",
            "zone-token-scope": [
              "predix-event-hub.zones.<predix_zone_id>.user",
              "predix-event-hub.zones.<predix_zone_id>.grpc.publish"
            ]
          }
        ],
      },
    },
    {
      "protocol": "wss",
      "uri": "wss://event-hub-aws-usw02.data-services.predix.io/v1/stream/messages/",
      "zone-token-scope": [
        "predix-event-hub.zones.<predix_zone_id>.user",
        "predix-event-hub.zones.<predix_zone_id>.wss.publish"
      ]
    }
  ]
}
```

1. Navigate to, and open the `<Predix_Machine_Runtime_Container_Location>/configuration/machine/com.ge.dspmicro.eventhubriver.send-0.config` file.
2. Set values for the following properties:

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte

S = Short	C = Character	B = Boolean
-----------	---------------	-------------

Array format is property=<type>["<value1>","<value2>"]. For example, array of integer property=I["1", "2", "3"]. A backslash may be used to break up the line for clarity.

Property	Description	Type	Required	Default Value
com.ge.dspmicro.eventhubriver.send.publish.uniqueName	A friendly and unique name for the Event Hub River.	String	Yes	"Event Hub Sender Service"
com.ge.dspmicro.eventhubriver.send.publish.proxyPort	Port on the stand-alone container host machine.	Integer		I"14941"
com.ge.dspmicro.eventhubriver.send.publish.url	The URL of the Event Hub endpoint. For WebSocket, this must begin with wss://, which is the only currently supported protocol.	String	Yes	
com.ge.dspmicro.eventhubriver.send.publish.headers.zoneId	The name of the header where the zone ID will be inserted	String	No	"Predix-Zone-Id"
com.ge.dspmicro.eventhubriver.send.publish.headers.zoneIdValue	The zone ID for the Event Hub service instance.	String	Yes	
com.ge.dspmicro.eventhubriver.send.publish.timeout	The timeout in milliseconds to wait for a response before assuming a transfer has failed.	Integer	No	I"10000"
com.ge.dspmicro.eventhubriver.send.publish.format	Format of the data in the stream. Current values are PDATAVALUE and EDGEDATA.	String	No	"EDGEDATA"

## HTTP Data Service

### HTTP Data Service

Predix Machine collects data from various sensors and devices and forwards the collected data to the Predix cloud through the Event Hub River service.

The HTTP Data service receives the data from the Event Hub River service and persists it to a relational database in the Predix cloud. A sample application is provided for you to illustrate how to use this service. Navigate to `<Predix Machine SDK installation location>/samples/sample-cloud-apps.zip` and extract the files to view the `httpdata` sample. A *Read Me* file is provided for your reference.

## *Command Framework*

### *Command Framework Overview*

You can use the Command Framework service to send commands to applications that are running on a device.

For example, the Edge Manager device management application in the Predix cloud can send commands to individual applications for any device that uses the Command Framework service.

To pass a command from the Edge Manager, Predix Machine needs a command handler for each application it tries to reach. Each command handler should be implemented to communicate with a specific application.

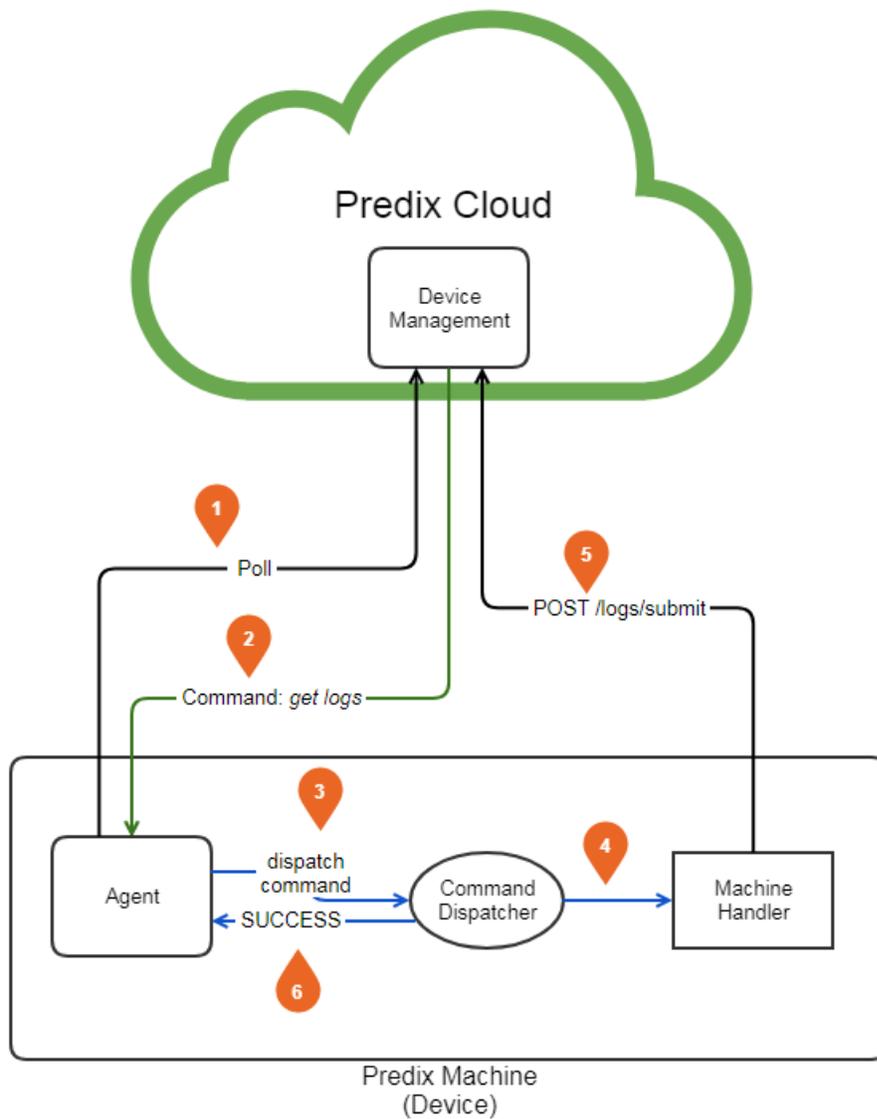
 **Note:** See [Creating a Command Handler \(page 94\)](#)

### **Workflow**

This example shows a workflow where a machine command handler executes a `getlogs` command.

1. Cloud Gateway performs a regular poll to check for commands.
2. Device Management sends the `get logs` command to the device.
3. Agent dispatches the `get logs` command.
4. Command Dispatcher identifies the command as intended for the Machine and sends a message to the Machine Handler.
5. The Machine Handler performs the requested action by posting the logs to the Device Management endpoint.
6. Command Dispatcher sends a SUCCESS status message to Agent.

The below figure illustrates the workflow.



## Command Framework Consumer Configuration

### Maven Dependencies

The following Maven dependencies are required to consume the Command Framework service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>device-common</artifactId>
  <version>17.x.x</version>
</dependency>
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>cloud-gateway</artifactId>
  <version>17.x.x</version>
```

```

</dependency>
<dependency>
  <groupId>com.ge.predixmachine</groupId>
  <artifactId>protobuf-models</artifactId>
  <version>17.x.x</version>
</dependency>

```

## OSGI Imports

The following OSGI imports are required in the consuming bundle:

```

Import-Package: com.ge.dspmicro.cloud.gateway.api;version="[1.0,2.0)",
Import-Package:
  com.ge.dspmicro.device.api.frameworkcomm;version="[2.0,3.0)"
Import-Package: com.ge.predixmachine.datamodel.gateway;version="[1.1,2.0)"
Import-Package:
  com.ge.predixmachine.datamodel.commandcomm;version="[1.0,2.0)"

```

## Installing the Command Framework into Predix Machine

To use the Command Framework service, you must generate a Predix Machine provisioning container, which includes the Command Framework and Cloud Gateway bundles.

You can generate the container using a command-line script or the Predix Machine SDK in Eclipse.

- Use a command-line script to generate the container.

 **Note:** For complete instructions on using scripts to generate a container, see [Generating a Predix Machine Runtime Container using Command Line Scripts \(page 33\)](#).

- Use the Predix Machine SDK to generate the container and add the **Predix Provisioning support** bundle in the **Predix Features Group** bundle type.

 **Note:** For complete instructions on using Eclipse to generate a container, see [Generating a Predix Machine Runtime Container Using Eclipse \(page 35\)](#).

## Creating a Command Handler

To subscribe to commands, an application must register a command handler service implementing the `ICommandHandler` interface. This service will be registered to the dispatcher and called when new commands are dispatched.

The following example shows how to implement a command handler:

```

import com.ge.dspmicro.cloud.gateway.api.ICloudGateway;
import com.ge.dspmicro.device.api.frameworkcomm.ICommandHandler;
import com.ge.predixmachine.datamodel.commandcomm.CommandStatus;
import com.ge.predixmachine.datamodel.commandcomm.EdgeCommand;

```

```

import com.ge.predixmachine.datamodel.gateway.TaskStatus;
...
@Component
public class SampleCommandHandler implements ICommandHandler
{
    public static final String HANDLER_TYPE = "SampleApplicationName";
    private ICloudGateway cloudGateway;
    ...

    @Override
    public void processCommand(EdgeCommand command)
    {
        // Get command information from EdgeCommand protobuf object
        String commandId = command.getId();
        String cmd = command.getCommand();
        String url = command.getUrl(); // URL to upload command output
        Map<String, String> params = command.getParamsMap();

        // Handle Command adding execution logs to a string buffer.
        StringBuffer executionLogs = new StringBuffer();
        executionLogs.append("Command execution logs");

        // Build a CommandStatus Object with command execution results.
        CommandStatus.Builder cmdStatusBuilder =
        CommandStatus.newBuilder();
        cmdStatusBuilder.setTaskId(commandId);
        cmdStatusBuilder.setStatus(TaskStatus.SUCCESS);
        cmdStatusBuilder.setStatusMessage("Successfully finished command
        execution.");

        cmdStatusBuilder.setStatusDetailedMessage(executionLogs.toString());
        cmdStatusBuilder.setOutput("Optional String Output. Up to
        1KB."); // Note: If output is returned through this status object, the
        output uploaded with the provided URL will be ignored.

        // Submit the CommandStatus to the Cloud Gateway to upload to the
        cloud
        this.cloudGateway.submitStatus(cmdStatusBuilder.build());
    }

    @Override
    public String getKey()
    {
        return HANDLER_TYPE;
    }

    /**
     * @param cloudGateway The Cloud Gateway
     */
    @Reference

```

```

public void setCloudGateway(ICloudGateway cloudGateway)
{
    this.cloudGateway = cloudGateway;
}

/**
 * @param cloudGateway The Cloud Gateway
 */
public void unsetCloudGateway(ICloudGateway cloudGateway)
{
    this.cloudGateway = null;
}
}

```

## Commands and Command Formats

Command formats for protobuf objects sent to Predix Machine and returned to the Cloud Gateway.

### EdgeCommand Protobuf Object sent to Machine

```

{
  "id": "Command ID",
  "commandHandlerType": "Command Handler Name/Type",
  "appName": "Name of the Container this command is intended
for",
  "command": "Command to execute",
  "url": "URL to post output",
  "params": {
    "Command parameter (i.e. PollingTime)": "Value (i.e.
30)"
  }
}

```

### CommandStatus Protobuf Object returned to CloudGateway

```

{
  "taskId": "ID found in EdgeCommand received by Machine
Command Handler",
  "status": "Status of type TaskStatus",
  "statusMessage": "Status Message",
  "statusDetailedMessage": "Status Detailed Message",
  "output": "Command String Output up to size 1KB. If output
is returned through this status object, the output uploaded
with the provided URL will be ignored."
}

```

## Commands

The following commands are provided by the Predix Machine Command Handler.

<b>Display Name</b>	<b>Command</b>	<b>Parameters</b>	<b>Operation Description</b>
Predix Machine: Refresh	refresh		Restarts the Predix Machine OSGi container.
Predix Machine: Restart	restart		Restarts the Predix Machine OSGi container process. This command is not permitted if the watchdog is not running.
Predix Machine: Upload Configurations	uploadconfig		Uploads the current machine configuration files.
HTTP Tunnel: Enable	enablehttptunnel		Enables the Http Tunnel Client.
HTTP Tunnel: Disable	disablehttptunnel		Disables the Http Tunnel Client.
Technician Console: Enable	enableweb		Enables the Technician Console and restarts the container.
Technician Console: Disable	disableweb		Disables the Technician Console and restarts the container.
Predix Machine: Get Log	getlogs	File Name: "machine:// machine/ machine.log"	Posts the specified log file from the Predix Machine OSGi Container to a predefined URL endpoint in device manager. If no name is specified, the current log file is posted. Log files follow a URI format: <application_name>:/<path_to_file>. The default application is machine, which posts machine.log. Other applications in containers can be referenced using the scheme portion of the URI.
Predix Machine: Set Polling Interval	setpolling	Polling Interval Time: 30	Sets the device manager polling interval to the value passed. This sets the configuration file: com.ge.dspmicro.cloud.gateway.config

Display Name	Command	Parameters	Operation Description
Predix Machine: Get IP Address	ipaddress		<p>Posts the network interface information with all available device IP addresses to the provided URL endpoint. Following is a sample format, which will be in the POST body of the return call:</p> <pre data-bbox="792 436 1414 867"> [   {     "name": "lo0",     "displayName": "TCP Loopback interface",     "inet4": ["127.0.0.1 / 8"],     "inet6": ["0:0:0:0:0:0:1", "fe80:0:0:0:0:0:1/64"]   },   {     "name": "en0",     "displayName": "Wireless Network Connection",     "inet4": ["3.39.57.3/23"],     "inet6": ["fe80:0:0:0:a299:9bff:fe10:4773/64"]   },   {     "name": "utun0",     "displayName": "utun0",     "inet4": [],     "inet6": ["fe80:0:0:0:59a7:74cb:9139:5aaa/64"]   } ] </pre>
Predix Machine: Get Available Logs	get_available_logs		<p>Returns a list of available logs on the machine. The format provides a URI. The scheme in the URI indicates which application the log pertains to and the path indicates which log file in that application. For all Predix Machine logs, the root of the path begins in the machine log folder. The following contents will be in the POST body of the return call.</p> <pre data-bbox="792 1087 1414 1255"> [   {     "filepath": "machine:/installations/yetil.log"   },   {     "filepath": "machine:/machine/machine.log"   },   {     "filepath": "machine:/machine/machine-1.log"   } ] </pre>
Predix Machine: List installed packages	run_dpkg		<p>Lists all packages installed on the device.</p> <p> <b>Note:</b> Only for Linux devices that support dpkg.</p>
Predix Machine: Run ifconfig command	run_ifconfig		<p>Runs the system command ifconfig on the device and returns the command output.</p> <p> <b>Note:</b> This applies only to devices that support the ifconfig command.</p>
Predix Machine: Run top command	run_top		<p>Runs the system command top on the device for one iteration.</p> <p> <b>Note:</b> This applies only to devices that support the top command.</p>

Display Name	Command	Parameters	Operation Description
Predix Machine: Run journalctl command	run_journalctl	<p><b>Filtering</b> parameters include:</p> <ul style="list-style-type: none"> <li>• <code>component</code> (String): <ul style="list-style-type: none"> <li>◦ Shows entries generated by the specified component.</li> <li>◦ On Linux, the path of the executable, for example: <code>/bin/systemd</code></li> </ul> </li> <li>• <code>priority</code> (Integer from 0 to 7): <ul style="list-style-type: none"> <li>◦ Shows entries with the specified log level, or a more important log level.</li> <li>◦ On Linux, valid log levels are: <ul style="list-style-type: none"> <li>■ Emergency (0)</li> <li>■ Alert (1)</li> <li>■ Critical (2)</li> <li>■ Error (3)</li> <li>■ Warning (4)</li> <li>■ Notice (5)</li> <li>■ Info (6)</li> <li>■ Debug (7)</li> </ul> </li> </ul> </li> <li>• <code>since</code> (date time value in <code>YYYY-MM-DD HH:mm:ss</code> format):  Shows entries on or newer than the specified timestamp.</li> <li>• <code>until</code> (date time value in <code>YYYY-MM-DD HH:mm:ss</code> format):  Shows entries</li> </ul>	<p>Uses system command <code>journalctl</code> to query the default system journal on the device according to the filtering and formatting values specified in the command parameters.</p> <p>All command parameters are optional. However, journals usually contain a lot of entries, therefore, it is best practice to use the filtering parameters to reduce the size of the output to send to the cloud.</p> <p> <b>Note:</b> Only for Linux devices that support <code>journalctl</code>.</p>

Display Name	Command	Parameters	Operation Description
Predix Machine: Refresh Device Detail	refresh_device_detail		<p>Sends all device detail information in the next sync. This command overrides the Cloud Gateway's polling interval (<code>com.ge.dspmicro.cloud.gateway.pollingInterval</code>) and schedules a sync immediately.</p> <p>This command also overrides the sync intervals of the following device detail categories. Information about the following, together with all other available device detail information, will be included in the next sync.</p> <ul style="list-style-type: none"> <li><code>com.ge.dspmicro.cloud.gateway.machineInfoUploadInterval</code></li> <li><code>com.ge.dspmicro.cloud.gateway.deviceStatusUploadInterval</code></li> <li><code>com.ge.dspmicro.cloud.gateway.dockerInfoUploadInterval</code></li> </ul>

## Command Output

Output for a command should be returned only in one of the two ways listed below. If both ways are implemented for a command, only the output in the `CommandStatus` object (first way) will be displayed in the Edge Manager UI.

1. Output is sent through the `CommandStatus` object in the `Output` field.

 **Note:** This is limited to String output of size < 1KB.

2. Output is uploaded through the provided URL.

## Command Status Enum

The following table describes the enum values and descriptions for settings in the `CommandStatus` object.

Enum	Description
ACKNOWLEDGED	The task is acknowledged by the device.
SUCCESS	The task completed successfully.
DELIVERED	The task has been delivered to an application/service to handle.
FAILURE	The task failed to complete.
NOTSUPPORTED	The task cannot be processed.

## Dispatching a Command to a Command Handler

Inject the Command Framework service using declarative services, for example:

```

import com.ge.dspmicro.device.api.frameworkcomm.ICommandDispatcher;
import com.ge.predixmachine.datamodel.commandcomm.EdgeCommand;
...
@Component(name = "CustomDipatcher")
public class SampleApp_DispatchingCommand
{
    private ICommandDispatcher commandDispatcher;
    ...

    public void processCommand()
    {
        // Create a command for a command handler to process
        EdgeCommand edgeCmd = EdgeCommand.newBuilder()
            .setCommand("CommandName")
            .setCommandHandlerType("CommandHandlerName")
            .setUrl("URL to upload command execution output, if any.")
            .putParams("Param1", "10")
            .putParams("Param2", "20")
            .build();

        // Pass command to Command Dispatcher so it dispatches to
        corresponding command handler
        this.commandDispatcher.dispatchCommand(edgeCmd);
    }

    /**
     * @param commandDispatcher The Command Dispatcher
     */
    @Reference
    public void setDispatcherCommand(ICommandDispatcher commandDispatcher)
    {
        this.commandDispatcher = commandDispatcher;
    }

    /**
     * @param commandDispatcher The Command Dispatcher
     */
    public void unsetCloudGateway(ICommandDispatcher commandDispatcher)
    {
        this.cloudGateway = null;
    }
}

```

## Machine Command Handler

You can use the Machine Command Handler to communicate with Predix Machine. For example, Edge Manager sends commands to the Machine Command Handler, which can enable or disable the Predix Machine Web Console.

### Configuring the Machine Command Handler

You configure the Machine Command Handler.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to and open the following file: `<Predix Machine runtime container location>/configuration/machine`.
2. Open the `com.ge.dspmicro.device.commandhandler.machinecomm.config` file and set values for the following optional properties:

Property	Description	Type	Default Value
<code>com.ge.dspmicro.device.commandhandler.machinecomm.config.additional.upload.folders</code>	List of folders to upload. This is a pipe separated list of folder names.	String	<pre>[ "configuration/machine", "configuration/pfa", "configuration/yeti", ]</pre>
<code>com.ge.dspmicro.device.commandhandler.machinecomm.config.additional.upload.url</code>	Alternative or additional upload other than predix cloud upload. Use two way TLS.	String	

# Connect Data to the Edge

## *Edge Services Overview*

### *About Connecting Data to the Edge*

In the Predix edge-to-cloud platform, the edge is the physical location that allows computing closer to the source of data.

Edge services enable you to establish machine-to-machine, machine-to-cloud, and machine-to-mobile connections.

The services that support these connections include the following.

#### **Core Services**

Includes services that support logging and provide security and certificate management.

#### **Application Services**

Includes services that support user management and the Git repository.

#### **Machine Gateway Services**

Includes services that support the machine gateway, which uses industrial protocols like OPC-UA, Modbus, and MQTT and their corresponding adapters, perform XML configuration monitoring, provide data store and forward, support route validation, provide route pinging, and allow you to determine machine health.

#### **Cloud Gateway Services**

Includes services that provide APIs to build client-side HTTP-compliant applications, allow communication of different network protocols with tunneling, and establish proxy settings.

#### **Mobile Gateway Services**

Includes the WebSocket Server service.

## Core Services

### Logging

Predix Machine uses the Pax Logging facility. Pax Logging defines its own API, and also supports the Log4J, SLF4J, JDK Logging, and Apache Commons Logging APIs, making it easy for applications to use these common APIs. Additional information on Pax Logging can be found at: <https://ops4j1.jira.com/wiki/display/paxlogging/Pax+Logging>.

**!** **Important:** Certain permissions are required for logging:

```
(org.osgi.framework.AdaptPermission
"(adaptClass=org.osgi.framework.wiring.BundleRevision)" "adapt")
```

### Logging Service Sample Application

Sample usage apps are found at <SDK installation location>/samples. These samples provide code snippets on how to use the Logging Service.

### Logging Service Configuration

To change logging configuration settings, change the <Predix Machine runtime container location>/configuration/machine/org.ops4j.pax.logging.cfg file.

By default, Predix Machine uses two log files:

- System messages specifying the bundle state are sent to the <Predix Machine runtime container location>/machine/bin/vms/jdk/logs/log.txt file. An example of this log file is shown below:

```
#0 INFO > Bundle with id #33 org.ops4j.pax.logging.pax-logging-api, 1.7.1
was started.

#0 INFO > Bundle with id #34 org.ops4j.pax.logging.pax-logging-service,
1.7.1 was started.
```

- Predix Machine bundles use the Pax Logging configuration. These logs are sent to the <Predix Machine runtime container location>/logs/machine/machine.log file. An example of this log file is shown below:

```
2015-03-10 18:19:27,994[Component Resolve Thread (Bundle 33)]|INFO|
com.ge.dspmicro.management.impl.SimpleLoginImpl
```

```
|61-com.ge.dspmicro.management-impl-15.1.0|Default administrative user
"predix" successfully created.
```

## *Bundle Security Service*

Predix Machine offers a strict but flexible set of security policies. A defined set of security permissions is invoked at startup. These can apply to bundles, system bundles, and the overall framework.

### **Permissions in Bundle Security Service**

The OSGi security model is an extension of the Java 2 security architecture. The Conditional Permission Admin (CPA) is a powerful feature that is added to the Java 2 model. The CPA uses an API based on policies. A policy is defined in the specification as follows:

*"Policies contain a set of permissions that are applicable when the related conditions are met. A policy can both allow (the Java 2 model) as well as deny access when the permissions are implied. Deny permissions can significantly simplify security management.*

*The real time management of Conditional Permission Admin enables management applications to control the permissions of other applications with immediate effect; no restart is required.*

*Policies are based on the very general concept of conditions. Conditions guard access to the policy's permissions. If they are not satisfied, then the permissions are not applicable. Conditions can be based on ... the bundle location, as well as on user-defined conditions." - OSGi R4v4.2 spec. The specification document can be referenced using this link - <http://www.osgi.org/Release4/Download>. Refer to section "50 Conditional Permission Admin Service Specification".*

The Predix Machine runtime container loads these permissions from each bundle using an OSGi-INF/permissions.perm file packaged in the bundle JAR file.

The format of the permission file only lists permissions; it does not specify to allow or deny because it only applies to the current bundle. A sample file can be found at <SDK installation location>/samples/sample-apps/sample-configuration/src/main/resources/OSGi-INF/permissions.perm.

If the local permission resource is not included in a bundle, the framework will set `java.security.AllPermission` as local permissions for the bundle:

```
(java.security.AllPermission "*" "*")
```

This may not be desired for production bundles

Even though it is not recommended, all bundle security can be disabled in the startup script file; turn on Java security permissions in `../bin/predixmachine` or (`or predixmachine.bat`). Comment out the line to turn it off.

```
#FWSECURITY=on
```

## *Predix Machine Web Console and Bundle Updates*

The Predix Machine Web Console and Bundle Updates service installs the new executable with the new binaries. The components are updated without changing the certificate stores or the configurations. The Predix Machine Web Console and Bundle Updates service can also install a new bundle that was not previously in the container or configured in the startup files.

### Updating Bundles with the Predix Machine Web Console

To use the Predix Machine Web Console and Bundle Updates service, follow these steps:

 **Note:** The Predix Machine Web Console is only available in the DEBUG version of Predix Machine, or if you add it manually using the Predix SDK.

For more instruction about the web console, visit <http://felix.apache.org/documentation/subprojects/apache-felix-web-console.html>

1. Start the container.
2. From your web browser, enter this URL, (as an example if you are running the Predix Machine as a localhost):  
`https://localhost:8443/system/console`
3. Enter the default login. If you changed the password, use the password you have established.  
`login: predix password: predix2machine`

On first log in, you must change the password.

-  **Note:** Your password must:
- Be at least eight characters long and not more than 15 characters long
  - Contain at least two uppercase letters
  - Contain at least one lowercase letter
  - Contain at least two numbers
  - Contain at least one special character
  - Not contain the user name
  - Not contain spaces

The **Bundles** page appears.

#### 4. Update bundles.

a. Click the **Install/Update** button.

The **Upload/Install Bundles** dialog box appears.

b. Complete the fields, locate your file, and click **Install** or **Update**.

## *MetaType - Configuration Type Checking*

The OSGi Metatype specification provides a language to describe configuration information in an XML file. Any bundles that have a configuration file, for example `<Predix Machine runtime container location>/configuration/machine{config}.cfg` file) also need to include a metatype XML file in the bundle. The metatype XML adds type-checking for properties so that improper values are not added. It also creates readable fields in the Predix Machine Web Console.

A sample metatype XML file is located in the `<Predix Machine runtime container location>/machine/bundles/{unzipped .jar file}/OSGI-INF/com.ge.dspmicro.{configuration name}.xml` ....

### **bndlib**

Use the bndlib project to generate the metatype XML file for each bundle that contains configuration files. This generates the XML for you and also provides an interface for working with the properties.

Refer to <http://www.aqute.biz/Bnd/MetaType> for more information.

A sample usage app is found at: `<SDK installation location>/samples/sample-apps//sample/sample-configuration` (after you unzip the `sample-apps.zip` file).

## *Security Administration Service: SSL Context and Certificate Management*

Configuration of the Security Administration bundle is done differently than other Predix Machine services.

- The Security Administration bundle does not use the configuration administration service and instead parses the configuration file itself. This means that the configuration is not dynamic and *can only be performed once at startup*.
- The configuration file (`com.ge.dspmicro.securityadmin.cfg`) is located in the `<Predix Machine runtime container location>/security` folder

instead of the typical `<Predix Machine runtime container location>/configuration/machine` folder.

**!** **Important:** The `<Predix Machine runtime container location>/security` folder contains security files and information. In the deployment environment, this folder should be secured and restricted to users that have administrative rights.

Configurations for client and server KeyStores/TrustStores are separated due to the differences in trust and key usage when Predix Machine acts as a client as opposed to a server. For example, trust as a server directly impacts client authentication when 2-way TLS is required by Predix Machine. The trusted certificates in the Predix Machine client truststore *should not* be used as trust anchors to authenticate clients connecting to Predix Machine.

## Generated KeyStores

Predix Machine generates new KeyStores on startup if the Security Admin configuration is not configured to point to a KeyStore. Specifically, if a property indicates the path of a KeyStore is blank, that KeyStore is generated along with a random password. The respective configuration properties are written to the Security Admin configuration file. The keys generated by the startup script are of sufficient strength to use for most standard production use cases. The algorithms and key strengths are selected based on current NIST recommendations.

The passwords for KeyStores and keys are generated randomly. Solutions will often require access to the KeyStores for a variety of reasons. For example trusting an instance of Predix Machine for TLS connections requires exporting certificates from a KeyStore. When a solution requires access to the KeyStore, they can find the password populated in the Security Admin Configuration file in respective fields pertaining to that particular KeyStore.

## Importing a New Private Key and CA Signed Certificate

Instead of using a generated KeyStore, a solution can use a CA signed certificate.

1. Look up the KeyStore passwords in the `<Predix Machine container location>/security` directory containing the KeyStore.
2. Use a Java Keytool to import this key pair into the KeyStore.

This example shows how to import a certificate/private key pair into your KeyStore:

```
keytool -importkeystore
  -srckeystore key.jks -destkeystore machine_keystore.jks
  -srcstoretype JKS -deststoretype JKS
  -srcstorepass changeit -deststorepass topsecret
  -srcalias myprivatekey -destalias myoldprivatekey
  -srckeypass oldkeypass -destkeypass mynewkeypass
```

## Importing Certificates into a TrustStore

Applications on Predix Machine use a variety of TrustStores pertaining to general categories of services. HttpClient, Felix HTTP Service, and WebSockets in particular, use the server TrustStore configured in the Security Admin configuration.

1. Look up the TrustStore password in the `<Predix Machine runtime container location>/security/com.ge.dspmicro.securityadmin.cfg` file.
2. Use the Java Keytool to import the certificate into a TrustStore.

The following example shows how to import the certificate into a TrustStore:

```
keytool -keystore machine_truststore.jks -import -alias <create_an_alias> -
file <certificate_filename>
```

## Configuring the Client KeyStore/TrustStore

If a server requires client certificates, you can set this in the Security Admin configuration file.

- Configure the `sslcontext` KeyStores to point to a KeyStore with a single key. If the KeyStore contains multiple keys, the first one will be used as the client certificate.

Once these values are configured, two-way SSL is enabled if required by the server. Any service using the `SecurityAdmin SSLContextFactoryService` to obtain an `SSLContext` will be able to use this KeyStore to provide a client certificate. This includes `HttpClient`, `WSClient`, and any services using `HttpClient` and `WSClient`.

1. Navigate to `<Predix Machine runtime container location>/security`.
2. Open the Security Administration configuration file `com.ge.dspmicro.securityadmin.cfg`.
3. Configure the following properties:

Property	Description	Default Value
<b>SSL Context Truststore configuration</b>		
<code>com.ge.dspmicro.securityadmin.sslcontext.truststore.type</code>	The type of TrustStore, any type supported by the available security provider	JKS

Property	Description	Default Value
<code>com.ge.dspmicro.securityadmin.sslcontext.client.truststore.path</code>	The path to the TrustStore. The TrustStore path should be a partial path from system property <code>predix.home.dir</code> (root folder of Predix machine).	security/ machine_client_truststore.jks
<code>com.ge.dspmicro.securityadmin.sslcontext.client.truststore.password</code>	Password to the TrustStore.	dspmicro
<b>SSL Context KeyStore configuration.</b>		
<code>com.ge.dspmicro.securityadmin.sslcontext.client.keystore.type</code>	The type of the KeyStore. This value will be overwritten if the <code>com.ge.dspmicro.securityadmin.sslcontext.client.keystore.path</code> property value is not set.	JKS
<code>com.ge.dspmicro.securityadmin.sslcontext.client.keystore.path</code>	(Optional) The KeyStore path should be a partial path from system property <code>predix.home.dir</code> (root folder of Predix machine).   <b>Note:</b> Leave this property blank to generate a new KeyStore with a random password on startup. If this occurs, the remaining KeyStore properties will be overwritten.	/security/tls_client_keystore.jks
<code>com.ge.dspmicro.securityadmin.sslcontext.client.keystore.password</code>	Password for the KeyStore. This value will be overwritten if the <code>com.ge.dspmicro.securityadmin.default.keystore.path</code> property value is not set.	<generated>
<code>com.ge.dspmicro.securityadmin.sslcontext.client.keystore.alias</code>	The alias for the key to use from the client KeyStore. This value will be overwritten if the <code>com.ge.dspmicro.securityadmin.default.keystore.path</code> property value is not set.	none
<code>com.ge.dspmicro.securityadmin.sslcontext.client.keystore.privateKeyEntryPassword</code>	The password for the private key entry in the KeyStore. This value will be overwritten if the <code>com.ge.dspmicro.securityadmin.default.keystore.path</code> property value is not set.	none

## Configuring the Server KeyStore/TrustStore

Import certificates signed by a CA into this keystore. Import trusted client certificates into this truststore for 2-way TLS (client authentication).

1. Navigate to <Predix Machine runtime container location>/security

2. Open the Security Administration configuration file  
`com.ge.dspmicro.securityadmin.cfg.`

3. Configure the following properties:

Property	Description	Default Value
<b>SSL Context TrustStore configuration</b>		
<code>com.ge.dspmicro.securityadmin.sslcontext.truststore.type</code>	The type of KeyStore, any type supported by the available security provider	JKS
<code>com.ge.dspmicro.securityadmin.sslcontext.truststore.path</code>	The path to the TrustStore. The TrustStore path should be a partial path from system property <code>predix.home.dir</code> (root folder of Predix machine).	security/ machine_server_truststore.jks
<code>com.ge.dspmicro.securityadmin.sslcontext.truststore.password</code>	Password to the TrustStore	dspmicro
<b>SSL Context KeyStore configuration.</b>		
<code>com.ge.dspmicro.securityadmin.sslcontext.server.keystore.type</code>	The type of the KeyStore. This value will be overwritten if the <code>com.ge.dspmicro.securityadmin.sslcontext.server.truststore.path</code> property value is not set.	JKS
<code>com.ge.dspmicro.securityadmin.sslcontext.server.keystore.path</code>	(Optional) The KeyStore path should be a partial path from system property <code>predix.home.dir</code> (root folder of Predix machine).   <b>Note:</b> Leave this property blank to generate a new KeyStore with a random password on startup. If this occurs, the remaining TrustStore properties will be overwritten.	system/security/ tls_server_keystore.jks
<code>com.ge.dspmicro.securityadmin.sslcontext.server.keystore.password</code>	Password for the KeyStore.	<generated>
<code>com.ge.dspmicro.securityadmin.sslcontext.server.keystore.alias</code>	The alias for the key to use from the client KeyStore. property value is not set.	none
<code>com.ge.dspmicro.securityadmin.sslcontext.server.keystore.password</code>	The password for the private key entry in the KeyStore.	none

## Configuring the Default KeyStore

Use the default KeyStore for services that require issuing digital signatures. The STS service uses the key configured here to sign SAML tokens.

1. Navigate to <Predix Machine runtime container location>/security.
2. Open the Security Administration configuration file `com.ge.dspmicro.securityadmin.cfg`.
3. Set values for the following properties:

Property	Description	Default Value
<code>com.ge.dspmicro.securityadmin.default.keystore.type</code>	The type of the KeyStore. This value will be overwritten if <code>com.ge.dspmicro.securityadmin.default.keystore.path</code> is not set.	JKS
<code>com.ge.dspmicro.securityadmin.default.keystore.path</code>	(Optional) The path to the KeyStore relative to the container home directory.   <b>Note:</b> Leave this property blank to generate a new TrustStore with a random password on startup. If this occurs, the remaining TrustStore properties will be overwritten.	<code>security/misc_keystore.jks</code>
<code>com.ge.dspmicro.securityadmin.default.keystore.password</code>	The password for the KeyStore. This value will be overwritten if <code>com.ge.dspmicro.securityadmin.default.keystore.path</code> is not set.	<generated>
<code>com.ge.dspmicro.securityadmin.default.keystore.alias</code>	The alias for the key to use from the client KeyStore. This value will be overwritten if <code>com.ge.dspmicro.securityadmin.default.keystore.path</code> is not set.	None
<code>com.ge.dspmicro.securityadmin.default.keystore.privateKeyPassword</code>	The password for the private key entry in the TrustStore. This value will be overwritten if <code>com.ge.dspmicro.securityadmin.default.keystore.path</code> is not set.	None

## Configuring the Encryption KeyStore

The Encryption KeyStore contains a symmetric key used to encrypt sensitive configuration properties in the configuration files located in the <Predix Machine runtime container location>/configuration/machine folder.

1. Navigate to <Predix Machine runtime container location>/security.
2. Open the Security Administration configuration file  
com.ge.dspmicro.securityadmin.cfg.
3. Set values for the following properties:

Property	Description	Default Value
com.ge.dspmicro.securityadmin.default.keystore.type	The type of the KeyStore. This value will be overwritten if com.ge.dspmicro.securityadmin.default.keystore.path is not set	JCEKS
com.ge.dspmicro.securityadmin.default.keystore.path	The path to the KeyStore relative to the container home   <b>Note:</b> Leave this property blank to generate a new KeyStore with a random password on startup. If this occurs, the remaining TrustStore properties will be overwritten.	security/ secret_keystore.jceks
com.ge.dspmicro.securityadmin.default.keystore.password	The password for the KeyStore. This value will be overwritten if com.ge.dspmicro.securityadmin.default.keystore.path is not set	<generated>
com.ge.dspmicro.securityadmin.default.keystore.alias	The alias for the key to use from the client KeyStore. This value will be overwritten if com.ge.dspmicro.securityadmin.default.keystore.path is not set.	None
com.ge.dspmicro.securityadmin.default.keystore.privateKeyPassword	The password for the private key entry in the KeyStore. This value will be overwritten if com.ge.dspmicro.securityadmin.default.keystore.path is not set	None

## Configuring Other Properties

See <http://felix.apache.org> for full list of Felix HTTP service configuration properties. Any property beginning with "org.apache.felix" can be added to the security admin configuration file..

1. Navigate to <Predix Machine runtime container location>/security.
2. Open the Security Administration configuration file  
com.ge.dspmicro.securityadmin.cfg.
3. Set values for the following properties:

Property	Description	Default Value
<code>com.ge.dspmicro.securityadmin.JCEProvider</code>	Sets the provider name for the preferred JCE security provider.	SunJCE
<code>com.ge.dspmicro.securityadmin.xmlSchemaValidation</code>	<p>True/false. Global flag for whether or not to perform XML schema validation.</p> <p> <b>Warning:</b> Disabling can render HTTP services protected by SAML authentication vulnerable to malicious XML attacks. Only disable when absolutely necessary for debugging.</p>	true
<code>com.ge.dspmicro.securityadmin.tokenAuthenticationFilter</code>	True/false. Set to true to protect HTTP services with SAML authentication filter.	false
<code>com.ge.dspmicro.securityadmin.HttpsURLConnectionVerifier</code>	<p>True/false. Set to true to turn on no host name verifier for HttpsURLConnection.</p> <p> <b>Warning:</b> This should only be enabled for debugging. Enabling can result in connections being intercepted by untrusted and malicious parties.</p>	false
<code>org.apache.felix.http.enable</code>	true/false. Enables or disables Felix HTTP service over HTTP (insecure)	false
<code>org.apache.felix.https.enable</code>	true/false. Enables or disables Felix HTTP service over HTTPS (secure)	true

### Using Security Administration APIs

Review the Security Administration APIs to understand how to fully implement Security Administration.

1. Navigate to and extract all files in the following file: `<SDK installation location>/docs/apidocs.zip`
2. Review the Security Administration API: `<SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.security.admin.api`

### Using the Security Administration Sample Application

A sample Security sample application is provided to illustrate how to use this service.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.

2. In the `sample-apps/sample` folder, open the `sample-security` application.

 **Note:** See [Building Samples from a Command Line \(page 46\)](#) and [Running Samples in Eclipse \(page 48\)](#) or [Running Samples from Generated Containers \(page 48\)](#) for instructions on building and running the sample application.

## Application Services

### Management Service (Users)

The Management Service (Users) service uses roles to provide administrative functions. A user role is an entity with any number of authentication credentials. Normally, user roles are used to authenticate the initiator of an action. User roles include the following:

- Human users with a username and password.
- Machines with a hostname and SSL-certificate.

The Management Service (Users) service is available in the Predix Machine web console.

### Managing Users

Start your Predix Machine debug container and open the Predix Machine Web Console.

To change the admin user's default user name and password, the web console requires a login user for the **administration** group. You can add new users or change the password in the web console: `https://localhost:8443/system/console/bundles`. The default `management-impl` bundle implementation creates the following user at startup: **username:** `predix` **password:** `predix2machine` **group:** `administration`.

 **Note:** You can provide your own user name and password by replacing the `management-impl` in the container with a custom version that implements the `com.prosyst.mbs.services.useradmin.SimpleLogin` interface.

1. Open `https://localhost:8443/system/console/bundles`.
2. On the **Predix** menu, click **User Administration** to create users.
3. Perform any of these procedures:
  - Add a user.
    - a. Click the plus (+) icon.

The **Create User** dialog box appears.

- b. In the **Username** box, type the name of the new user.
- c. In the **Password** box, type the new password.
- d. In the **Verify Password** box, retype the new password.
- e. Click **Create User**.

The user appears in the **Users** list.

- Change a user password.
  - a. In the **Users** list, click the name of the user.
  - b. In the **Old Password box**, type the user's old password.
  - c. In the **New Password** box, type the new password.
  - d. In the **Verify Password** box, retype the new password.
  - e. Click **Change Password**.
- Remove a user.
  - a. In the **Users** list, click the name of the user.
  - b. Click the delete icon next to the user's name.
  - c. On the **Confirm Delete** dialog box, click **Delete User**.

The user is removed from the **Users** list.

## Using the Management Bundle Sample Application

A Management Bundle sample application is provided to illustrate how to use the service.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.
2. In the `sample-apps/sample` folder, open the `sample-management` application.

 **Note:** See *Building Samples* and *Running Samples in the Predix SDK* for instructions on building and running the sample application.

## *Git Repository Management Service*

The `IGitRepository helper` interface clones a repository on startup, or opens an existing clone, and then provides simple Git commands like: `clean`, `reset`, `pull` and `push`. The API gives access to the Git object and security provider for committing files and providing access to more advanced features. Configuration files are managed in the Git repository.

### **To get example code for JGit**

For examples and code snippets for the JGit Java Git implementation, click the following link:

<https://github.com/centic9/jgit-cookbook>

To download Javadoc for JGit, click the following link:

<http://download.eclipse.org/jgit/docs/latest/apidocs/>

To download public GUIs for accessing a Git repository, click the following link:

<http://git-scm.com/downloads/guis>

To access the Git command line documentation, click the following link:

<http://www.kernel.org/pub/software/scm/git/docs/>

To access a local or remote repository from inside Eclipse, install new software from inside Eclipse, to add Git support, or to add a URL:

<http://download.eclipse.org/egit/updates>

## Using the Git Repository APIs

Review the Git Repository Javadoc APIs to understand how to implement the Git Repository.

1. Navigate to and extract all files in the following file: `<SDK installation location>/docs/apidocs.zip`
2. Navigate to and open the following API: `<SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.gitrepository.api`.

## Using the Git Repository Sample Application

A Git Repository sample application is provided to illustrate how to use this service.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.
2. Navigate to and open `<SDK installation location>/sample-apps/sample/sample-gitrepository`.

 **Note:** See *Building Samples* and *Running Samples in the Predix SDK* for instructions on building and running the sample application.

## Configuring the Git Repository Management Service

Security permissions are set up by default to only allow JGit to create repositories in `<Predix Machine runtime container location>/appdata/GitRepositories`.

The helper features are available under REST if the configuration file property is set to expose this feature. This property is `com.ge.dspmicro.gitrepository.rest.enabled=true`. A simple HTML page can be returned for these options from the server: <http://localhost:8181/git.api/v1/commands>

The `IGitRepository` is a singleton service. All applications using the `IGitRepository` service share a single instance. It provides simple methods for accessing a default Git repository setup in the configuration file.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to and open the file, `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.gitrepository.config`.
2. Set values for the following properties to clone a remote repository on first access:

Property	Description	Default Value
<code>com.ge.dspmicro.gitrepository.clone</code>	Clone remote repository on first access. Set this to clone a remote repository on startup. After the clone, use the service commands to pull, push, or reset the local repository.  For example: <code>https://anonymous@openge.ge.com/git/gumf</code>	
<code>com.ge.dspmicro.gitrepository.clonePath</code>	The name of the folder to place the clone repository in the local <code>gitRepositories</code> folder. If nothing is specified, the path name of the clone URI is used; <code>default</code> is used if there is no path.	

Property	Description	Default Value
<code>com.ge.dspmicro.gitrepository.credentials</code>	<p>User name for authentication if needed when accessing the repository.</p> <p>Credentials can be passed in the URI also by "prepending" the address with</p> <pre>username:password@https:// username:password@openge.ge.com/git/project</pre> <p>Anonymous can be used for projects that allow read access to everyone. No "credentials" need to be passed in this case. <code>https://anonymous@openge.ge.com/git/project</code></p>	
<code>com.ge.dspmicro.gitrepository.credentials</code>	<p>Password for authentication in accessing the repository. The password gets encrypted.</p>	
<code>com.ge.dspmicro.gitrepository.rest.enabled</code>	<p>Should the default repository support REST interface boolean - <code>true</code> for enable, <code>false</code> for no REST support.</p> <p>When enabled, a sample HTML page containing the commands is returned:</p> <pre>https://{hostname}:{port}/git.api/v1/commands</pre>	B"false"

## Machine Gateway Services

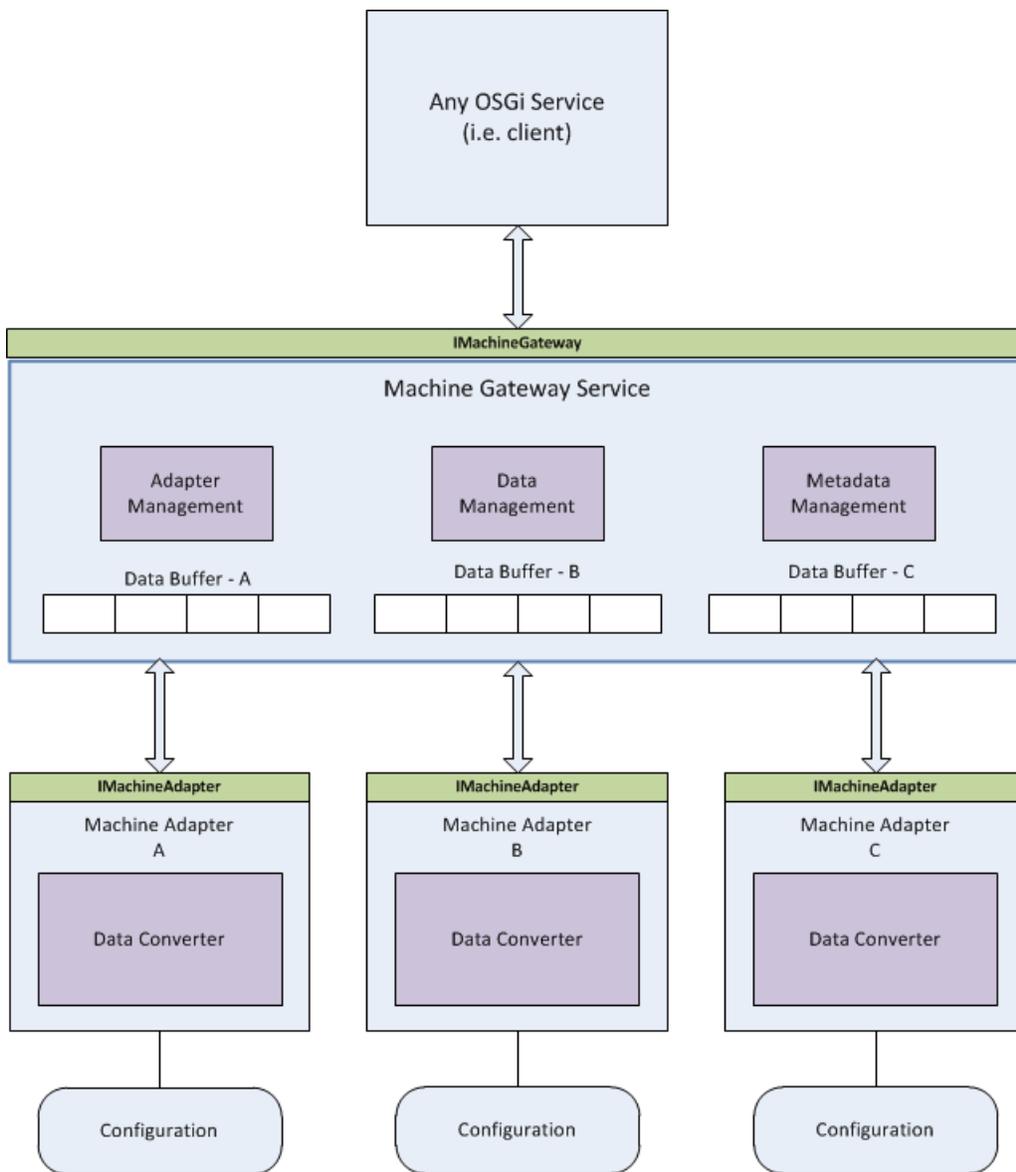
### Machine Gateway Service

The Machine Gateway service provides centralized configuration, monitoring, and management of Machine Adapters that run on the same OSGi framework instance.

It also exposes real-time data streams coming from multiple Machine Adapters through the common APIs and acts as a data aggregator to serve multiple clients.

The Machine Gateway service has a flexible and extensible architecture, which allows developers to implement a custom adapter for live data streaming. By default, the MQTT, OPC-UA, Modbus, and Health Monitor Machine adapters are provided.

At start up, the Machine Gateway queries the OSGi service registry for all installed Machine Adapters and keeps track of their status. The `IMachineGateway` interface allows client services to get a list of Machine Adapters and data nodes, and read and write data. The following diagram shows a high-level component view of the Machine Gateway service and how it interacts with other services



## Dependencies

The package `com.ge.dspmicro.machinegateway.types` and classes in the package have been moved to a new `device-common` bundle. Add the following dependency for those classes.

```
<groupId>com.ge.dspmicro</groupId>
<artifactId>device-common</artifactId>
<version>{Predix Machine version}</version>
```

## Using the Machine Gateway APIs

Review the Machine Gateway Javadoc APIs to understand how to implement the Machine Gateway service.

Navigate to and open the following API: `<SDK installation location>/docs/apidocs/index.html/com.dspmicro.machinegateway.api`.

## Using the Machine Gateway Sample Applications

Two sample applications are provided to show you how to build custom Machine Gateway adapters.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.
2. Open the `sample-basicmachineadapter` and `sample-sample-subscriptionmachineadapter` applications.

 **Note:** See [Sample Applications \(page 45\)](#) for instructions on building and running the sample application.

## Machine Gateway Ping

The Machine Gateway Ping service sends a PING message along the gateway from the Machine Gateway, Machine Adapter, Data Subscription, Hoover Spillway and Processor, to the river service to verify a connection.

The Machine Gateway Ping service implements the `IValidateRoute` interface. Before the `ping (IPingMessage)` method is called, an `IPingMessage` instance should be created from the Machine Gateway Ping service by calling the method `createPingMessage (Map<String, Object> params, IPongNotification pongNotification)`. Two arguments are required for the creation: three parameters with keys, adapter, subscription, and Spillway, defined in the `E2EPingMessage` class to identify the gateway, and a pong notification callback to handle the pong message along the gateway.

If one or more parameters are invalid or missing, the PONG message will contain the valid options for you to change the parameters and retry.

## Dependencies

Maven Dependencies and OSGi imports are required to consume the service:

- The following Maven dependencies are required to consume the Machine Gateway Ping service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>validateroute-api</artifactId>
  <version>{Predix Machine version}</version>
</dependency>
```

- The following OSGi imports are required in consuming the bundle

```
Import Package: com.ge.dspmicro.validateroute.api;version="[1.0,2)"
```

## Obtaining the Machine Gateway Ping Service

Inject the Machine Gateway Ping service using declarative services.

Example on how to inject the Machine Gateway Ping Service:

```
@Reference(type = '*')
public void setMachineGatewayPing(IValidateRoute validateRoute)
{
    if
    (E2EPingMessage.ROUTE_ADAPTER_TO_DATA_RIVER.equals(validateRoute.getRouteName()))
    {
        //set service here
        this.machineGatewayPing = validateRoute;
    }
}
```

## *Data Store and Forward*

The Data Store and Forward service continuously stores data generated by machines, sensors, and devices and forwarded to the cloud.

Predix Machine gathers data generated by machines, sensors, and devices and forwards them to the cloud. The Data Store and Forward service can continuously store that data to prevent data loss due to a system crash or disrupted network connection. The temporarily stored data will be forwarded when your network is back.

With the Data Store and Forward service, you can persist data in data stores and forward it to the destination. Each data store has a separate file in the <Predix Machine runtime container location>/appdata/storeforward/<storeName>. The data is stored and forwarded as a binary large object (BLOB). Serializing and deserializing the binary data depends on the contract between the data producer and the data consumer.

## Spillway Using Data Store and Forward

The Data Store and Forward service provides a way to temporarily store data, which is read by Machine Adapters, when a connection to the destination is not available. After the connection to the destination is recovered, persisted data is forwarded to the destination in the order in which it was received.

Data Store and Forward is a generic service. Each Spillway instance in Hoover can have a Data Store and Forward instance to store the data and forward it to a river. Review the following data flow for the Hoover service with a Spillway using the Data Store and Forward service.

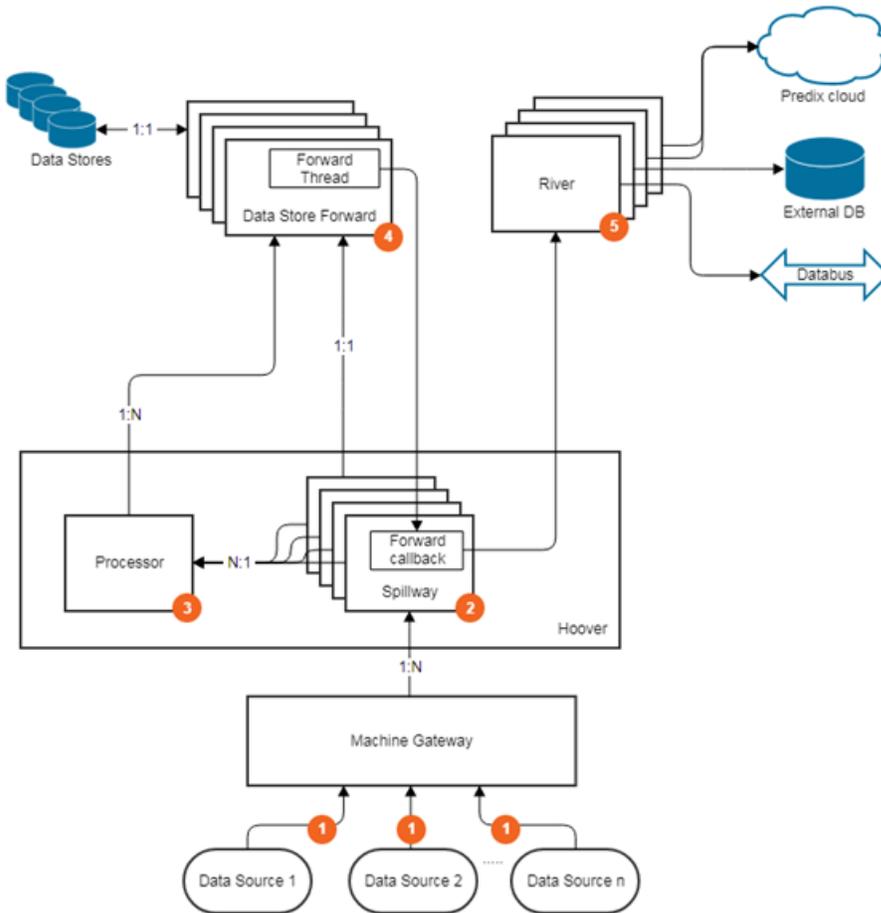
1. Each Spillway is associated with one Data Store and Forward service instance to back up the data.
2. A Data Store and Forward service instance can only be used by one Spillway. The Spillway adds its forward callback when the Data Store and Forward instance is injected.
3. The Spillway always persists data to the data store after the processor and before sending to a river.
4. The Forward Thread pulls any existing data from the database and calls the Forward Callback to forward the data to the river if available.
  - a. When the data is successfully sent out, the callback returns SUCCESSFUL, and the data is removed from the data store by the Forward Thread. The Forward Thread continues to forward data in the order in which it was received until the database is emptied.
  - b. If the river is down, the callback returns FAILED, and the data remains in the data store. The Forward Thread sleeps for the configured interval and tries again. Meanwhile, the service continues persisting new data to the database.
  - c. If the data is not valid, the callback returns FAILED\_BUT\_DELETE, an information message is logged, and the data is removed from the database.

Other scenarios with the Spillway and the Data Store and Forward service include the following.

- If a Spillway has no Data Store and Forward service configured, the data is forwarded directly to the river.
- If a Spillway has a Data Store and Forward service configured, but there is no Forward Callback registered in the Data Store and Forward service, an error is logged by the Forward Thread and the data is persisted to the database.
- If a Spillway has a Data Store and Forward service configured but the Data Store and Forward is not found, an error is logged and the data is forwarded to the river. In this case, if there is data in the data store, the order of the data is not guaranteed.

The data flow is illustrated in the following diagram.

Figure: Spillway using Data Store and Forward Service



1. Data is collected from sensors, machines, and devices.
2. Each spillway links the ingress channel to an optional Processor (3), an optional Store and Forward service (4), and an egress channel (river) (5).
3. A user-implemented processor allows custom-manipulation of ingress data before it is stored and/or forwarded to the destination. Only one processor is supported.
4. Data is persisted in the data stores by the Store and Forward service instances until the rivers are ready to send the data.
5. Data is forwarded to the destination by the rivers.

### Limitations

- Only one `ForwardCallback` instance is allowed per data store.
- Only one `Processor` per Predix Machine is supported. When multiple `Processors` exist, the OSGi framework may not load any of them.

## Dependencies

Maven dependencies and an OSGi import package are required to consume the service:

- The following Maven dependency is required:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>storeforward-api</artifactId>
  <version>${dspmicro.storeforward.project.version}</version>
</dependency>
```

- The following OSGi import is required to consume the bundle:

```
Import-Package: com.ge.dspmicro.storeforward.api;version="[1.0,2)"
```

## Obtaining the Data Store and Forward Service

Inject the `IStoreForward` service using declarative services.

Example of how to inject the `IStoreForward` service:

```
import com.ge.dspmicro.storeforward.api;

...

@Reference(type = '*')
public void addStoreForward(IStoreForward storeForward)
{
    //get instance whose name matches;
}
```

## Configuring the Data Store and Forward Service

You can customize the configuration of your Data Store and Forward service.

To do so, add a unique name and description, and specify the intervals at which the callback reads and forwards data.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
------------	-------------	----------

F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is property=<type>["<value1>",<value2>"]. For example, array of integer property=I["1", "2", "3"]. A backslash may be used to break up the line for clarity.

1. Navigate to and open the following file: <Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.storeforward-[n].config.
2. Set values for the following properties:

Property	Description	Type	Default Value	Required
com.ge.dspmicro.storeforward.name	Unique name of the service.	String	"DefaultStoreForward"	Yes
com.ge.dspmicro.storeforward.description	Description of the Store and Forward service	String	"Default store forward"	No
com.ge.dspmicro.storeforward.persistenceType	The underlying data store used to persist data. Valid values are: <ul style="list-style-type: none"> <li>• <b>TAPE</b> – Starting with Predix Machine 17.2, the Tape implementation is used by default. If you are using the legacy H2 database implementation, it is recommended that you switch to the Tape implementation. For information about Tape, see <a href="https://github.com/square/tape">https://github.com/square/tape</a>.</li> <li>• <b>H2_FILE</b> – Starting with Predix Machine 17.2, the H2 database implementation is being deprecated, and it is recommended that you switch to the Tape implementation.</li> </ul>	Enum	TAPE	No
com.ge.dspmicro.storeforward.readInterval	The number of seconds between which it reads and forwards the data if the callback returns FAILED. If there is data in the database and the callback returns SUCCESSFUL, the forward thread will continue to read and forward the data until the database is empty.	Integer	"10"	No

Property	Description	Type	Default Value	Required
com.ge.dspmicro.storeforward	<p>Specifies whether to enable or disable database compression. The database compression is disabled when this property is set to <code>false</code>. You can also remove the property from the configuration file to disable database compression.</p> <p>If this property does not exist in the configuration file (because you are using an earlier version of Predix Machine), this defaults to <code>false</code>.</p>	Boolean	true	No
com.ge.dspmicro.storeforward	<p>Auto-generated database password. Users should not change the password. Once the password is changed, the existing database cannot be opened and the data in the database will be lost. To recover from the wrong password to open the database, the database file in the <code>appdata/storeforward</code> folder must be deleted so that a new database can be created.</p>	String		Yes
com.ge.dspmicro.storeforward	<p>Specifies whether to enable or disable database encryption. The database file encryption can be disabled by setting this property to <code>false</code> or by removing this property from the configuration file. When the encryption is flipped between <code>on</code> and <code>off</code>, the old database if any will not be opened and should be removed.</p> <p>If this property does not exist in the configuration file (because you are using an earlier version of Predix Machine), this defaults to <code>false</code>.</p>	Boolean	true	No
com.ge.dspmicro.storeforward	<p>By default, the database file encryption password is the same as the generated database password and is encrypted in the configuration file. The database file encryption can be disabled by removing this property from the configuration file. When the encryption is flipped between <code>on</code> and <code>off</code>, the old database, if any, will not be opened and should be removed.</p>	String	same as databasePassword	No

Property	Description	Type	Default Value	Required
com.ge.dspmicro.store	<p><b>Database file size limit</b> at which the Data Store and Forward service stops writing data to the file. The default value is <code>-1</code>, which indicates unlimited file size.</p> <p> <b>Important:</b> GE strongly recommends that this parameter be set to a reasonable value based on the storage space available on the device, so the device does not run out of space in case connectivity is lost for extended periods of time.</p> <p> <b>Note:</b></p> <p>This note applies only to Predix Machine versions 17.2, 17.2.1, and 17.2.2. This limitation does not apply to Predix Machine version 17.2.3.</p> <p>The Store and Forward service may not stop writing precisely at the specified limit. Tape data stores are created with 4KB initially and its size doubles every time it runs out of space. Store and Forward allows the doubling as long as the size is below the specified limit. For example, if the <code>dbSizeLimitMB</code> is set to 3MB, and the current data store size is at 2MB, expansion will happen, pushing its size to 4MB. The next time it runs out of space, expansion will not happen because the size is already above the set <code>dbSizeLimitMB</code>.</p> <p> <b>Note:</b></p> <p>This note applies only to Predix Machine versions 17.2, 17.2.1, and 17.2.2. This limitation does not apply to Predix Machine version 17.2.3.</p> <p>Once the Tape data store reaches or exceeds the set <code>dbSizeLimitMB</code>, the Store and Forward service stops writing to the store and does not resume until all the data in the store is forwarded. This behavior is different from the previous H2 implementation, which resumed writes as soon as the store size dropped below the <code>dbWriteResumeLimitMB</code>.</p>	Integer	"-1"	No

Property	Description	Type	Default Value	Required
<p> <b>Warning:</b> Do not change the parameters below this warning unless instructed to do so by GE Customer Support.</p>				
com.ge.dspmicro.store	<p><b>Maximum number of attempts</b> to dispatch messages before messages are discarded. Default is unlimited.</p>	Integer	!"-1"	No
com.dspmicro.store	<p><b>forward.dbWriteResumeLimitMB</b></p> <p> <b>Note:</b> Applies only to the H2 database implementation.</p> <p>Database size limit at which writing to the database is resumed after the file size exceeds the dbSizeLimitMB value, causing database writing to be suspended. Writing resumes only after the file size falls below the dbWriteResumeLimitMB value. When defined, this value must be less than or equal to the dbSizeLimitMB value. The default value matches the dbSizeLimitMB value.</p>	Integer	!"<x>"  The value will match the value specified for the dbSizeLimitMB.	No
com.ge.dspmicro.store	<p><b>forward.dbSizeLimitMBCompact</b></p> <p> <b>Note:</b> Applies only to the H2 database implementation.</p> <p>Database file size limit at which the Data Store and Forward service attempts to compact the database file.</p>	Integer	!"1"	No
com.ge.dspmicro.store	<p><b>forward.compactIntervalSec</b></p> <p> <b>Note:</b> Applies only to the H2 database implementation.</p> <p>Defines how frequently the Data Store and Forward service checks the file size and compacts the database, if necessary. The default value is 60 seconds.</p>	Integer	!"60"	No
com.ge.dspmicro.store	<p><b>forward.purgeOnFullPercentage</b></p> <p> <b>Note:</b> Applies only to the H2 database implementation.</p> <p>Percentage of oldest messages to be discarded on every compactIntervalSec cycle when the database is full.</p>	Integer	!"0"	No

## Using the Data Store and Forward APIs

Review the Data Store and Forward Javadoc API to understand how to fully implement the Data Store and Forward service.

1. Navigate to and extract all files in the following file: `<SDK installation location>/docs/apidocs.zip`
2. Navigate to the Data Store and Forward Javadoc APIs at `<SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.storeforward.api`.

## Using the Data Store and Forward Sample Application

A sample Data Store and Forward application is provided to illustrate how to use the service.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.
2. Open the `sample-storeforwardclient` application.

 **Note:** See [Sample Applications \(page 45\)](#) for instructions on building and running the sample application.

## Managing Data Store and Forward Instances

You can manage Data Store and Forward instances in the Predix Machine Web Console.

The **Store and Forward** page lists all active and inactive data stores present in the machine's file system.

- Active data stores are associated with a running `StoreForward` service instance.
- Inactive data stores are not associated with a running `StoreForward` service instance.

You can perform the following tasks in the Predix Machine Web Console:

- [Deleting the Oldest Item in an Active Data Store \(page 130\)](#)
- [Deleting All Items in an Active Data Store \(page 131\)](#)
- [Deleting Inactive Data Stores \(page 131\)](#)

## Deleting the Oldest Item in a Data Store

Start your Predix Machine DEBUG container and open the Predix Machine Web Console.

You can delete the oldest item in an Active Data Store.

1. In the Predix Machine Web Console, on the **Predix** menu, click **Store and Forward**.
2. In the **Active Data Stores** section, click a data store.  
The last 10 data items in the data store appear in the **Store Contents** section.
3. Click **View Oldest Item** to review details about the oldest item in the data store.  
Only the first KB of data is displayed; if the data cannot be read, it is not displayed.
4. Click **Delete Oldest Item**.

### Deleting All Items in an Active Data Store

If the data in an Active Data Store is too old, or to stop sending its data, you can delete the store's contents.

Start your Predix Machine DEBUG container and open the Predix Machine Web Console.

1. In the Predix Machine Web Console, on the **Predix** menu, click **Store and Forward**.
2. In the **Active Data Stores** section, click a data store.  
The oldest 10 data items in the data store appear.
3. Click **Delete All Items** to delete all of the data items in the data store.

### Deleting Inactive Data Stores

Deleting an inactive store removes the store permanently.

Start your Predix Machine DEBUG container and open the Predix Machine Web Console.

1. In the Predix Machine Web Console, on the Predix menu, click **Store and Forward**.
2. In the **Inactive Data Stores** section, click the name of the inactive data store to delete and click **Delete Store**.  
A **Confirm delete** confirmation appears.
3. Click **Delete** to delete the inactive data store.

## *Data Store and Forward Ping*

The Data Store and Forward Ping service enables you to send a PING message to the data store, persist it to the data store, and forward it to the forward callback to validate the Data Store and Forward route.

The `StoreForward` service implements the `IValidateRoute` interface. Before the `Ping` (`IPingMessage`) method is called, an `IPingMessage` instance should be created from the Data Store Forward Ping service by calling the `createPingMessage(Map<String, Object> params,`

`IPongNotification pongNotification)` method. No parameter is required for the Data Store and Forward route. A Pong notification callback is required to handle the PONG message along the route.

## Dependencies

Maven dependencies and OSGi imports are required to consume the service:

- The following Maven dependencies are required to consume the Data Store and Forward Ping service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>validateroute-api</artifactId>
  <version>{Predix Machine version}</version>
</dependency>
```

- The following OSGi imports are required in the consuming bundle:

```
Import Package: com.ge.dspmicro.validateroute.api;version="[1.0,2)"
```

## Obtaining the Data Store and Forward Ping Service

You can obtain the Data Store and Forward Ping service using either of the following ways.

- Obtain the instance of the Data Store and Forward Ping service and then call the `StoreForward` interface method `getValidateRoute()`.
- Inject the Data Store and Forward Ping service using declarative services. By default, the name of the route of the specific `StoreForward` instance is the combination of the store name and the `Route` string.

The following code sample shows how to inject the Data Store and Forward Ping service:

```
Example on how to inject the Store Forward Ping Service:
@Reference(type = '*')
public void setStoreForwardPing(IValidateRoute validateRoute)
{
  if (YourStoreForwardRouteName.equals(validateRoute.getRouteName())
  {
    //set service here
    this.storeForwardPing = validateRoute;
  }
}
```

## Gateway Validation

Use Gateway Validation service to validate gateway connectivity for any components that require active gateway connections from one point to another, (for example, machine-to-machine or machine-to-cloud.)

### Functionality

The Gateway Validation service provides the ability to validate gateway connections between components that require gateway connectivity and validation. It implements the `IValidateRoute` interface. All River services can use the Gateway Validation service.

### Example Use Case

In the Data River, the Data River Send and Data River Receive services require gateway connection and validation to ensure communication between the two. The Data River service can use the Gateway Validation service to validate the Data River Send and Data River Receive gateway connections.

Data River Ping services sends a PING message from a Data River Send service to a Data River Receive service to verify a connection. Two Data River Ping services are used in the ping process: one for the Data Send service and one for the Data Receive services.

The Data River Ping service implements the `IValidateRoute` interface. When the `ping` method is called, the Data River Ping service that is implemented in the Data River Send service, sends a request message to a paired `Data Receive` service.

To send a PING, the `IPingMessage` interface must be implemented and passed to the Data River Ping service. The service holds any Data River Send services available; you can specify the target river by adding a Key/Value pair in the map returned from the `IPingMessage` interface implementation. The Data River key can be found in the `IPingConstants` interface in the common bundle, and the value in the map is the target data river to PING.

Once a PING is sent, the service waits as specified by the `timeout` value that is specified in the Data River Send configuration; if no response is received, or if a successful PING response is received, the callback implemented in the `IPingMessage` is notified with a `PongMessage`.

### Dependencies

Before you begin using the Gateway Validation service, you must ensure that the following dependencies are satisfied:

- Maven Dependencies

The following maven dependencies are required to consume the Gateway Validation service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>validateroute-api</artifactId>
  <version>{Predix Machine version}</version>
</dependency>
```

- OSGi Manifest.MF Imports

The following OSGi imports are required in the consuming bundle:

```
Import-Package: com.ge.dspmicro.validateroute.api;version="[1.0,2)"
```

## Creating Gateway Validation Ping Requests

You can use the Gateway Validation service in the Predix Machine Web Console. The Gateway Validation service provides the ability to validate routes using a PING.

 **Note:** The generated Predix Machine runtime container must be the Predix Machine Debug version, or you must manually add the Web Console feature using the Predix SDK.

1. Start Predix Machine.

- a. From a command line, navigate to the `<Predix Machine runtime container location>/bin` directory.
- b. Use one of the following to start Predix Machine:  
`start_predixmachine.sh` (Linux and Mac)

2. Start the Predix Machine Web Console.

- a. Open a web browser and enter the following URL:

```
https://localhost:8443/system/console/
```

A login prompt appears.

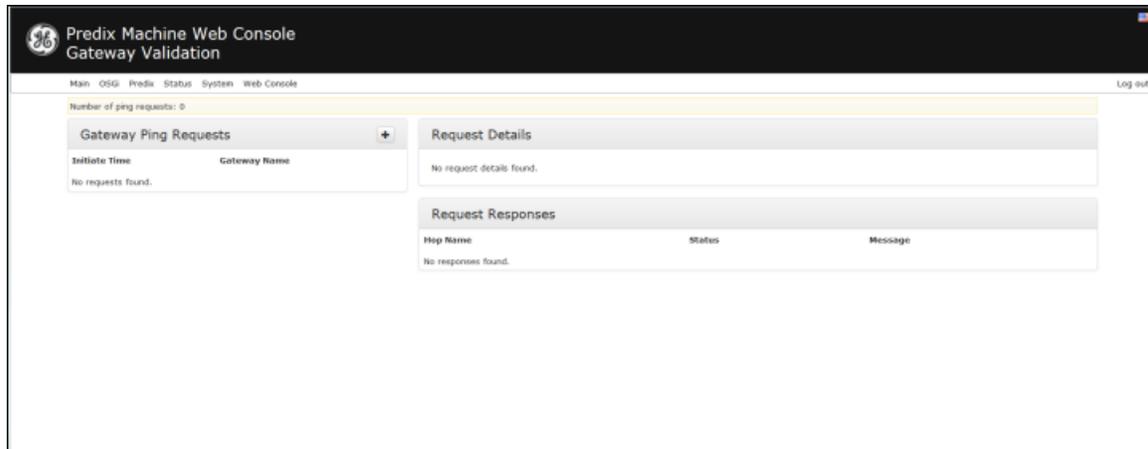
- b. Enter your user name and password. Predix Machine uses the following defaults:

**Username:** predix

**Password:** predix2machine

 **Note:** On first login, you must change the password.

3. On the **Predix** menu, click **Gateway Validation**.



4. Click the + icon to create a PING request.
5. Click the **Ping** button when you are finished setting up the PING request.  
You will receive either a **SUCCESSFUL** or **FAILED** Status message. Review the contents of any **FAILED** message to troubleshoot any problems associated with your PING.
6. To delete a PING request, click the  icon in the **Gateway Ping Requests** section.
7. To repeat a PING, click the  icon.

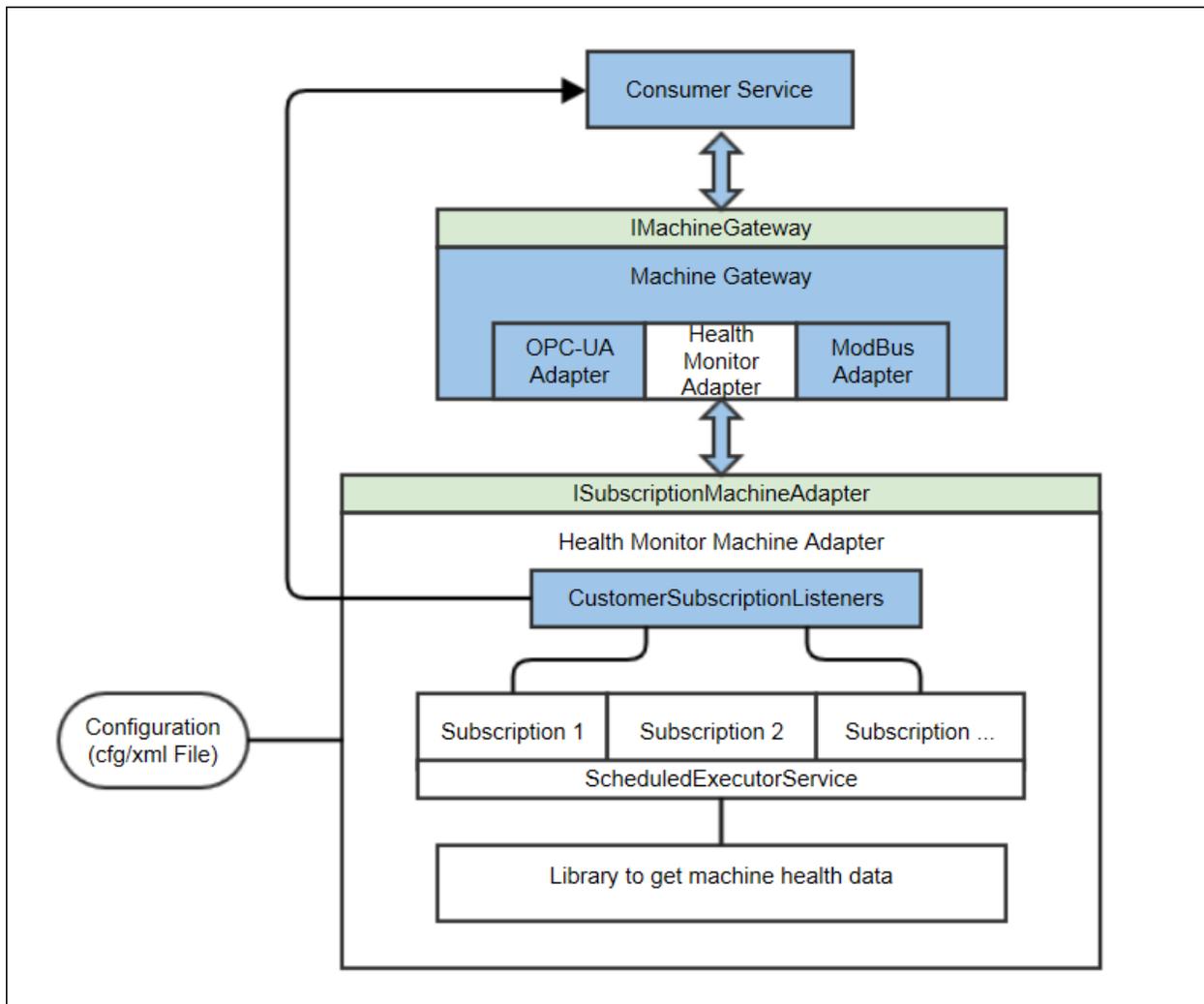
## *Health Monitor Machine Adapter*

The Health Monitor Machine Adapter is a data-acquisition adapter that runs in the Predix Machine runtime container as an OSGI bundle and monitors OS-level health metrics on the machine.

It implements the `ISubscriptionMachineAdapter` interface and is managed by the Machine Gateway service.

The following diagram illustrates the overall Machine Gateway and adapter architecture that includes the Health Monitor Machine Adapter:

Figure: Health Monitor Machine Adapter Architecture



## Consume the Health Monitor Machine Adapter

The Machine Gateway service consumes the Health Monitor Machine Adapter like other adapters. You can search for the adapter in the returned list of adapters from Machine Gateway using the `AdapterType` property in the `MachineAdapterInfo` class.

 **Note:** See [Machine Gateway \(page 119\)](#) for how to obtain adapters.

Use the `ISubscriptionMachineAdapter` interface to interact with the Health Monitor Machine Adapter.

## Limitations

The Health Monitor Machine Adapter returns data for the following health categories:

- CPU usage.

- Memory usage.
- Disk usage (by file system).
- ProSys Predix Machine JVM memory usage.

## Configuring the Health Monitor Machine Adapter

You can configure your implementation of the Health Monitor Machine Adapter.

You must specify the path to the Health Monitor Machine Adapter XML file and provide a name and description of the adapter.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to and open the following file: `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.healthmonitor.config`.
2. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.machineadapter.healthmonitor.configFile</code>	The path to the Health Monitor Machine Adapter XML file.  If this property value is blank, the adapter is disabled.	String	"<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.healthmonitor.xml"	No, but if this property value is blank, the adapter is disabled.
<code>com.ge.dspmicro.machineadapter.healthmonitor.name</code>	Human-readable name of the adapter.	String	"healthmonitor"	?
<code>com.ge.dspmicro.machineadapter.healthmonitor.description</code>	Human-readable description of the adapter.	String	"Health\ Monitor"	?

The Health Monitor Machine Adapter XML configuration file describes the adapter connection nodes and subscriptions. A sample file is provided with the container. Its read nodes and subscriptions are statically configured.

The following example shows a Health Monitor Machine Adapter XML configuration file. The `nodeTypes` are the only types currently supported, you should not change these values. You can change the `updateInterval` values.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<healthMonitorMachineAdapterConfig>
  <name>Machine health monitor nodes</name>
  <description>Machine health monitor nodes</description>
  <dataNodeConfigs>
    <dataNode name="Node-1-1" nodeType="SYSTEM_CPU_USAGE"
    dataType="DOUBLE" description="CPU Usage" />
    <dataNode name="Node-2-1" nodeType="SYSTEM_MEMORY_USAGE"
    dataType="DOUBLE" description="Memory Usage" />
    <!-- If only one file system is available, you do not have to
    specified drive attribute. By default, the first one is returned. -->
    <dataNode name="Node-3-1" nodeType="SYSTEM_DISK_USAGE"
    dataType="DOUBLE" description="Disk Usage Example" />
    <!--
    For Linux, drive name is constructed as
    <mounted name><space><(Filesystem name)>
    -->
    <!-- The followings are examples on how to specify drive
    attribute
    <dataNode name="Node-3-2" nodeType="SYSTEM_DISK_USAGE"
    dataType="DOUBLE" drive="/" (/dev/mapper/ubuntu--vg-root)"
    description="Linux Disk_Usage Example" />
    <dataNode name="Node-3-3" nodeType="SYSTEM_DISK_USAGE"
    dataType="DOUBLE" drive="/mnt (192.168.30.66:/export)" description="Linux
    Disk_Usage Example" />
    -->
    <dataNode name="Node-4-1" nodeType="JVM_MEMORY_USAGE"
    dataType="DOUBLE" description="JVM Memory Usage" />
  </dataNodeConfigs>
  <!-- updateInterval is in seconds. -->
  <dataSubscriptionConfigs>
    <dataSubscriptionConfig name="CPU Memory Usage" updateInterval="60"
    >
      <nodeName>Node-1-1</nodeName>
      <nodeName>Node-2-1</nodeName>
      <nodeName>Node-4-1</nodeName>
    </dataSubscriptionConfig>
    <dataSubscriptionConfig name="Disk Usage" updateInterval="1200" >
      <nodeName>Node-3-1</nodeName>
    </dataSubscriptionConfig>
  </dataSubscriptionConfigs>

```

```
</healthMonitorMachineAdapterConfig>
```

## Using the Health Machine Adapter Sample Application

A sample Health Machine Adapter Sample Application is provided to illustrate how to use the service.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.
2. Navigate to and open the `<SDK installation location>/samples/sample-apps/sample/sample-healthmachineadapter` application.

 **Note:** See [Sample Applications \(page 45\)](#) for instructions on building and running the sample application.

## *Hoover Service*

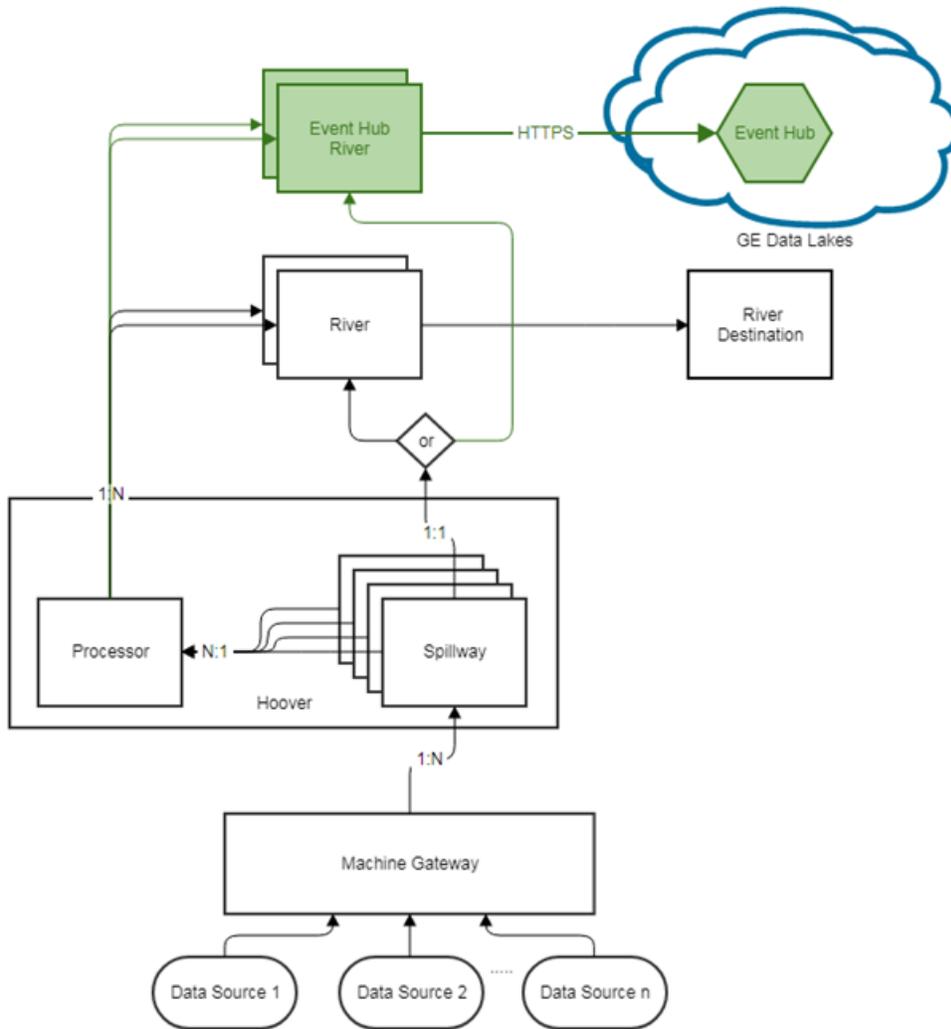
The Hoover bundle manages multiple Spillways to collect data from the Machine Gateway, and then uses a Processor to process the data and send it to the destination through a river, or to the Predix cloud through the Event Hub River.

A Hoover Spillway collects data from the Machine Gateway, and processes it to determine if the data goes through a river, or an Event Hub River. If the Hoover Spillway configuration specifies an Event Hub River or WebSocket River destination, it sends it through an Event Hub River or WebSocket River to the HTTP Data service or Time Series database in the Predix cloud.

The Processor is a routine to process the data before sending it out. Such processing can include filtering, changing data format, and converting data.

Data can be collected from different sensors through different `SubscriptionMachineAdapter` types. The data type and format may be different.

The following diagram illustrates how the Hoover bundle processes and sends data to the Event Hub River.



### Using Hoover APIs

Review the Hoover Processor and Hoover Spillway Javadoc APIs to understand how to implement the Hoover service.

1. Navigate to and extract all files in the following file: `<SDK installation location>/docs/apidocs.zip`
2. Navigate to the Process API at `<SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.hoover.api.processor` and `<installation location>/docs/apidocs/index.html/com.ge.dspmicro.hoover.api.spillway`.
3. Navigate to the Spillway API at `<SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.hoover.api.spillway`.

## Configuring the Hoover Service

You can customize the configuration of your Hoover service by specifying the name and description of the Spillway, listing the data subscribers, and the name of the destination river.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to and open the configuration file: `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.hoover.spillway-{n}.config`
2. Set values in the following properties:

Property	Description	Type	Default Value
<code>com.ge.dspmicro.hoover.spillway.name</code>	A unique name of the spillway.	String	DefaultSpillway
<code>com.ge.dspmicro.hoover.spillway.description</code>	(Optional) A brief description of the spillway	String	Simple test spillway
<code>com.ge.dspmicro.hoover.spillway.dataSubscriptions</code>	Pipe (" ")-separated list of DataSubscription names.	List of strings	<ul style="list-style-type: none"> <li>• Temperature_Subscription</li> <li>• Pressure_Subscription</li> <li>• OPCUA_Subscription_1</li> <li>• OPCA_Subscription_2</li> </ul>
<code>com.ge.dspmicro.hoover.spillway.dataSubscriptionsArray</code>	An array of data subscriptions where the data will come from.	List of strings	sampleEdgeTopic
<code>com.ge.dspmicro.hoover.spillway.destination</code>	The name of the river.	String	Sender Service
<code>com.ge.dspmicro.hoover.spillway.processType</code>	(Optional) Type of business logic defined in the Processor to process the data.	Strings	

Property	Description	Type	Default Value
<code>com.ge.dspmicro.hoover.spillway.storeforward</code>	Name of StoreForward associated with this spillway.	String	DefaultStoreForward
<code>com.ge.dspmicro.hoover.spillway.timeout</code>	Sets the length of time, in seconds, the Spillway waits for a response from Data River.	Integer	1"10"

## Using the Hoover Sample Application

A sample Hoover application is provided to illustrate how to use the service.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.
2. In the `sample-apps/sample` folder, open the `sample-hoover` application.

 **Note:** See [Sample Applications \(page 45\)](#) for instructions on building and running the samples.

## Modbus Machine Adapter

The Modbus Machine Adapter is a data-acquisition component based on the Modbus standard.

It discovers and connects to Modbus nodes, browses the nodes' address spaces, and reads/writes data using different data access mechanisms.

 **Tip:** For more information on Modbus, see <http://www.modbus.org/>

The Modbus Machine Adapter runs on top of the Predix Machine runtime container as an OSGi service and supports the `ISubscriptionMachineAdapter` interface. The Modbus Machine Adapter is managed by the Machine Gateway service. It can be consumed through the Machine Gateway service like all other adapters. It can be found in the returned list of adapters from Machine Gateway using the `Adapter Type` property in the `MachineAdapterInfo` class.

Modbus standard protocols support both TCP/IP and Serial communication, but for the Modbus Machine Adapter in Predix Machine to support Serial communication, you must deploy RXTX with its embedded native libraries to the Predix Machine runtime container.\*

\*Some platforms are not supported by RXTX libraries. The following table points to locations of library builds of third-party software that works with certain platforms:

Platform	Download Link	Notes
Raspberry Pi II (ARM7 Linux)	<a href="https://blogs.oracle.com/jtc/resource/rxtx-arm7/librxtxSerial-2.1-7.so">https://blogs.oracle.com/jtc/resource/rxtx-arm7/librxtxSerial-2.1-7.so</a>	This link only gives native libraries. The RXTXComm.jar is OK.  The library should be renamed (or softlinked to by) librxtxSerial.so

1. Download the RXTX library ZIP file from <http://rxtx.qbang.org/pub/rxtx/rxtx-2.2pre2-bins.zip>.
2. Create an rxtx directory in the following location: <Predix Machine runtime container location>machine/lib
3. Unzip the file to the <Predix Machine runtime container location>machine/lib/rxtx folder.

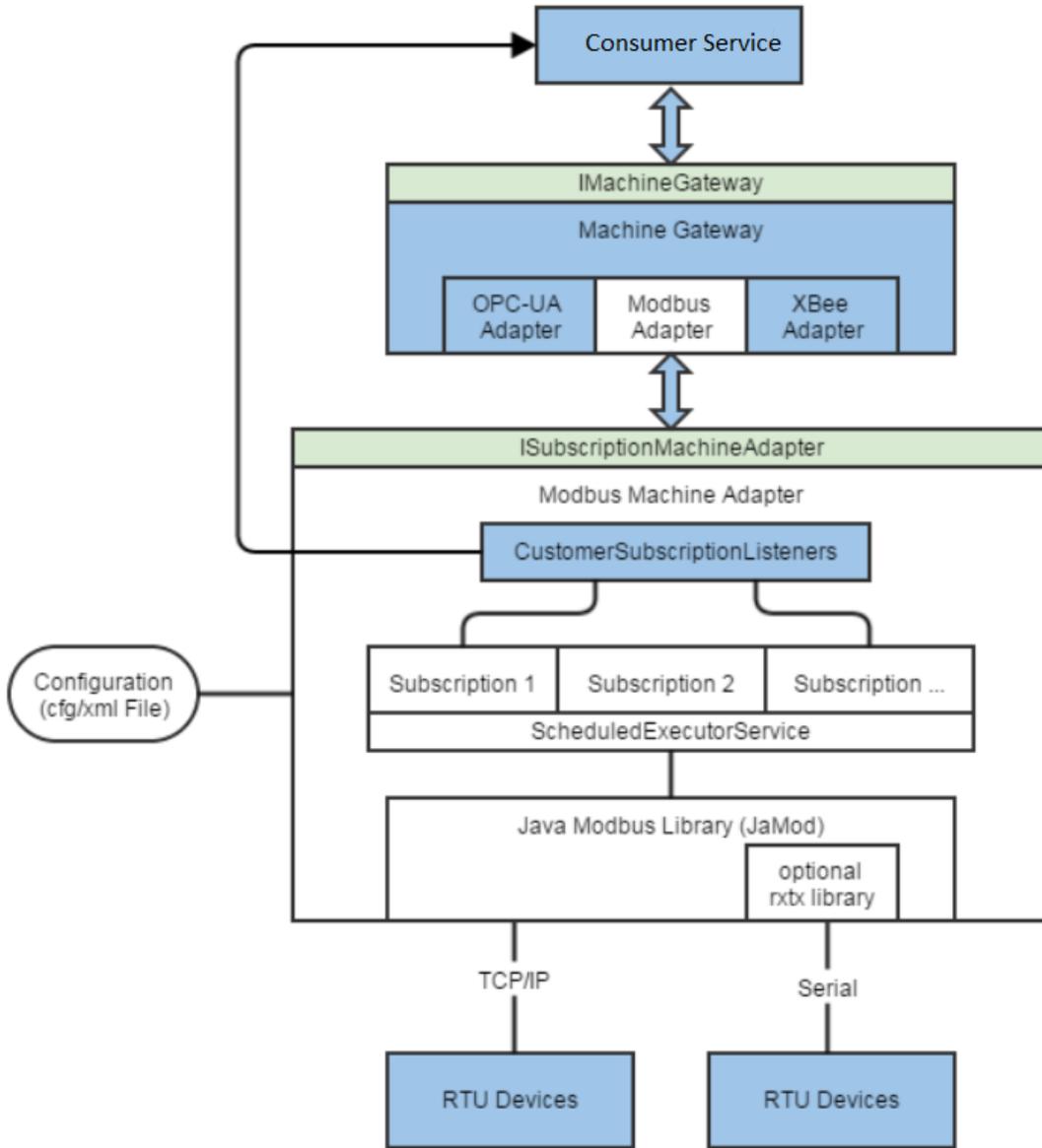
An rxtx-2.2pre2-bins directory is created within the rxtx directory. This should result in a directory structure like this:

```
Predix Machine container location
machine
  lib
    rxtx
      rxtx-2.2pre2-bins
        RXTXcomm.jar
        x86_64-unknown-linux-gnu
        mac-10.5
        win64
        win32
```

 **Note:** You only need to include the directories for the platforms you want to support. For example, if you want to run on Linux, you do not need to include the mac-10.5, win64, or win32 directories. You must always include the RXTXcomm.jar file.

The following diagram illustrates how the Modbus Machine Adapter fits within the overall architecture of the Machine Gateway.

Figure: Modbus Machine Adapter Architecture



## Modbus Machine Adapter Functionality

The Modbus Machine Adapter supports the following functions:

- Loading the Modbus Machine Adapter XML configuration file to read the following information:
  - Connection information
  - Data nodes
  - Subscription nodes
  - Publishing intervals

- Registering with the Machine Gateway:
  - Exposing the configuration.
  - Exposing the adapter state.
- Connecting to Modbus nodes using TCP and Serial transport.
- Data Reading/Writing Operations.
- Subscribing to Register Value Attributes.

## Limitations

Due to the nature of the Modbus protocol, there are no security features. Solutions should not use Modbus unless absolutely necessary.

- Note that data is unencrypted and no role-based access control or authentication mechanisms are in place.
- Make sure that no external access to the Modbus server is allowed.
- Make sure that the environment is isolated from external network access.

## Using the Modbus Machine Adapter APIs

Review the Modbus Machine Adapter APIs to understand how to implement the Modbus Machine Adapter.

1. Navigate to and extract all files in the following file: `<SDK installation location>/docs/apidocs.zip`
2. Navigate to the Modbus Javadoc APIs at `<SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.machineadapter.modbus.api`.

## Configuring the Modbus Machine Adapter

You can configure both the connection node and subscriptions in the XML configuration file.

You can configure the name, description, and location of the XML configuration in a configuration (`.config`) file per instance.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is property=`<type>["<value1>","<value2>"]`. For example, array of integer property=`I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. To configure an instance of the Modbus Machine Adapter connection nodes and subscriptions, edit the XML configuration file: <Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.modbus-[n].xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<modbusMachineAdapterConfig>
  <name>Onsite monitor modbus nodes</name>
  <description>Onsite monitor modbus nodes</description>
  <dataNodeConfigs>
<!-- REMOVE THIS LINE FOR TCP/IP
    <channel protocol="TCP_IP" tcpIpAddress="127.0.0.1"
tcpIpPort="502">
      <unit id="1">
        <register name="Node-1-1" dataType="INTEGER"
address="10" registerType="HOLDING" description="temperature" />
        <register name="Node-1-2" dataType="DECIMAL"
address="11" registerType="HOLDING" description="pressure" />
      </unit>
      <unit id="2">
        <register name="Node-2-1" dataType="INTEGER"
address="20" registerType="HOLDING" description="temperature" />
        <register name="Node-2-2" dataType="INTEGER"
address="21" registerType="HOLDING" description="pressure" />
      </unit>
    </channel>
  REMOVE THIS LINE -->
<!-- REMOVE THIS LINE FOR SERIAL
    <channel protocol="SERIAL" encoding="RTU" portName="COM1"
baudRate="9600" parity="NONE">
      <unit id="3">
        <register name="Node-3-1" dataType="INTEGER"
address="30" registerType="HOLDING" description="temperature" />
        <register name="Node-3-2" dataType="INTEGER"
address="31" registerType="HOLDING" description="pressure" />
      </unit>
      <unit id="4">
        <register name="Node-4-1" dataType="INTEGER"
address="40" registerType="INPUT" description="temperature" />
        <register name="Node-4-2" dataType="INTEGER"
address="41" registerType="INPUT" description="pressure" />
      </unit>
    </channel>
  REMOVE THIS LINE -->
  </dataNodeConfigs>
  <!-- Both updateInterval and startPointOffset are in seconds. If
startPointOffset == -1, it will start immediately -->
  <dataSubscriptionConfigs>
<!-- REMOVE THIS LINE FOR TCP/IP
```

```

        <dataSubscriptionConfig name="Temperature_Subscription"
updateInterval="60" startPointUnit="MINUTES" startPointOffset="10">
        <nodeName>Node-2-1</nodeName>
        <nodeName>Node-1-1</nodeName>
        </dataSubscriptionConfig>
REMOVE THIS LINE -->
<!-- REMOVE THIS LINE FOR SERIAL
        <dataSubscriptionConfig name="Pressure_Subscription"
updateInterval="3600" startPointUnit="HOURS" startPointOffset="600">
        <nodeName>Node-3-2</nodeName>
        <nodeName>Node-4-2</nodeName>
        </dataSubscriptionConfig>
REMOVE THIS LINE -->
    </dataSubscriptionConfigs>
    </modbusMachineAdapterConfig>

```

2. To configure the name, description, and location of the XML configuration file for each instance of the Modbus Machine Adapter, edit the properties in the <Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.modbus-[n].config file.

Property	Description	Type	Default Value	Required
com.ge.dspmicro.machineadapter.modbus.[n].description	A human-readable description of the adapter.	String	"Supports basic read/write capability from Modbus nodes. Supports subscription to a group of Modbus nodes."	No
com.ge.dspmicro.machineadapter.modbus.[n].name	The name of the adapter.	String	"Modbus Machine Adapter"	Yes
com.ge.dspmicro.machineadapter.modbus.nullRetentionBehavior	When set to <code>false</code> , an exception is thrown on null data.  If set to <code>true</code> , null datapoints are retained and the quality is set to "BAD."	Boolean	false	No

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.machineadapter.modbus.config.maxRetries</code>	<p><b>Maximum number</b> of tries to read/write to devices before an exception is thrown. Minimum value is 2 so that the cached connections can be recycled if they fail.</p> <p>If, after the retries the device is not reachable, the cached data previously retrieved is returned with BAD quality.</p> <p>If there is no cached data, an exception will be logged and value 0 with BAD quality will be returned.</p>	Integer	"2"	No
<code>com.ge.dspmicro.machineadapter.modbus.config.configurationFile</code>	<p><b>The path to the</b> Modbus XML configuration file.</p> <p> <b>Important:</b> If you leave this property blank, the adapter will be disabled.</p>	String	"configuration/machine/com.ge.dspmicro.machineadapter.modbus-0.xml"	Yes
<code>com.ge.dspmicro.machineadapter.modbus.config.threadPoolSize</code>	<p><b>Thread pool size to</b> handle subscriptions to retrieve values from nodes. Minimum value is 1.</p>	Integer	"5"	No
<code>com.ge.dspmicro.machineadapter.modbus.config.registerLimit</code>	<p><b>Sets the maximum</b> number of Modbus registers that can be read in a single request.</p>	Integer	"125"	No

## Common Channel Configuration

The following attributes are commonly used to define how a device is accessed:

<b>Common Channel Attribute</b>	<b>Default</b>	<b>Valid Values</b>	<b>Required</b>	<b>Comments</b>
protocol		TCP_IP, SERIAL	Yes	<p><b>Tcp/Ip vs Serial Communication Protocol</b></p> <p>Specifies which communication protocol will be used to connect to the Modbus device.</p>
regBaseAddress	0	0,1	No	<p><b>Zero vs. One Based Addressing</b></p> <p>Specifies the register address numbering convention in the configuration file for the device. Starts at zero or one. If it is one-based addressing, user-entered addresses will have one subtracted when frames are constructed to communicate with a Modbus device.</p>
bitBaseAddress	0	0,1	No	<p><b>Zero vs. One Based Bit Addressing within Registers</b></p> <p>Memory types that allow bits within register (16 bits) can be referenced as a Boolean. Bit Addressing within registers provides two ways of addressing a bit within a given register: Zero Based and One Based. Zero Based Bit addressing within registers means that the first bit begins at 0. One Based Bit Addressing within registers means that the first bit begins at 1.</p>
defaultModbusByteOrder	true	true, false	No	<p><b>Default Modbus Byte Order</b></p> <p>The byte order used by the Modbus adapter can be changed from the default Modbus byte ordering (big endian) to Intel byte ordering (little endian) by using this option. By default it is the normal setting for Modbus compatible devices. If the device uses Intel byte ordering, setting this option to false will enable the Modbus driver to properly read Intel formatted data.</p>

<b>Common Channel Attribute</b>	<b>Default</b>	<b>Valid Values</b>	<b>Required</b>	<b>Comments</b>
<code>first16BitLow</code>	true	true, false	No	<p><b>First 16-bit Low in 32-Bit Data Types</b></p> <p>Two consecutive registers' addresses in a Modbus device are used for 32-bit data types, like Integer and Float. It can be specified whether the adapter should assume the first 16 bits is the low or the high word of the 32-bit value. The default, first word low, follows the convention of the Modicon Modsoft programming software. This is also applicable to the two 32-bit data in 64-bit data types, like Long and Double.</p>
<code>first32BitLow</code>	true	true, false	No	<p><b>First 32-bit Low in 64-Bit Data Types</b></p> <p>Four consecutive registers' addresses in a Modbus device are used for 64-bit data types, like Long and Double. It can be specified whether the driver should assume the first 32 bits is the low or the high double word of the 64-bit value. The default, first 32 bits low, follows the default convention of 32-bit data types.</p>

Common Channel Attribute	Default	Valid Values	Required	Comments																																																																
<code>mostSigBit</code>	false	true, false	No	<p><b>Use Modicon Bit Ordering</b></p> <p>Specifies whether the most significant bit is bit 0/1. When this option is true, the adapter will reverse the bit order on reads and writes to registers to follow the convention of the Modicon Modsoft programming software. For example, a write to address 40001.0/1 will affect bit 15/16 in the device when this option is enabled. This option is false by default. For example, <code>bitBaseAddress=0</code> and <code>mostSigBit=true</code> for value 0x1357, the first bit is 0 and the 15th bit is 1.</p> <p>For example, <code>bitBaseAddress=0</code> and <code>mostSigBit=true</code> for value 0x1357, the first bit is 0 and the 15th bit is 1.</p> <table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr> </table> <p>For example, <code>bitBaseAddress=0</code> and <code>mostSigBit=false</code> for value 0x1357, the first bit is 1 and the 15th bit is 0.</p> <table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	1	0	0	1	1	0	1	0	1	0	1	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1	1	1	0	1	0	1	0	1	1	0	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																					
0	0	0	1	0	0	1	1	0	1	0	1	0	1	1	1																																																					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																					
1	1	1	0	1	0	1	0	1	1	0	0	1	0	0	0																																																					

### TCP/IP-Specific Channel Configuration

The following attributes are used specifically for TCP/IP communication.

TCP/IP Channel Attribute	Required	Comments
<code>tcpIpAddress</code>	Yes	Specifies the IP address or host name of the Modbus slave device.
<code>tcpIpPort</code>	Yes	Specifies the port number of the Modbus slave device. Generally, 502 is used.

## Serial-Specific Channel Configuration

Attributes for serial communication are listed in the following table:

Serial Channel Attribute	Default	Valid Values	Required	Comments																						
encoding	RTU	RTU, ASCII	No	<p>Serial Modbus connections can use two basic transmission modes, ASCII or remote terminal unit (RTU) . The transmission mode in serial communications defines the way the Modbus messages are coded. With Modbus/ASCII, the messages are in a readable ASCII format. The Modbus/RTU format uses binary coding, which makes the message unreadable when monitoring but reduces the size of each message. This allows for more data exchange in the same time span. All nodes on one Modbus network segment must use the same serial transmission mode. A device configured to use Modbus/ASCII cannot understand messages in Modbus/RTU and vice versa. Depending on the encoding mode, the Start Bit and Data Bits will be set accordingly as below. Depending on the Parity value, the Stop Bits will be set accordingly as in the following table.</p> <table border="1"> <thead> <tr> <th></th> <th>RTU</th> <th>ASCII</th> </tr> </thead> <tbody> <tr> <td>Character Set</td> <td>Binary 0 ...255</td> <td>ASCII 0..9 and A..F</td> </tr> <tr> <td>Start Bit</td> <td>1</td> <td>1</td> </tr> <tr> <td>Data Bits</td> <td>8</td> <td>7</td> </tr> <tr> <td>Parity</td> <td>EVEN/ODD</td> <td>NONE</td> <td>EVEN/ODD</td> <td>NONE</td> </tr> <tr> <td>Stop Bits</td> <td>1</td> <td>2</td> <td>1</td> <td>2</td> </tr> </tbody> </table>		RTU	ASCII	Character Set	Binary 0 ...255	ASCII 0..9 and A..F	Start Bit	1	1	Data Bits	8	7	Parity	EVEN/ODD	NONE	EVEN/ODD	NONE	Stop Bits	1	2	1	2
	RTU	ASCII																								
Character Set	Binary 0 ...255	ASCII 0..9 and A..F																								
Start Bit	1	1																								
Data Bits	8	7																								
Parity	EVEN/ODD	NONE	EVEN/ODD	NONE																						
Stop Bits	1	2	1	2																						
portName			Yes	Specifies the serial port name for the communication.																						
baudRate			Yes	Specifies the rate at which information is transferred in a communication channel.																						
parity	NONE	ODD, EVEN, NONE	No	A parity bit, or check bit is a bit added to the end of a string of binary code that indicates whether the number of bits in the string with the value one is even or odd.																						

## Register Configuration

Use the following attributes to configure a node.

Register Attribute	Default	Valid Values	Required	Comments
name			Yes	Unique name of the node

Register Attribute	Default	Valid Values	Required	Comments			
<code>dataType</code>		BOOLEAN, BYTE, SHORT, INTEGER, LONG, FLOAT, DOUBLE, STRING	Yes	Data type of the node.			
<code>address</code>			Yes	Register address. Depending on the value of <code>regBaseAddress</code> , the address can start with 0 or 1.			
<code>registerType</code>		HOLDING, INPUT, COIL, DISCRETE	Yes	Register Type	Object Type	Type of	Comments
				HOLDING	One or multiple 16-byte words	Read-Write	This type of data can be alterable by an application program.
				INPUT	One or multiple 16-byte words	Read-Only	This type of data can be provided by an I/O system.
				COIL	Single bit	Read-Write	This type of data can be alterable by an application program.
				DISCRETE	Single bit	Read-Only	This type of data can be provided by an I/O system.
<code>description</code>			No	Description of the node.			
<code>count</code>	1		No	Only valid for STRING data type to specify the number of registers where the string will be accessed.			
<code>bitIndex</code>	1	0~15 (if <code>bitBaseAddress=0</code> ); 1~16 (if <code>biBaseAddress=1</code> )	No	Only valid for BOOLEAN data type of HOLDING and INPUT register.			

## Register and Data Types

The Modbus Machine Adapter supports the following register and data types:

Register Type	Access	Data Type	Bits	Number of Registers	Comments
HOLDING	READ WRITE	BOOLEAN	1	1	<code>bitIndex</code> attribute specifies which bit to access
		BYTE	8	1	Least significant byte is accessed
		SHORT	16	1	
		INTEGER	32	2	
		LONG	64	4	
		FLOAT	32	2	
		DOUBLE	64	4	
		STRING			depends on value of <code>count</code> attribute
INPUT	READ ONLY	BOOLEAN	1	1	<code>bitIndex</code> attribute specifies which bit to access
		BYTE	8	1	Least significant byte is accessed
		SHORT	16	1	
		INTEGER	32	2	
		LONG	64	4	
		FLOAT	32	2	
		DOUBLE	64	4	
		STRING			depends on value of <code>count</code> attribute
COIL	READ WRITE	BOOLEAN	1		
DISCRETE	READ ONLY	BOOLEAN	1		

## MQTT Client

Message Queuing Telemetry Transport (MQTT) is a simple, lightweight messaging protocol used in a publish/subscribe model.

It is designed for constrained devices and low-bandwidth, high-latency or unreliable networks. Use the MQTT client to publish messages to a broker or to subscribe to a topic to receive messages.

 **Note:** For more information on MQTT, see <http://mqtt.org/>.

The MQTT client exposes all interfaces defined in the [Paho Java Client](#). The Paho Java Client is an MQTT client library written in Java for developing applications that run on the Java Virtual Machine (JVM) or other Java-compatible platforms. The Paho Java Client provides two APIs:

- `MqttAsyncClient`: A fully asynchronous API where activity completion is notified through registered callbacks.
- `MqttClient`: A synchronous wrapper around `MqttAsyncClient` that makes functions appear synchronous to the application.

## Dependencies

A Maven dependency and an OSGi import are required to consume the MQTT client:

- The following Maven dependency is required to consume MQTT client:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>mqtt-client</artifactId>
  <version>{Predix Machine version}</version>
</dependency>
```

- The following OSGi import is required in the consuming bundle:

```
Import-Package: org.eclipse.paho.client.mqttv3;version="[1.0,2.0)",
  org.eclipse.paho.client.mqttv3.persist;version="[1.0,2.0)"
```

## Using the MQTT Client Sample Application

An MQTT client sample application is provided to illustrate how to use this service.

1. Navigate to `<SDK installation location>//sample/sample-apps.zip` and extract the files.
2. In the `sample-apps/sample` folder, open the `sample-mqttclient` application.

 **Note:** See [Building Samples \(page 46\)](#) and [Running Samples in the Predix SDK \(page 48\)](#) for instructions on building and running the sample application.

## *MQTT Machine Adapter*

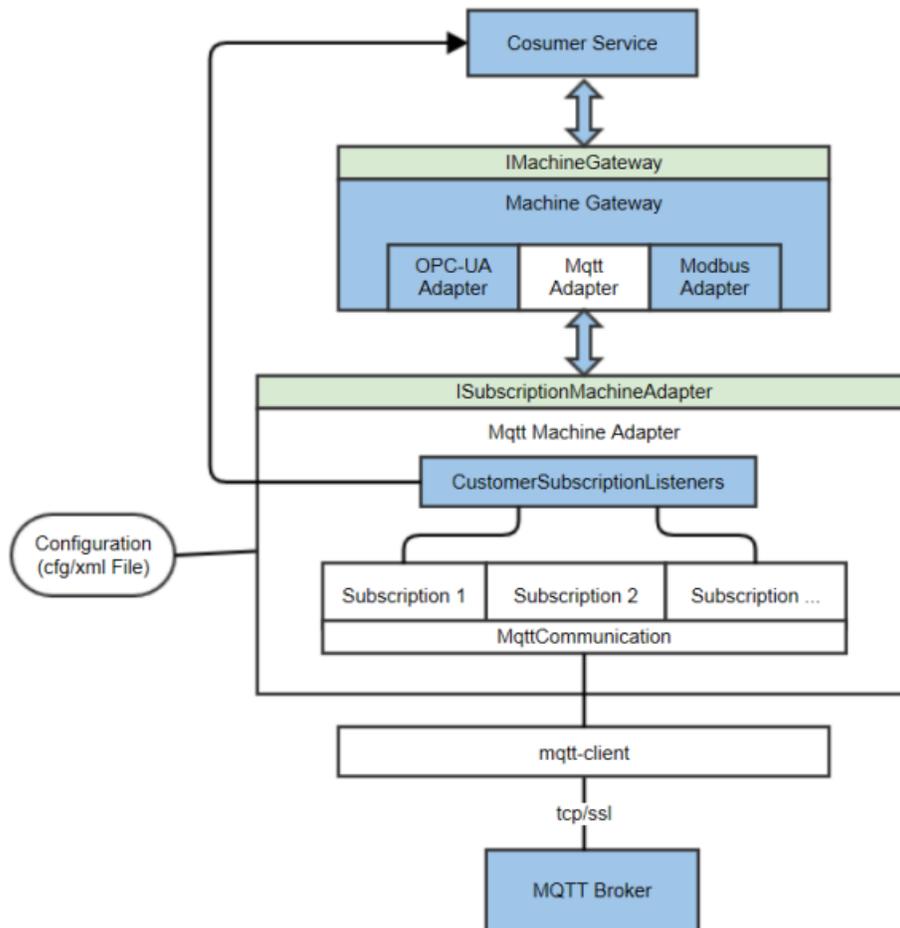
The MQTT Machine Adapter receives data from the MQTT broker before being consumed by the Machine Gateway.

Based on the MQTT protocol, it runs on top of a Predix Machine runtime container as an OSGi service and supports the `ISubscriptionMachineAdapter` interface. The MQTT Machine Adapter appears in the returned list of adapters from Machine Gateway using the `Adapter_Type` property in the `MachineAdapterInfo` class.

The MQTT Machine Adapter subscribes to an MQTT broker to receive messages. The broker publishes messages to the adapter and notifies its subscribers. The MQTT broker converts messages and sends them to the MQTT Machine Adapter as a JSON string in the `PDataValue` class.

Because the MQTT broker publishes to the adapter before notifying subscription listeners, it does not support direct reading from the adapter. In addition, it does not support a subscription interval.

The following diagram illustrates the high-level architecture and flow of messages from the MQTT broker to the subscribers, then to the MQTT Adapter.



## Configuring the MQTT Machine Adapter

To configure the MQTT Machine Adapter, define its connection nodes and subscriptions in the .XML configuration file. A sample one is provided with the container.

- All nodes and subscriptions are statically configured in this file.
- Each node includes `name`, `topic`, `qos`. You can also specify if the data received is the `serializedData` of a list of `PDataValue`
  - If the data is serialized data, the MQTT Machine Adapter will deserialize the data and notify the listeners the list of `PDataValue`.
  - By default, the value of `serializedData` is `false`, which will cause the MQTT Machine Adapter to treat the data as string.

To configure the name, description, and location of the XML configuration file, set the property values in a configuration (.CONFIG) file for each instance.

1. Configure an instance of the MQTT Machine Adapter connection nodes and subscriptions in the XML configuration file, <Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.mqtt-[n].xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mqttMachineAdapterConfig>
  <name>MQTT nodes</name>
  <description>MQTT nodes and subscriptions</description>
  <dataNodeConfigs>
    <dataNode name="Node-1" topic="TestTopic1" qos="0"
serializedData="false" description="TestTopic 1 with Qos 0" />
    <dataNode name="Node-2" topic="TestTopic2" qos="1"
description="TestTopic 2 with Qos 1" />
    <dataNode name="Node-3" topic="TestTopic3" qos="2"
description="TestTopic 3 with Qos 2" />
  </dataNodeConfigs>
  <dataSubscriptionConfigs>
    <dataSubscriptionConfig name="Analytics Subscription">
      <nodeName>Node-1</nodeName>
      <nodeName>Node-2</nodeName>
    </dataSubscriptionConfig>
  </dataSubscriptionConfigs>
</mqttMachineAdapterConfig>
```

2. Configure the name, description, and location of the XML configuration file for each instance of the adapter by setting the property values in the configuration file, <Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.mqtt-[n].config.

When you format values for properties in .config files (as opposed to .cfg files), use the type character followed by a quoted string representation of value. For example, a Boolean property=B"true". Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is property=<type>["<value1>",<value2>"]. For example, array of integer property=I["1", "2", "3"]. A backslash may be used to break up the line for clarity.

Property	Description	Type	Default Value	Required
com.ge.dspmicro.machineadapter.mqtt.config	<p>The path to the MQTT Adapter XML configuration file.</p> <p><b>!</b> <b>Important:</b> If you leave this property blank, the adapter will be disabled.</p>	String	"configuration/machine/com.ge.dspmicro.machineadapter.mqtt-0.xml"	Yes
com.ge.dspmicro.machineadapter.mqtt.name	Unique name of the adapter.	String	"Mqtt Machine Adapter"	Yes
com.ge.dspmicro.machineadapter.mqtt.description	Description of the adapter.	String	Supports basic read/write capability from Mqtt nodes. Supports subscription to a group of Mqtt nodes.	No
com.ge.dspmicro.machineadapter.mqtt.broker	The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients. (for example, tcp://hostname:port or ssl://hostname:port).	String		Yes
com.ge.dspmicro.machineadapter.mqtt.clientid	MQTT client ID to connect to a broker. Each instance of the MQTT adapter should use a different client ID.	String		No
com.ge.dspmicro.machineadapter.mqtt.heartbeatinterval	Heart beat interval in seconds. to check the connection. If disconnected, it will try to reconnect.	Integer	"2"	No
com.ge.dspmicro.mqtt.user	User name for authentication, if the broker requires it.	String		No
com.ge.dspmicro.mqtt.password	Password for authentication, if the broker requires it.	String		No

Property	Description	Type	Default Value	Required
com.ge.dspmicro.machine.gateway.mqtt.password	Encrypted password for authentication, if the broker requires it.	String		No.

## *MQTT River*

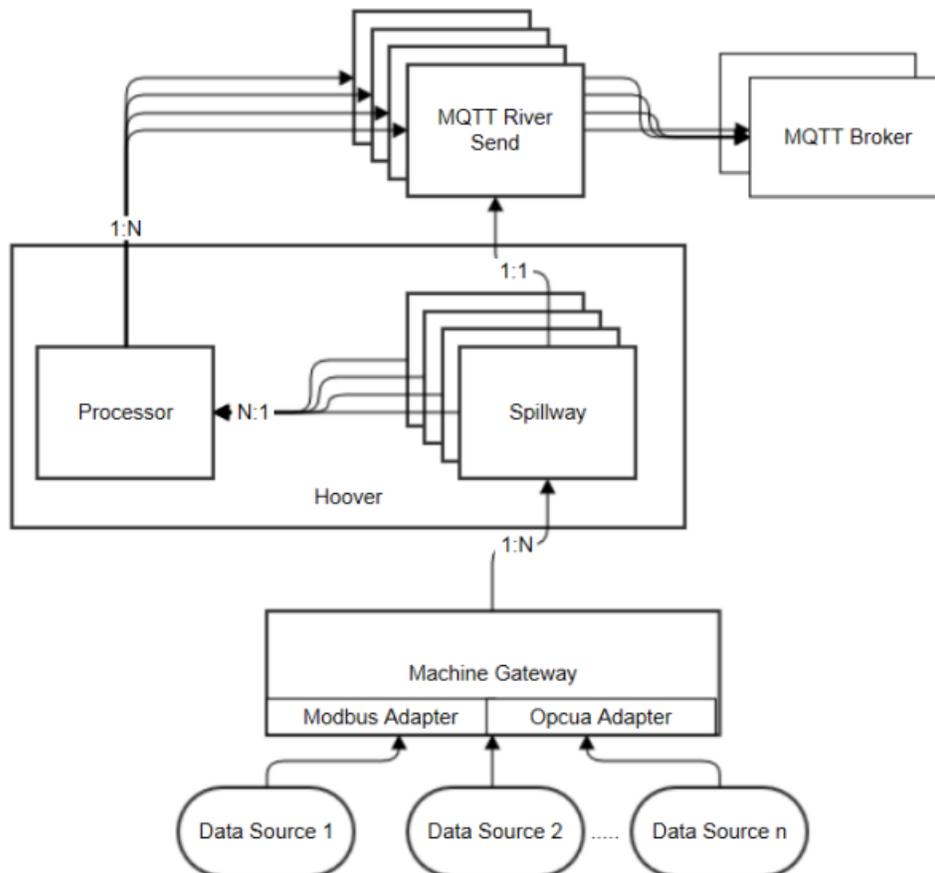
The MQTT River provides a publish/subscribe communication method between edge devices and the Predix cloud.

Edge devices transmit data to the MQTT broker, which receives and filters all messages, then sends the message to MQTT client subscribers located in the Predix cloud or a local network.

You can use the MQTT River to send data or commands from one client to another through the MQTT Broker.

The flow of information from the data source, through the machine gateway to the Hoover spillway and processor to the MQTT River and ultimately, to the MQTT Broker is illustrated in the following figure.

Figure: MQTT River using the Hoover Spillway and Processor



### Note:

When using SSL to connect to the broker, you must also import the MQTT broker server certificate to the client TrustStore, `machine_client_truststore.jks`.

## Dependencies

A Maven dependency and an OSGi import are required to consume the MQTT River.

- The following Maven dependency is required to consume the MQTT River:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>mqttriver-send</artifactId>
```

```
<version>{Predix Machine version}</version>
</dependency>
```

- The following OSGi import is required in the consuming bundle:

```
Import-Package: com.ge.dspmicro.mqttriver.send.api;version="[1.0,2)"
```

## Obtaining the MQTT River Service

Inject the `IMqttRiverSend` service using declarative services.

The following example shows how to inject the MQTT River Send service using declarative services.

```
import com.ge.dspmicro.mqttriver.send.api.IMqttRiverSend;

...

@Reference
public void setService(IMqttRiverSend service)
{
    //set service here
}
```

## Configuring the MQTT River

You can configure the MQTT River by giving it a name and specifying the MQTT broker URI, MQTT client ID, and the topic.

You can also provide authentication information that the MQTT broker might require.

The following circumstances may determine if certain configuration properties are used:

- When the `topic` and `Qos` in `MqttRiverSendRequest` are not set, MQTT River uses these default configuration properties to send the data.
- If the `topic` and `Qos` in `MqttRiverSendRequest` are initialized, MQTT River uses them to send the data.
- When the `transfer(InputStream data, Map<String, String> properties)` method is used to transfer data, if the `topic` and `Qos` properties are not configured with the `MqttRiverSendRequest.PROPERTY_MQTT_TOPIC` and `MqttRiverSendRequest.PROPERTY_MQTT_QOS` keys respectively, the MQTT River uses the default `topic` and `Qos` from the configuration file to send data. .

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean

property=B"true". Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is property=<type>["<value1>","<value2>"]. For example, array of integer property=I["1", "2", "3"]. A backslash may be used to break up the line for clarity.

1. Navigate to <Predix Machine runtime container location>/configuration/machine.
2. Open the com.ge.dspmicro.mqtttriver.send-[n].config file.
3. Set values for the following properties:

Property	Description	Type	Default Value	Required
com.ge.dspmicro.mqtttriver.send-[n].river.uuid	<b>ID of the river.</b> Generated and persisted automatically when the bundle is started.	UUID	(system-generated)	Yes
com.ge.dspmicro.mqtttriver.send-[n].river.name	<b>A unique name for the river.</b> Used to link the river to other services, such as the Spillway.	String	"Mqtt Sender Service"	Yes
com.ge.dspmicro.mqtttriver.send-[n].broker.url	<b>MQTT broker URL.</b> For example, tcp://hostname:port or ssl://hostname.port	String		Yes
com.ge.dspmicro.mqtttriver.send-[n].clientId	<b>Client ID of the MQTT client</b> used to connect to a broker.	String		No
com.ge.dspmicro.mqtttriver.send-[n].filter	<b>A UTF-8 string</b> used by the broker to filter messages for each connected client.	String	"Default MQTT Sender Topic"	Yes

Property	Description	Type	Default Value	Required
com.ge.dspmicro.mqtt	<p><b>QoS</b></p> <p>An agreement between the sender and receiver of a message regarding the guarantees of delivering a message. Valid values are:</p> <ul style="list-style-type: none"> <li>• 0 At most one delivery. The receiver sends no response, and the receiver accepts ownership of the message.</li> <li>• 1 At least one delivery. Ensures that the message arrives at the receiver at least once.</li> <li>• 2 Exactly one delivery. Neither loss nor duplication of messages are allowed.</li> </ul>	Integer	"2"	Yes
com.ge.dspmicro.mqtt	<p><b>Heartbeat interval</b></p> <p>in seconds. to check the MQTT connection. If disconnected, it will try to reconnect.</p>	Integer	"2"	No
com.ge.dspmicro.mqtt	<p><b>User name for authentication</b></p> <p>User name for authentication if required for the MQTT broker.</p>	String		No
com.ge.dspmicro.mqtt	<p><b>Password for authentication</b></p> <p>Password for authentication if required for the MQTT broker.</p>	String		No

## Using MQTT River APIs

You can review the MQTT River APIs to understand how to fully implement the MQTT River service.

1. Navigate to `<SDK installation location>/docs/apidocs.zip` and extract the files.
2. Review the MQTT River API:  
`<SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.mqttriver.send.api`

## Using the MQTT River Sample Application

The Predix Machine SDK provides a sample MQTT River application to show how to use this service.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.
2. In the `<sample-apps>/sample` folder, open the `sample-mqttriver` application.

 **Note:** See [Building Samples from a Command Line \(page 46\)](#) and [Running Samples in Eclipse \(page 48\)](#) for instructions on building and running the sample application.

## *MQTT River Ping*

Use the MQTT River Ping service to send a PING message to the MQTT broker and check MQTT broker connectivity to validate the MQTT River route.

The MQTT Ping service implements the `IValidateRoute` interface. Before the `ping (IPingMessage)` method is called, an `IPingMessage` instance should be created from the MQTT River Ping service by calling the `createPingMessage(Map<String, Object> params, IPongNotification pongNotification)` method. To specify the MQTT river name, the MQTT River route requires a parameter with a `com.ge.dspmicro.river.api.IPingConstants.RIVER_NAME_KEY` key. The `pong` requires a `pong` notification callback to handle the PONG message along the route.

## Dependencies

Maven dependencies and OSGi imports are required to consume the service.

- The following Maven dependencies are required to consume the MQTT River Ping service:

```
<dependency>
```

```

<groupId>com.ge.dspmicro</groupId>
<artifactId>validateroute-api</artifactId>
<version>{Predix Machine version}</version>
</dependency>

```

- The following OSGi imports are required in the consuming bundle:

```
Import Package: com.ge.dspmicro.validateroute.api;version="[1.0,2]"
```

## Obtaining the MQTT River Ping Service

You can obtain the MQTT River Ping service in either of the following ways:

- Obtain the MQTT River Ping service instance, and then call the River interface method `getvalidateRoute()`.
- Inject the MQTT River Ping service using declarative services.

This example shows how to inject the MQTT River Ping service:

```

@Reference(type = '*')
public void setMqttRiverPing(IValidateRoute validateRoute)
{
    if (YourMqttRiverRouteName.equals(validateRoute.getRouteName()))
    {
        //set service here
        this.mqttRiverPing = validateRoute;
    }
}

```

## OPC-UA Adapter

The OPC-UA Machine Adapter is a data-acquisition component that is built based on the OPC-UA standard. The purpose of the adapter is to discover and connect to an OPC-UA server, read/write from a specified node, and subscribe to a specific node.

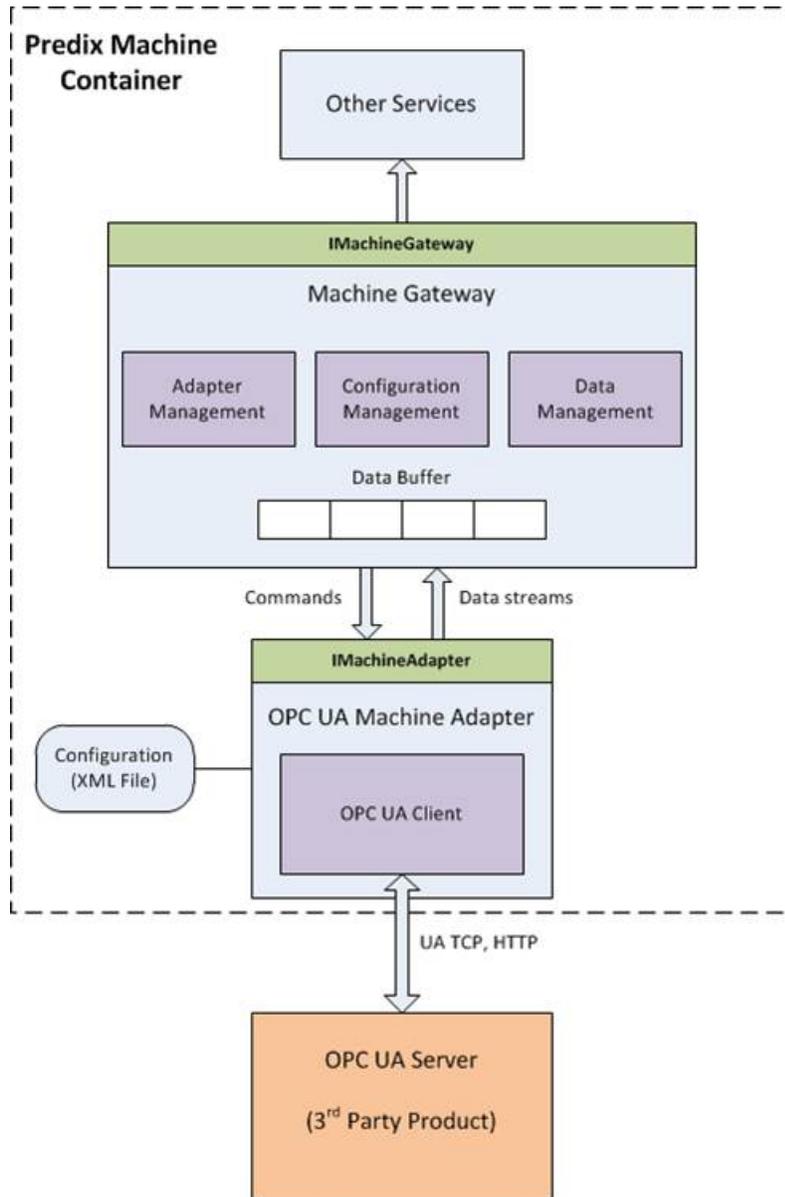
The OPC-UA Machine Adapter runs on top of the Predix Machine runtime container as an OSGi service and supports the `IMachineAdapter` interface. It is consumed through the `Machine Gateway` service like all other adapters.

 **Note:** See [Machine Gateway \(page 119\)](#) for how to obtain adapters.

The OPC-UA Adapter is found in the list of adapters returned from `Machine Gateway` using the `AdapterType` property in the `MachineAdapterInfo` class.

The following diagram shows the high-level architecture:

Figure: OPC-UA Architecture



During industrial asset setup, the third-party OPC-UA server that the OPC-UA adapter connects to should be configured explicitly to enable or disable write access. This adapter has write capabilities. If the use case does not require write access, this permission should be disabled from the server side as a best practice.

### OPC-UA Adapter Communication Patterns with the OPC-UA Server

The OPCUA Machine Adapter subscription has two types of communication patterns with the OPCUA server.

The default pattern is Subscribe. Both patterns are described in the following table.

Pattern	Description
Subscribe	The OPCUA adapter registers monitored nodes to the OPCUA server. Based on the Data Change Filter and interval configured in the subscription or in the node, the OPCUA server will push the data change to the adapter.
Pull	The OPCUA adapter pulls data from the OPCUA server regularly based on the interval of the subscription, regardless whether the data is changed on the server. The Data Change Filter is not applicable in this pattern.

## Configuring the OPC-UA Adapter

You can perform the following actions:

- Configure an instance of the OPC-UA Adapter connection nodes and subscriptions in the `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.opcua-[n].xml` file.

The OPC-UA Adapter uses the OPC-UA Client subscription to monitor the data. The OPC-UA client subscription only notifies the OPC-UA Adapter when the data is changed. The publishing interval specified in the XML file is the interval at which to check whether data has changed. If no data has changed, no data will be pushed to Hoover.

- Customize your implementation of the OPC-UA Adapter by setting property values in the `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.opcua-[n].config` file.

1. Configure the OPC-UA Adapter connection nodes and subscriptions in the `<Predix Machine runtime container location>/configuration/machine`. All read/write nodes and subscriptions are statically configured in this file.

The XML supports two tags for denoting OPC-UA node identifiers. The *StringId* tag supports string identifiers only. For example -

```
<DataNode>
<Name>Node-1</Name>
<StringId>2:temperature</StringId>
</DataNode>
```

The tag *NodeId* supports four OPCUA node identifier types. The standard representation using these types is as follows:

```
ns=<namespace>;<type>=<text>
```

where *namespace* is the namespace index, and *type* can be one of 's' (string), 'i' (numeric), 'g' (GUID) or 'b' (opaque). The text that follows the second '=' sign is a textual representation of the node id and is a string, integral numeric, GUID (without the curly braces) or base 64 encoded opaque data.

The *StringId* tag will continue to be supported for backward compatibility. If both the tags are found under a *DataNode*, the *NodeId* tag will take precedence.

2. To specify which of the time stamps should be included in the data packet sent to the spillway subscribing to this node, set one of the following case-sensitive values in the `TimestampOrigin` element under `DataSubscriptionConfig`.

TimestampOrigin	Description
Adapter	Use the time stamp as the machine adapter (The computer running Predix Machine)
Server	Use the time stamp at the OPC-UA server. (The computer running the OPC-UA server serving the data.)   <b>Note:</b> If the <code>Server</code> time stamp is requested but is not found in the data received, the <code>Source</code> time stamp is used.
Source	Use the time stamp at the data source. (The device where the data originates)   <b>Note:</b> If the <code>Source</code> time stamp is requested but is not found in the data received, the <code>Server</code> time stamp is used.  If neither the <code>Source</code> nor <code>Server</code> time stamps are found, the adapter time stamp is used.

3. To filter data that triggers a data change notification, configure a `DataChangeFilter` for subscription and/or node settings in the XML file:

 **Note:** If the node in the subscription has a specific data filter, it overwrites the data filter of the subscription. If the node in the subscription does not have the specific data filter, only the subscription data filter is applied.

Settings	Description	Default
trigger	<p>The type of change that triggers a notification message. Valid values are:</p> <p><b>Status</b> The status of the value</p> <p><b>StatusValue</b> The value or status of the value.</p> <p><b>StatusValueTimestamp</b> The source time stamp, value, or status of the value.</p>	StatusValue
deadbandType	<p>Indicates if a deadband is applied, and specifies the type of deadband applied.</p> <p>Deadband is a permitted range for value changes that will not trigger a data change notification. Deadband can be applied as a filter when subscribing to Variables and is used to keep noisy signals from updating the Client unnecessarily.</p> <p>Valid values are:</p> <p><b>None</b> No deadband is specified.</p> <p><b>Absolute</b> Deadband based on absolute value change.</p> <p><b>Percent</b> Deadband based on the percentage of the data range.</p>	None

Settings	Description	Default
deadbandValue	<p>Specifies the values for the deadbandTypes. Valid values are:</p> <p><b>Absolute</b></p> <p>Contains the absolute change in a numeric data value that triggers a notification to be generated. Triggers a value change if <math>\text{abs}(\text{last value} - \text{new value}) &gt; \text{DeadbandValue}</math>.</p> <p><b>Percent</b></p> <p>The percentage of the EURange. Only applied to variables having a EURange property. Triggers a value if the value changed more than the percentage of the configured value range.</p>	

An example of the XML configuration file follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OPCUAMachineAdapterConfig>
  <OPCUAClientConfig>
    <AppName>PredixMachine OPCUAAdapter</AppName>
    <ServerUri>opc.tcp://localhost:4841</ServerUri>
    <AppUri>urn:localhost:UA:PredixMachine OPCUAAdapter</AppUri>
    <ProductUri>urn:ge.com:UA:PredixMachine OPCUAAdapter</ProductUri>
  </OPCUAClientConfig>

  <DataNodeConfigs>
    <DataNode>
      <Name>Node 1</Name>
      <StringId>2:Demo.SimulationSpeed</StringId>
    </DataNode>
    <DataNode>
      <Name>Node 2</Name>
      <StringId>2:Demo.Dynamic.Scalar.Int32</StringId>
    </DataNode>
    <DataNode>
      <Name>New_Node 1</Name>
      <StringId>2:Demo.Massfolder_Dynamic.Variable0009</StringId>
    </DataNode>
    <DataNode>
      <Name>Node 3</Name>
```

```

        <NodeId>ns=2;s=Temperature</NodeId>
    </DataNode>
    <DataNode>
        <Name>Node 4</Name>
        <NodeId>ns=2;i=4294967294</NodeId>
    </DataNode>
    <DataNode>
        <Name>Node 5</Name>
        <NodeId>ns=2;g=22bc1f2f-e598-4aac-b7a9-bede30b7711f</
NodeId>
    </DataNode>
    <DataNode>
        <Name>Node 6</Name>
        <NodeId>ns=2;b=YWJjZA==</NodeId>
    </DataNode>
</DataNodeConfigs>

<DataSubscriptionConfigs>
    <DataSubscriptionConfig>
        <Name>OPCUA_Subscription_1</Name>
        <TimestampOrigin>Source</TimestampOrigin>
        <PublishingInterval>1000</PublishingInterval>
        <DataChangeFilter trigger="StatusValue"
deadbandType="Absolute" deadbandValue="1.0"/>
        <DataNode>
            <Name>Node 1</Name>
            <DataChangeFilter trigger="StatusValue"
deadbandType="Absolute" deadbandValue="2.0"/>
        </DataNode>
        <DataNode>
            <Name>Node 2</Name>
        </DataNode>
    </DataSubscriptionConfig>
    <DataSubscriptionConfig pattern="Pull">
        <Name>OPCUA_Subscription_2</Name>
        <PublishingInterval>1500</PublishingInterval>
        <DataNode>
            <Name>New_Node 1</Name>
        </DataNode>
    </DataSubscriptionConfig>
</DataSubscriptionConfigs>

</OPCUAMachineAdapterConfig>

```

4. Navigate to and open the following file: <Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.opcua-[n].config.

5. Set values for the following properties:

Property	Type	Description	Default Value
<code>com.ge.dspmicro.machineadapter.opcua.configFile</code>	String	Name of the configuration file.	
<code>com.ge.dspmicro.machineadapter.opcua.name</code>	String	Identifier for the OPC-UA Machine Adapter.	"OPC-UA Machine Adapter"
<code>com.ge.dspmicro.machineadapter.opcua.description</code>	String	Supports basic read/write capability from OPC-UA nodes. Supports subscription to a group of OPC-UA nodes.	"Supports basic read/write capability from OPC-UA nodes. Supports subscription to a group of OPC-UA nodes."
<code>com.ge.dspmicro.machineadapter.opcua.security.mode</code>	Enum	<ul style="list-style-type: none"> <li>• NONE</li> <li>• BASIC128RSA15_SIGN</li> <li>• BASIC128RSA15_SIGN_ENCRYPT</li> <li>• BASIC256_SIGN</li> <li>• BASIC256_SIGN_ENCRYPT</li> <li>• BASIC256SHA256_SIGN</li> <li>• BASIC256SHA256_SIGN_ENCRYPT</li> </ul>	"NONE"
<code>com.ge.dspmicro.machineadapter.opcua.keystore.path</code>	String	The path (relative to the Predix Machine root) to the keystore containing OPC-UA keypair used for TLS	"security/opcua_kestyore.jks"
<code>com.ge.dspmicro.machineadapter.opcua.keystore.type</code>	String	Keystore type.	"JKS"
<code>com.ge.dspmicro.machineadapter.opcua.keystore.password</code>	String	Password for the keystore. The value is automatically encrypted.	"dspmicro"
<code>com.ge.dspmicro.machineadapter.opcua.key.alias</code>	String	The alias referencing entry in the keystore to use for TLS.	"dspmicro"
<code>com.ge.dspmicro.machineadapter.opcua.key.password</code>	String	The alias password. The value is automatically encrypted.	"dspmicro"
<code>com.ge.dspmicro.machineadapter.opcua.truststore.path</code>	String	The path to the truststore.	"security/machinegateway_truststore.jks"
<code>com.ge.dspmicro.machineadapter.opcua.truststore.type</code>	String	The type of truststore.	"JKS"
<code>com.ge.dspmicro.machineadapter.opcua.truststore.password</code>	String	Password for truststore access.	"dspmicro"
<code>com.ge.dspmicro.machineadapter.opcua.connection.checkInterval</code>	Integer	Interval, in milliseconds, for ensuring connection is valid.	"5"

Property	Type	Description	Default Value
<code>com.ge.dspmicro.machineadapter.opcua.connection.invalid</code>	Integer	Number of server connection checks and retries before invalidating data with BAD quality (-1 turns invalidation off)	"-1"
<code>com.ge.dspmicro.machineadapter.opcua.opcua.server.user</code>	String	The user name for server authentication	
<code>com.ge.dspmicro.machineadapter.opcua.server.password</code>	String	The password for the user.	

## OPC-UA Server

The OPC-UA server allows Predix Machine-enabled applications to expose data through the OPC-UA protocol, a common machine-to-machine protocol.

The service supports multiple instances on different ports and allows you to define application information, port type, and security modes.

The Predix Machine-enabled applications control the OPC-UA node structure.

OPC-UA clients can connect to Predix Machine, which acts as an industrial device, because the server abstracts the devices connected through the Machine Gateway.

## Example Use Cases

The OPC-UA Server supports the following sample use cases:

- A Predix Machine-enabled application aligns Machine Gateway devices with the node structure in the OPC-UA server. Predix Machine acts as a gateway for an external OPC-UA client to connect to this server.
- The OPC-UA River uses OPC-UA as a data-transfer protocol.

## Limitations

The OPC-UA Server only accepts the `PDataValue` format from the Machine Gateway API. Server address space is defined by Machine Gateway Addressing.

## Dependencies

Maven Dependencies and OSGi imports are required to consume the service.

- The following Maven dependencies are required to consume the OPC-UA server:

```
<dependency>
```

```

<groupId>com.ge.dspmicro</groupId>
<artifactId>opcua-server</artifactId>
<version>{Predix Machine version}</version>
</dependency>

```

- The following OSGi imports are required in the consuming bundle:

```
Import-Package: com.ge.dspmicro.opcua.server.api;version="[1.0,2)"
```

## Obtaining the IDataManager Service

Inject the `IDataManager` service using declarative services.

The following example shows how to inject the `IDataManager` service.

```

import com.ge.dspmicro.opcua.server.api.IDataManager;

...

@Reference
public void setService(IDataManager service)

{
    //set service here
}

```

## Configuring the OPC-UA Server

You can configure your implementation of the OPC-UA server by setting property values in the `com.ge.dspmicro.opcua.server-[n].config` file.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to and open the `<Predix Machine runtime container location>/configuration/machine /com.ge.dspmicro.opcua.server-[n].config` file.
2. Set values for the following properties:

Property	Description	Type	Default Value	Required
com.ge.dspmicro.opcua.OpcuaServerName	OPC-UA server name	String	"DefaultOpcuaServer"	Yes
com.ge.dspmicro.opcua.OpcuaServerUri	OPC-UA server URI	String	N/A	Yes
com.ge.dspmicro.opcua.OpcuaServerProductUri	OPC-UA server product URI	String	N/A	Yes
com.ge.dspmicro.opcua.OpcuaServerNamespaceUri	OPC-UA server name space URI	String	"http://www.ge.com/OPCUA/PredixMachineAddressSpace"	Yes
com.ge.dspmicro.opcua.OpcuaDiscoveryServerUri	OPC-UA discovery server URI	String		
com.ge.dspmicro.opcua.OpcuaServerTcpPort	The TCP port that exposes the OPC-UA server.  Leave this value blank to disable TCP.	Integer	"48010"	No
com.ge.dspmicro.opcua.OpcuaServerHttpPort	The HTTP port that exposes the OPC-UA Server.  Leave this value blank to disable HTTP.	Integer	"48080"	No
com.ge.dspmicro.opcua.OpcuaServerHttpsPort	The HTTPS port that exposes the OPC-UA server.	Integer	"48443"	
com.ge.dspmicro.opcua.OpcuaServerSecurityModes	Pipe " " separated list of security modes. Select modes for use by the server. <ul style="list-style-type: none"><li>• NONE</li><li>• BASIC128RSA15_SIGN</li><li>• BASIC128RSA15_SIGN_ENCRYPT</li><li>• BASIC256_SIGN</li><li>• BASIC256_SIGN_ENCRYPT</li><li>• BASIC256SHA256_SIGN</li><li>• BASIC256SHA_SIGN_ENCRYPT</li></ul>	Enum	"NONE"	No. Default selected if left blank.
com.ge.dspmicro.opcua.OpcuaServerKeystorePath	The folder path to the keystore used for TLS.	String	"security/opcuaserver_keystore.jks"	Yes
com.ge.dspmicro.opcua.OpcuaServerKeystoreType	The type of keystore.	String, supported JCE keystore types.	"JKS"	Yes
com.ge.dspmicro.opcua.OpcuaServerKeystorePassword	The password to the keystore.	String	"dspmicro"	Yes

Property	Description	Type	Default Value	Required
com.ge.dspmicro.opcua.alias.of.the.key	The alias of the key to use inside of the configured keystore.	String	"dspmicro"	Yes
com.ge.dspmicro.opcua.password.of.the.key	The password for the key.	String	"dspmicro"	Yes

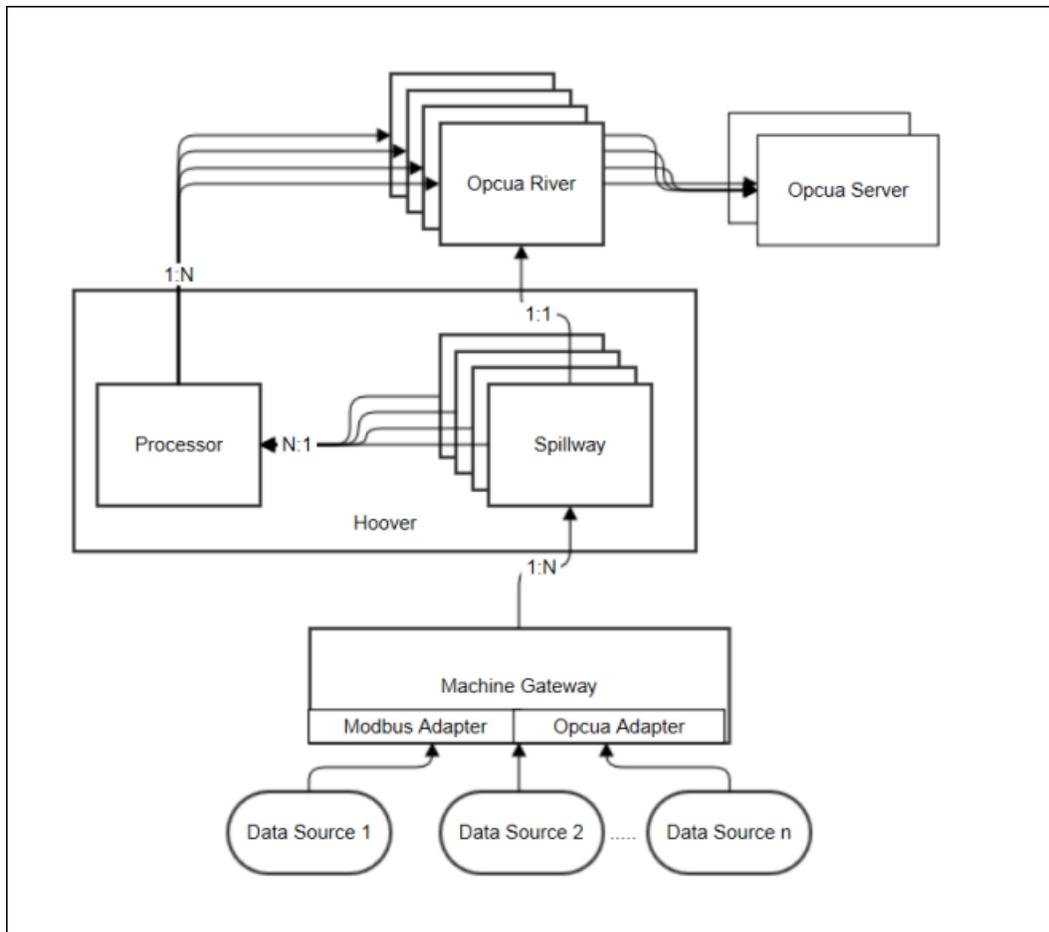
### OPC-UA River

You can transmit data from edge devices to the OPC-UA server that is defined in Predix Machine using the OPC-UA river.

You can collect data from different kinds of edge devices and store the data on an OPC-UA server for aggregation and processing.

The following figure illustrates the flow of data from machine gateway adapters through the machine gateway to the OPC-UA river, and eventually, the OPC-UA server.

Figure: Data Flow from Gateway Adapters to OPC-UA Server



## Dependencies

Maven dependencies and an OSGi import are required to use the OPC-UA river:

- The following Maven dependencies are required to consume the service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>opcuariver-send</artifactId>
  <version>{Predix Machine version}</version>
</dependency>
```

- The following OSGi imports are required in the consuming bundle.

```
Import-Package: com.ge.dspmicro.opcuariver.send.api;version="[1.0,2]"
```

## Limitations

The JSON data from the Machine Gateway adapters uses a specific format. If you have a customized processor or customized `PDataValue.toBytes()` implementation which converts the data from adapters to a different format, the OPC-UA river cannot parse the data and store it to the OPC-UA server. *See Data Conversion.*

## Data Conversion

Machine Gateway adapters send data to the OPC-UA River in the following JSON format:

```
{
  "timestamp":1440608242631,
  "category": "REAL",
  "address": "com.ge.dspmicro.machineadapter.modbus://127.0.0.1:502/2/20",
  "name": "Node-2-1",
  "quality": "NOT_SUPPORTED (20000000)",
  "value": 211,
  "datatype": "INTEGER"
}
```

You can also send batch data, for example:

```
[{
  "timestamp":1493332927450,
  "category": "REAL",
  "address": "com.ge.dspmicro.machineadapter.modbus://127.0.0.1:502/2/20",
  "name": "Node-2-1",
  "quality": "NOT_SUPPORTED (20000000)",
  "value": 212,
```

```

    "datatype": "INTEGER"
  }
},
"timestamp": 1493333100104,
"category": "REAL",
"address": "com.ge.dspmicro.machineadapter.modbus://127.0.0.1:502/2/20",
"name": "Node-2-1",
"quality": "GOOD (0)",
"value": 215,
"datatype": "INTEGER"
]
}

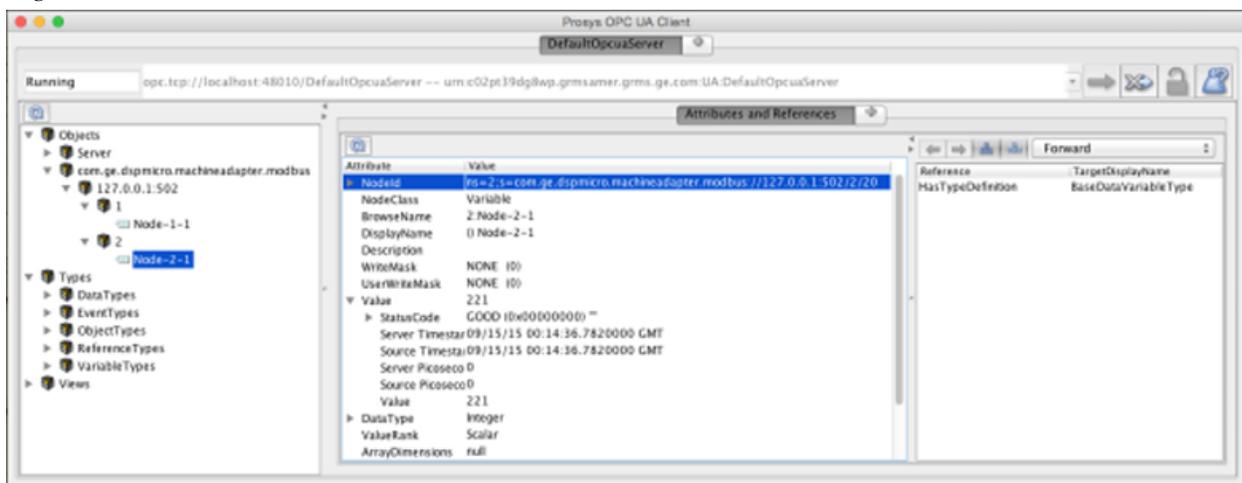
```

The OPC-UA River parses the address information to create the folder and data node on the OPC-UA server. The conversion from the JSON data to a data node on the OPC-UA server occurs in the following manner:

- The protocol, usually the service PID of the adapter, is converted to a folder under `Objects` on the OPC-UA server.
- The host name and port are converted to a device under the `Protocol` folder.
- The path between the port and the last element (exclusive) are converted to a list of folders under the device.
- The whole address is used as the node identifier of the node.
- The node name is used as variable name.
- The data value is converted to the datatype specified in JSON and set to the value on OPC-UA server.

The following image illustrates how the data is converted into a folder structure on the OPC-UA Server:

*Figure: OPC-UA Server*



## Obtaining the OPC-UA River Service

Inject the OPC-UA River service using declarative services.

The following example shows how to inject the service.

```
import com.ge.dspmicro.opcuariver.send.api.IOpcuaRiverSend;

...
@Reference

public void setService(IOpcuaRiverSend service)
{
    //set service here
}
```

## Using the OPC-UA River APIs

Review the OPC-UA River APIs to understand how to fully implement the OPC-UA River.

1. Navigate to and extract all files in the following file: <SDK installation location>/docs/apidocs.zip
2. Navigate to and open the following API: <SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.opcuariver.send.api.

## Configuring the OPC-UA River

You can configure your implementation of the OPC-UA River by specifying the name and ID of the OPC-UA River and providing the name of the OPC-UA server instance.

When you format values for properties in .config files (as opposed to .cfg files), use the type character followed by a quoted string representation of value. For example, a Boolean property=B"true". Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is property=<type>["<value1>",<value2>"]. For example, array of integer property=I["1", "2", "3"]. A backslash may be used to break up the line for clarity.

1. Navigate to and open the following file: <Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.opcuariver.send-[n].config.
2. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.opcuasender.uuid</code>	Unique system-generated identifier for the OPC-UA River.	UUID	N/A	Yes
<code>com.ge.dspmicro.opcuasender.name</code>	Unique name of the OPC-UA River.	String	"Opcua Sender Service"	Yes
<code>com.ge.dspmicro.opcuasender.opcua-server-name</code>	Name of the OPC-UA server instance.	String	"DefaultOpcuaServer"	Yes

## OPC-UA River Ping

Use the OPC-UA River Ping service to send a PING message to the OPC-UA River and check OPC-UA server connectivity to validate the OPC-UA River route.

The OPC-UA Ping service implements the `IValidateRoute` interface. Before calling the `ping(IPingMessage)` method, create an `IPingMessage` instance from the OPC-UA River Ping service by calling the `createPingMessage(Map<String, Object> params, IPongNotification pongNotification)` method. A parameter with a `com.ge.dspmicro.river.api.IPingConstants.RIVER_NAME_KEY` key is required for the OPC-UA River route to specify the OPC-UA river name. A `pongNotification` callback is required to handle the `PONG` message along the route.

## Dependencies

A Maven dependency and an OSGi import are required to consume the service.

- The following Maven dependency is required to consume the OPC-UA River Ping service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>validateroute-api</artifactId>
  <version>{Predix Machine version}</version>
</dependency>
```

- The following OSGi import is required in the consuming bundle:

```
Import Package: com.ge.dspmicro.validateroute.api;version="[1.0,2)"
```

## Obtaining the OPC-UA River Ping Service

You can obtain the OPC-UA River Ping service in the following ways:

- Obtain the instance of the OPC-UA River service and call the `IRiverSend` interface method `getValidateRoute()`.
- Inject the OPC-UA River Ping service using declarative services. The following example shows how to inject the OPC-UA River Ping service:

```
@Reference(type = '*')
public void setOpcuaRiverPing(IValidateRoute validateRoute)
{
    if (YourOpcuaRiverRouteName.equals(validateRoute.getRouteName()))
    {
        //set service here
        this.opcuaRiverPing = validateRoute;
    }
}
```

## *XML Configuration Monitoring and Notification*

The XML Configuration Monitoring and Notification service monitors XML configuration files to determine if they have been modified, added, or removed, and then notifies the Predix Machine services using these XML configuration files about the changes.

These services are called back to respond to changes in their registered XML files

### **Example Use Cases**

The Predix Machine OPC-UA Machine Gateway Adapter refers to an XML configuration file for connection information to the OPC-UA server and metadata on nodes and subscriptions in the OPC-UA server. The OPC-UA Machine Gateway Adapter implements the `IXmlListener` interface to relay notifications when the XML configuration has changed. The OPC-UA adapter can then reinitialize itself to the changes.

### **Functionality**

The XML Configuration Monitoring and Notification process consists of the following steps:

1. The Apache `Fileinstall` service configuration file (`felix.fileinstall.filter=.*\\. (cfg|config|xml)`) in the Predix Machine runtime container allows the `Fileinstall` service to review files with `.xml` extensions for changes.
2. The `Fileinstall` service uses the XML Configuration Monitoring and Notification service, which implements the `org.apache.felix.fileinstall.ArtifactListener` interface to determine that an XML file in the `<Predix Machine runtime container location>/configuration/machine` folder has been modified. The XML Configuration Monitoring and Notification service also implements the

`org.apache.felix.fileinstall.ArtifactInstaller` service to receive notification of all XML updates in the `configuration` folder.

3. Predix Machine services that require XML configuration file change notification implement a `com.ge.dspmicro.device.api.xmlmonitor.IXmlListener` service.
  - a. When the Predix Machine services start, declarative services add the listener to the XML Configuration Monitoring and Notification service.
  - b. When the XML Configuration Monitoring and Notification service adds a Predix Machine service, it queries the new service for the list of XML files to receive notifications.
  - c. When files are updated or deleted, the corresponding function is called, giving the Predix Machine service an opportunity to adapt to the changes.

```
package com.ge.dspmicro.device.api.xmlmonitor;

public interface IXmlListener
{
    void xmlFileAdded(String xmlFile);

    void xmlFileUpdated(String xmlFile);

    void xmlFileDeleted(String xmlFile);

    String[] getXmlFilesList();
}
```

## Assumptions

- The XML Configuration Monitoring and Notification service depends on each service that is registering a callback to provide the name of the file to monitor.
- There is no restriction on different services sharing the same XML configuration file, or how many XML files a service can use. The XML Configuration Monitoring and Notification service stores a list of files with each callback. When updates take place, it provides the name of the updated file in the callback.

## Limitations

- The XML Configuration Monitoring and Notification service needs the `FileInstall` configuration to be modified to watch XML files. If this is not configured, the service is not called with any updates.
- If a service provides a file name that is not in the monitored `configuration` folder, the `FileInstall` service does not provide updates for it.

- If a service provides a file name that does not exist on disk, `FileInstall` does not provide updates for it.
- Only the lowercase `.xml` extension is supported.

## Dependencies

A Maven dependency and an OSGi import are required to consume the service.

- The following Maven dependency is required to consume the service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>device-common</artifactId>
  <version>{Predix Machine Version}</version>
</dependency>
```

- The following OSGi import is required to consume the bundle:

```
Import-Package: com.ge.dspmicro.device.api.xmlmonitor;version="[1.0,2)"
```

## Obtaining the XML Monitoring and Notification Service

Unlike most services in which you can inject a service using declarative services, there are no services to obtain and use from the XML Monitoring and Notification service.

1. Identify services that include and should monitor XML configuration file changes.
2. For each service, provide the `com.ge.dspmicro.device.api.xmlmonitor.IXmlListener` service.

The XML Monitoring and Notification service adds references to those services when they are activated and removes them when they are deactivated.

## *Cloud Gateway Services*

### *Cloud Gateway*

Cloud Gateway manages packages and notifications sent from the cloud.

Packages can be Commands, Containers, Software, or Configurations that can be processed by Predix Machine, or any application Predix Machine can communicate with. For example, Cloud Gateway downloads commands and hands them to the command dispatcher, which gives the command to the corresponding command handler.

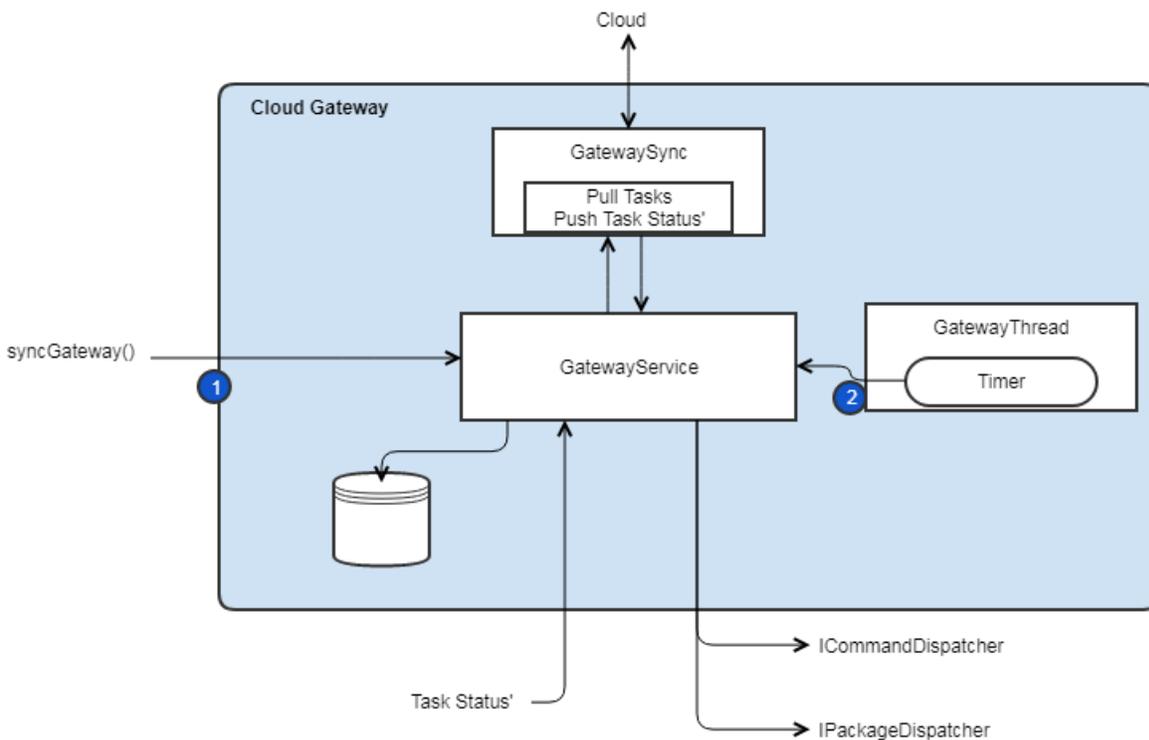
- Cloud Gateway syncs to manage the status uploads and task downloads with Edge Gateway (the cloud) based on these conditions:

An application running on machine sends a sync request.

- Polling interval set in the Cloud Gateway configuration.
- `GatewayThread` is a timer that calls to sync the `GatewayService` with the cloud after a certain period of time specified in the `pollingInterval` configuration property.
- Store and Forward is optional. If the Predix Machine image does not have Store and Forward, the task statuses are saved in memory until the next sync.

**Note:** Task statuses saved in memory may be lost if the machine is restarted.

The following diagram illustrates the different ways Cloud Gateway syncs with the cloud.



## End to End Flow for Commands

Cloud Gateway manages the end to end flow for commands as follows:

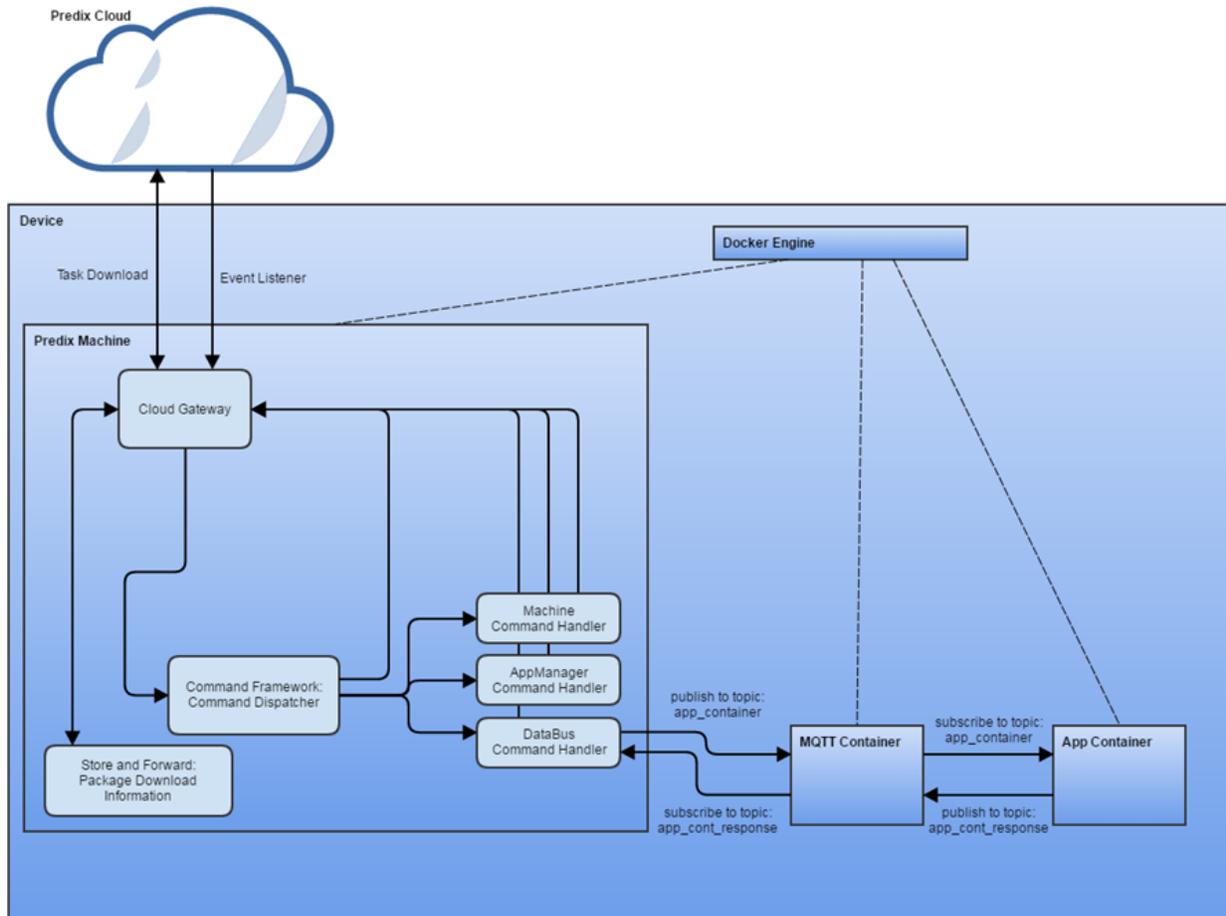
1. Download a command from the cloud.
2. Pass the command to the Command Dispatcher.
3. Command Dispatcher gives the command to the corresponding Command Handler.

4. The Command Handler executes the command and then calls the ICloudGateway with the Status of the command.

If the command is for an app container, the following steps take place before the DataBus Command Handler submits the Status to ICloudGateway:

- a. Assuming the command is for an app container, the DataBus Command Handler gets the command and publishes it to the DataBus.
  - b. The container subscribed to the topic where the command is published receives and executes the command.
  - c. Once the command has been processed, the app container publishes a status for the command.
  - d. The DataBus Command Handler will have also subscribed to the DataBus and receives the Status to the command published from any app containers.
  - e. The DataBus Command Handler then calls the ICloudGateway with the Status of the command.
5. The Cloud Gateway saves the status in Store and Forward (if applicable) until the next sync time with the cloud, or when the cloud requests to process commands.

The following figure shows the end to end flow for commands.



## Cloud Gateway Consumer Configuration

### Maven Dependencies

The following Maven dependencies are required to consume this service:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>cloud-gateway</artifactId>
  <version>17.x.x</version>
</dependency>
```

### OSGI Imports

The following OSGi imports are required in the consuming bundle:

```
Import-Package: com.ge.dspmicro.cloud.gateway.api;version="[1.0,2.0]"
```

### Obtaining the Cloud Gateway Service

Inject the `ICloudGateway` service using declarative services.

The following example shows how to inject the `ICloudGateway` service:

```
import com.ge.dspmicro.cloud.gateway.api.ICloudGateway;
...
@Reference
public void setCloudGateway(ICloudGateway cloudGateway)
{
    //set service here
}
```

### Configuring Cloud Gateway

Configure the polling, machine upload, and status upload intervals as well as the server connection to listen to events.

1. Navigate to `<Predix Machine runtime container location>/configuration/machine`.
2. Open the `com.ge.dspmicro.cloud.gateway.config` file and set values for the following properties:

Property	Description	Type	Required	Default Value
com.ge.dspmicro.cloud.gateway.pollingInterval	<p>The time (in seconds) to wait between the completion of the last poll and starting the next poll to the cloud.</p> <p>The maximum polling interval you can set is 2592000 seconds (1 month). The minimum is 0 seconds. If you use a polling interval of 0 seconds, automatic polling is disabled.</p>	Integer	No	"300"  <b>Note:</b> GE recommends that this value be set to at least 300 or higher.
com.ge.dspmicro.cloud.gateway.machineUploadInterval	<p>The time (in seconds) to wait between uploading machine information to the cloud.</p> <p>The maximum value you can set is 2592000 seconds (1 month). The minimum is 0 seconds. If you set this value to 0 seconds, automatic upload to the cloud of machine information is disabled.</p>	Integer	No	"604800"
com.ge.dspmicro.cloud.gateway.devicesUploadInterval	<p>The time (in seconds) to wait between uploading device status information to the cloud.</p> <p>The maximum value you can set is 2592000 seconds (1 month). The minimum is 0 seconds. If you set this value to 0 seconds, automatic upload to the cloud of device status information is disabled.</p>	Integer	No	"600"

Property	Description	Type	Required	Default Value
<p><code>com.ge.dspmicro.cloud.gateway.dockerUploadInterval</code></p>	<p>This property is used only if you are running containerized Predix Machine.</p> <p>The time (in seconds) to wait between uploading docker information to the cloud.</p> <p>The maximum value you can set is 2592000 seconds (1 month). The minimum is 0 seconds. If you set this value to 0 seconds, automatic upload to the cloud of device status information is disabled.</p>	Integer	Yes (if running containerized Predix Machine)	"604800"
<p><code>com.ge.dspmicro.cloud.gateway.retryInterval</code></p>	<p>The time (in seconds) to wait before retrying connection to cloud when poll to the cloud fails after enrollment.</p> <p>The maximum value you can set is 2592000 seconds (1 month). The minimum is 0 seconds. If you set this value to 0 seconds, it disables any retries upon failure to poll from cloud.</p>	Integer	No	"0"
<p><code>com.ge.dspmicro.cloud.gateway.maxRetries</code></p>	<p>The maximum number of attempts to connect to the cloud after a failure to poll.</p> <p>The minimum is 0 retries.</p>	Integer	No	"0"

Property	Description	Type	Required	Default Value
<code>com.ge.dspmicro.cloud.gateway.uploadTaskStatusOnSubmitEnabled</code>	<p>The default value is <code>true</code>, which means any task statuses are immediately uploaded to the cloud when submitted to the Cloud Gateway service and that Predix Machine retrieves the next pending tasks when it uploads the task status.</p> <p>If set to <code>false</code>, which means the task statuses will be uploaded on the next polling interval.</p>	Boolean	No	B"true"

## Using Cloud Gateway APIs

Review the Cloud Gateway Javadoc API to understand how to use Cloud Gateway.

1. Navigate to, and extract all files in, `<SDK installation location>/docs/apidocs.zip`.
2. Navigate to the Javadoc APIs at `<SDK installation location>/docs/apidocs/index.html`, then find, and click on, the `com.ge.dspmicro.cloud.gateway.api` on the left side panel.

Cloud Gateway APIs provide methods for talking with the cloud for commands, events, alerts, and software updates.

Method	Description
<code>isRunning</code>	Checks whether Cloud Gateway is running.
<code>syncGateway</code>	Pulls any packages from the cloud, on demand.
<code>submitStatus(DefaultStatus)</code>	Returns results for unknown/unsupported tasks.
<code>submitStatus(CommandStatus)</code>	Returns results for Command tasks.

## Package Handler Service

Use the Package Handler service to configure the location for third party package downloads.

You can use the Package Handler service in cases where you want to download and install custom packages using your own installer, or if you are downloading large packages of information from Edge Manager to the device.

## Configuring Package Handler

Configure the name and installation path for the package handler.

 **Note:** You must provide your own installer service, which monitors the installation directory you specify in the package handler configuration file.

1. Navigate to `<Predix Machine runtime container location>/configuration/machine` and make a copy of the default package handler configuration file `com.ge.dspmicro.packagehandler.machine.config`.
2. Rename the file to include a different name or number after the hyphen, for example: `com.ge.dspmicro.packagehandler-1.config`.
3. Configure the name and installation path for the package handler.

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.packagehandler.handler.name</code>	Name of the handler processing the package received.	String	machine	Yes
<code>com.ge.dspmicro.packagehandler.installations.path</code>	Package Framework service will place downloaded package meant for this handler in this directory path may be absolute or relative to Predix Machine root.	String	installation	Yes

Example configuration values:

```
com.ge.dspmicro.packagehandler.handler.name="myhandler"
com.ge.dspmicro.packagehandler.installations.path="my-install-path"
```

4. Once the package is processed by the installation service, you must drop a "status" JSON file in the `<machine>/appdata/packageframework` directory.

 **Note:** The `<status>.json` file must have the name of the downloaded package.

For example:

Name of downloaded package: `A7F4FA68-B76E-4185-8C21-316CF57DAE2F.APPLICATION.deviceid-version.zip`

Expected JSON name: `A7F4FA68-B76E-4185-8C21-316CF57DAE2F.APPLICATION.deviceid-version.json`

The JSON file must have the following four fields:

Field	Type	Description
status	String	The status must be either "success" or "failure."   <b>Note:</b> Only failure statuses are displayed in the EdgeManger UI.
errorcode	Integer	The error code should be zero (0) for success, or in the range of 50 - 255 for failure.  See <a href="#">Upgrade Return Codes (page 71)</a> for more information.
message	String	A message to display for the status. For example, "The configuration could not be updated."
starttime	Long	The start time must be in seconds.
endtime	Long	The end time must be in seconds.   <b>Note:</b> The end time is not displayed in the Edge Manager UI.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

**Example:**

```

{
  "status" : "success",
  "message" : "The configuration was updated successfully.",
  "starttime" : 1489541848,
  "endtime" : 1489541855
}

Failure JSON example
{
  "status" : "failure",
  "message" : "The configuration could not be updated.",
  "starttime" : 1489541848,
  "endtime" : 1489541855
}

```

```
}
```

## Related tasks

Uploading Software and Configuration Packages to the Predix Edge Manager Repository ([page](#))

## Package Handler Execution Logs and Signature Validation

For execution log information to be saved for the installation in Edge Manager, you must save execution log information to a file with the name of the package that was downloaded.

The file must have a `.log` extension and be saved in the `<machine>/logs/installations` directory.

## Signature Validation

After saving the log, the signature file must be dropped into the `<machine>/appdata/packageframework` directory. This tells Predix Machine to check for a `.log` file of this package ID for upload.

 **Note:** Installation happens only after the `.zip` and the `zip.sig` files are both in the `install` folder.

Use the `<Predix_Machine_SDK installation location>/yeti/com.ge.dspmicro.yetiappsiganture-{version}.jar` to validate the signature of the `.zip` before installing. This also unzips the contents into the current folder, after validation.

The following shows an example of running the verification process using the `package.zip` and `package.zip.sig` and unzipping the package:

```
java -Xmx25m -jar "${PREDIX_MACHINE_DATA}"/yeti/
com.ge.dspmicro.yetiappsiganture-17.1.0.jar "$PREDIX_MACHINE_HOME"
"$YOUR_INSTALL_FOLDER/$ZIPNAME.zip" >> $YOUR_INSTALL_FOLDER/
$ZIPNAME.log 2>&1
```

 **Note:** After installation, the custom package handler should clean up the zips and any extra files to preserve space on the device.

## Example

This is an example script that takes, as an argument, the base file name of the `.zip` (package id) that was placed into your `install` folder. It then checks the signature, creates the JSON, log file, and return. For testing purposes, it does not clean up any files.

```
cd "$(dirname "$0")/.."
PREDIX_MACHINE_HOME="$(pwd)"
PREDIX_MACHINE_DATA=${PREDIX_MACHINE_DATA_DIR-$PREDIX_MACHINE_HOME}
PACKAGEFRAMEWORK=$PREDIX_MACHINE_DATA/appdata/packageframework
```

```

LOGS=$PREDIX_MACHINE_DATA/logs/installations
MY_YETI=$PREDIX_MACHINE_DATA/my-install

ZIPNAME="$1"

mkdir "${MY_YETI}/tmp"
cd "${MY_YETI}/tmp"

# This function writes a JSON to the appdata/packageframework directory to
# indicate installation failure
writeFailureJSON () {
    printf "{\n\t\"status\" : \"failure\",\n\t\"message\" : \"${message}\",\n
\t\"starttime\" : \"1489542848\",\n\t\"endtime\" : \"1489542855\"\n}\n" >
    "${PACKAGEFRAMEWORK}/${ZIPNAME}.json"
}
writeSuccessJSON () {
    printf "{\n\t\"status\" : \"success\",\n\t\"message\" : \"${message}\",\n
\t\"starttime\" : \"1489541848\",\n\t\"endtime\" : \"1489541855\"\n}\n" >
    "${PACKAGEFRAMEWORK}/${ZIPNAME}.json"
}

echo "##### My Yeti rules #####" > $MY_YETI/$ZIPNAME.log

# Run the verification process using the package.zip and package.zip.sig
# and unzip the package
java -Xmx25m -jar "${PREDIX_MACHINE_DATA}"/yeti/
com.ge.dspmicro.yetiappsignture-17.1.0.jar "$PREDIX_MACHINE_HOME"
"$MY_YETI/$ZIPNAME.zip" >> $MY_YETI/$ZIPNAME.log 2>&1
if [ $? -ne 0 ]; then
    message="Package origin was not verified to be from the Predix Cloud.
    Installation failed"
    writeFailureJSON
else
    message="Awesome job"
    writeSuccessJSON
fi

echo $message >> $MY_YETI/$ZIPNAME.log
cp $MY_YETI/$ZIPNAME.log $LOGS
cp $MY_YETI/$ZIPNAME.zip.sig $PACKAGEFRAMEWORK

```

## Uploading the Package to Edge Manager

Once you have configured the Package Handler service and created the installation script, you can upload the package to Edge Manager for deployment to enrolled devices.

1. Create a package that includes the configuration or applications you want to install.
2. Create an `install` directory that includes an `install.sh` script.

1. Login to Edge Manager.
2. In the left navigation pane, click **Repository**.
3. In Predix Repository, click **Upload**.
4. In the **Upload** dialog, enter:
  - Name – (Required) name of the software package.
  - Vendor – Device vendor.
  - Type – (Required) Choose from:
    - **Application** – application software for the device.
    - **System** – software that manages the hardware and resources on the device.
    - **Configuration** – device configuration software.
    - **Container** – Docker containers package your application software into a standardized unit for automation of application deployment.
  - Version – (Required) software version.
  - Description – Description for the software or container. You should make the description meaningful, for example, add the location of the device group to which you are pushing the application.
  - Notes – Optionally, you can include additional notes.
    -  **Note:** Notes have a 1024 character limit.
  - Package Handler – name of the package handler you configured.
5. Click **Choose File** to select the files to upload to the repository.
6. Click **Upload**.
7. In the Confirm Upload dialog box, click **Done**.

### Linux/Mac OS Example:

1. Create a folder named `<machine>my-install`.
2. Place the example script inside this folder, and name it `my_yeti.sh`.
3. When the two new files come (`<package-id>.zip` and `<package-id>.zip.sig`), run the script with the `p<package-id>` as an argument, for example: `sh my_yeti.sh <package-id>`.

This unzips the content into a sub-folder called "tmp" and returns a `success` status.

4. After the unzip, the script creates the JSON and `.log` files and places both in the correct locations. Predix Machine then pushes these back to Edge Manager.
5. Edge Manager then shows the correct installation information for the package that was downloaded.

### Related concepts

About Predix Edge Manager Repository ([page](#) )

## Device Detail Service

The Device Detail service retrieves dynamic information for devices and propagates that information to Edge Manager.

The data collected by the Device Detail service is stored in the cloud and may be displayed in the Edge Manager UI, or used for other analytics purposes.

### Device Detail Service Data Flow

1. The hardware vendor writes the device details in JSON files.
2. The Cloud Gateway service periodically requests device details.
3. The Device Detail service reads data from the device and passes it to the Cloud Gateway service.
4. The Cloud Gateway service then passes the information to the Edge Gateway service (may or may not go through the Store and Forward service).
5. The Edge Gateway dispatched the information to the Edge Manager service.
6. The Edge Manager service stores the device information in the database.
7. The Edge Manager UI requests the device details.
8. The Edge Manager service queries the database and returns the information to the Edge Manager UI.

### Device Detail service categories

The table below shows the information provided by the Device Detail service.

Category	Data Source	Description	Data Provided
Device information	Vendor-provided JSON	Static information	Hardware components, SIM card information, and so on.
Device status	Vendor-provided JSON	Dynamic information	Power supply status, Bluetooth, cellular, and Wi-Fi statuses.
	System generated	Dynamic information	CPU utilization and load, memory, boot time, and network interface status.
Software information	Vendor-provided JSON	Software installed on the device from outside of Edge Manager	
Machine information	Information on Predix Machine	Information on Predix Machine	Predix Machine and JDK versions, bundle and component states, and so on.

Category	Data Source	Description	Data Provided
Device properties	Vendor-provided JSON	Custom information for device defined by the user	Generic information of the device not covered by other categories.
Edge alerts	Vendor-provided JSON	Vendor-provided alerts that are propagated to Edge Manager	

## *Requirements for Information Provided by the Vendor*

The following general guidelines apply to the functionality of the vendor-provided information feature:

- The information file must have a `.json` extension. Files with other extensions are ignored.
- JSON files must contain the full set of information. For example, `softwareInfo.json` must contain all installed software. Edge Gateway performs a "delete all," then "insert all" operation to replace old information with new information.
- The new status file must overwrite the old status file so only the latest status is read.
- Files are deleted after a successful read.
- Invalid files are renamed to `<filename>.bad.<timestamp>` for debugging purposes. Invalid files can be invalid JSON files or valid JSON files containing attributes not matching the protobuf definition.
- To remove all records of an array-type category, send an array with an empty item.

For example, to remove all sim info, create a JSON file with an empty object:

```
{"simInfo": [{}]}
```

## *Configuring the Device Detail Service*

The locations of where the Device Detail service picks up vendor-provided JSON files are configurable.

**!** **Important:** The same location must not be used in multiple properties.

1. Navigate to, and open the `<Predix_Machine_Runtime_Container_Location>/configuration/machine/com.ge.dspmicro.devicedetail.config` file.
2. Configure the locations for the vendor-provided JSON files.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.deviceinfo.location</code>	Location of device info JSON files.	String	<code>&lt;Predix_Machine_Root&gt;/appdata/device/detail/deviceinfo</code>	No
<code>com.ge.dspmicro.deviceinfo.status</code>	Location of device status JSON files.	String	<code>&lt;Predix_Machine_Root&gt;/appdata/device/detail/device/status</code>	No
<code>com.ge.dspmicro.deviceinfo.software</code>	Location of device software JSON files.	String	<code>&lt;Predix_Machine_Root&gt;/appdata/device/detail/softwareinfo</code>	No
<code>com.ge.dspmicro.deviceinfo.properties</code>	Location of device properties JSON files.	String	<code>&lt;Predix_Machine_Root&gt;/appdata/device/detail/deviceproperties</code>	No
<code>com.ge.dspmicro.deviceinfo.alerts</code>	Location of device alerts JSON files.	String	<code>&lt;Predix_Machine_Root&gt;/appdata/device/detail/edge/alerts</code>	No

### Device Info JSON Requirements

Device info has JSON requirements for hardware, SIM, and properties.

The device information file must have a `.json` extension and must be dropped in the external location you specify. The default location is `<PREDIX_MACHINE_ROOT>/appdata/device/detail/deviceinfo`.

### JSON Requirements for Hardware info

The `hardwareinfo.json` file has the following requirements:

- `hardwareInfo` should always contain the full set of hardware. Edge Manager services replaces the old information with the new information.
- `hardwareInfo` attributes are arbitrary key-values. Keys (like `"numberOfProcessors"` in the below example) are strings. Values contain string representations of the value and the data type of the data. Refer to the proto definition for a complete list of supported data types.

- Other fields, such as category, manufacturer, model, firmware, and so on, are defined fields.

The following is an example of a hardware information file (`hardwareInfo.json`):

```
{
  "hardwareInfo": [
    {
      "category": "cpu",
      "manufacturer": "ABC Inc.",
      "model": "CPU123",
      "firmware": "1.1.1",
      "attributes": {
        "numberOfProcessors": {
          "value": "1"
        },
        "frequency": {
          "value": "1GHz"
        },
        "numberOfCores": {
          "value": "2"
        }
      }
    },
    {
      "category": "modem",
      "manufacturer": "ABC Inc.",
      "model": "101",
      "firmware": "1.2.3"
    },
    {
      "category": "ups",
      "manufacturer": "ABC Inc.",
      "model": "LT700",
      "firmware": "1.1.1",
      "attributes": {
        "Detail": {
          "value": "http://abc.com"
        },
        "Full load runtimes": {
          "value": "11.5 min"
        },
        "Weight": {
          "value": "70 lbs"
        },
        "VA": {
          "value": "700"
        },
        "Dimensions": {
          "value": "8.125\" W x 17.5\" D x 17.5\" H"
        },
        "Hall load runtimes": {
          "value": "30 min"
        }
      }
    }
  ]
}
```

```
"WATTS": {
  "value": "500"
},
"BTU's/Hour": {
  "value": "256"
}
},
{
  "category": "memory",
  "manufacturer": "ABC Inc.",
  "model": "generic",
  "attributes": {
    "Detail": {
      "value": "http://abc.com"
    },
    "Speed": {
      "value": "1866MHz"
    },
    "Storage Temperature": {
      "value": "-55°C to 100°C"
    },
    "Type": {
      "value": "DDR3"
    },
    "Dimensions": {
      "value": "133.35mm x 32.8mm"
    },
    "Voltage": {
      "value": "1.35V"
    },
    "Capacity": {
      "value": "8GB"
    },
    "Operating Temperature": {
      "value": "0°C to 85°C"
    }
  }
},
{
  "category": "memory",
  "manufacturer": "ABC Inc.",
  "model": "generic",
  "attributes": {
    "Speed": {
      "value": "1333MHz"
    },
    "Type": {
      "value": "DDR3"
    },
    "Voltage": {
      "value": "1.35V"
    }
  }
},
```

```

    "Part Numbers": {
      "value": "M393B1K70CH0-YH9, M393B1K70DH0-YH9"
    },
    "Capacity": {
      "value": "8GB"
    }
  }
]
}

```

## SIM info

The `siminfo.json` file has the following requirements:

- `SimInfo` should always contain the full set of SIM information. Edge Manager services replace the old information with the new information.
- `SimInfo` attributes are arbitrary key-values. Keys (like "firmware" in the below example) are strings. Values contain string representations of the value and the data type of the data. Refer to the `proto` definition for a complete list of supported data types.
- The Timestamp field must be in [RFC3399](#) format and default to epoch time if not specified.

The following is an example of a SIM information JSON file (`simInfo.json`):

```

{
  "simInfo": [
    {
      "iccid": "8991101200003111111",
      "imei": "490154203237512",
      "attributes": {
        "firmware": {
          "value": "1.04.14",
          "dataType": "DATATYPE_STRING"
        },
        "module": {
          "value": "M10",
          "dataType": "DATATYPE_STRING"
        },
        "imsi": {
          "value": "228 01 9876543211",
          "dataType": "DATATYPE_STRING"
        },
        "mno": {
          "value": "Swisscom (Schweiz) AG",
          "dataType": "DATATYPE_STRING"
        }
      }
    },
    {
      "iccid": "8991101200003987654",
      "imei": "490154203237512",

```

```

      "attributes": {
        "firmware": {
          "value": "1.04.13",
          "dataType": "DATATYPE_STRING"
        },
        "module": {
          "value": "M10",
          "dataType": "DATATYPE_STRING"
        },
        "imsi": {
          "value": "228 01 9876543210",
          "dataType": "DATATYPE_STRING"
        },
        "mno": {
          "value": "Swisscom (Schweiz) AG",
          "dataType": "DATATYPE_STRING"
        }
      },
      {
        "iccid": "8991101200003123456",
        "imei": "990000862471854"
      }
    ]
  }

```

## JSON Requirements for Device Info Properties

The `deviceInfoProps.json` file has the following requirements:

- Properties should always contain the full set of properties. Edge Manager services replaces the old information with the new information.
- Attributes are arbitrary key-values. Keys (like "Prop1" in the example below) are strings. Values contains a string representation of the value and the data type of the data. Refer to the `proto` definition for a complete list of supported data types.
- The Timestamp field must be in RFC3399 format and default to the epoch time if not specified.

```

{
  "attributes": {
    "Prop2": {
      "value": "2147483647",
      "dataType": "DATATYPE_INT"
    },
    "Prop1": {
      "value": "This is a string",
      "dataType": "DATATYPE_STRING"
    },
    "Prop7": {
      "value": "false",
      "dataType": "DATATYPE_BOOLEAN"
    }
  }
}

```

```

    },
    "Prop8": {
      "value": "XDFDSLKFIERTW*EGJ$(%*HFKDF" ,
      "dataType": "DATATYPE_BINARY"
    },
    "Prop9": {
      "value": "2016-10-25T03:54:43.230Z" ,
      "dataType": "DATATYPE_TIMESTAMP"
    },
    "Prop3": {
      "value": "9223372036854775807" ,
      "dataType": "DATATYPE_LONG"
    },
    "Prop4": {
      "value": "1.7976931348623157E308" ,
      "dataType": "DATATYPE_DOUBLE"
    },
    "Prop5": {
      "value": "3.4028235E38" ,
      "dataType": "DATATYPE_FLOAT"
    },
    "Prop6": {
      "value": "true" ,
      "dataType": "DATATYPE_BOOLEAN"
    }
  }
}

```

### Related reference

[Device Details Protobuf Definitions \(page 210\)](#)

## Device Status JSON Requirements

Status info has JSON requirements for power supply and connection status.

The device status file must have a .json extension and must be dropped in the external location you specify. The default location is <PREDIX\_MACHINE\_ROOT>/appdata/devicedetail/devicestatus.

JSON requirements for the device status file include the following:

- Status files are written when status is updated. The new status file should always overwrite the old status file so only the latest status is read.
- Set `percentageFull` to `-1` if it is not applicable to the power source.

The following examples are for reference only. See [Device Details Protobuf Definitions \(page 210\)](#) for the data structure definitions.

The following is an example of a power supply status JSON file (`powerSupplyStatus.json`):

```

{
  "powerSupplyStatus": [
    {
      "type": "AC",
      "state": "OK",
      "percentageFull": -1,
      "description": "Everything looks good."
    },
    {
      "type": "Battery",
      "state": "Discharging",
      "percentageFull": 30,
      "description": "Battery low!!!",
      "attributes": {
        "Estimated time remaining (minutes)": {
          "value": "40",
          "dataType": "DATATYPE_INT"
        },
        "isPlugged": {
          "value": "false",
          "dataType": "DATATYPE_BOOLEAN"
        }
      }
    }
  ],
  {
    "type": "UPS",
    "state": "FAULTY",
    "percentageFull": -1,
    "description": "Attention needed."
  }
]
}

```

The following is an example of a Bluetooth connection status JSON file (bluetoothStatus.json):

```

{
  "bluetoothStatus": [
    {
      "enabled": true,
      "connected": true,
      "profile": "PAN",
      "connectedDevice": "iotNetwork"
    },
    {
      "enabled": true,
      "connected": true,
      "profile": "HID",
      "connectedDevice": "wirelessKeyboard"
    },
    {
      "enabled": true,

```

```

    "connected": true,
    "profile": "HID",
    "connectedDevice": "wirelessMouse",
    "attributes": {
      "Manufacturer": {
        "value": "Logitech",
        "dataType": "DATATYPE_STRING"
      },
      "Color": {
        "value": "Pink",
        "dataType": "DATATYPE_STRING"
      }
    }
  }
]
}

```

The following is an example of a cellular connection status JSON file (cellularStatus.json):

```

{
  "cellularStatus": [
    {
      "id": "cell_1",
      "networkMode": "CDMA",
      "dataVolume": 0,
      "signalStrength": {
        "rssi": -55
      }
    },
    {
      "id": "cell_2",
      "networkMode": "LTE",
      "dataVolume": 18446744073709551615,
      "signalStrength": {
        "rssi": -100
      },
      "attributes": {
        "MCC": {
          "value": "310",
          "dataType": "DATATYPE_INT"
        },
        "MNC": {
          "value": "410",
          "dataType": "DATATYPE_INT"
        },
        "Carrier": {
          "value": "AT&T"
        }
      }
    }
  ],
  {
    "id": "490154203237512",
    "networkMode": "LTE",

```

```

    "dataVolume": 132000,
    "signalStrength": {
      "sinr": 29,
      "rsrp": -86,
      "ecio": 0,
      "rssi": -55,
      "rscp": 0,
      "rsrq": -6
    }
  }
]
}

```

The following is an example of a WiFi connection status file (`wifiStatus.json`):

```

{
  "wifiStatus": [
    {
      "enabled": true,
      "connected": true,
      "ssid": "iotNetwork1",
      "attributes": {
        "Network key": {
          "value": "D0D0DEADF00DABBADEAFBEADED",
          "dataType": "DATATYPE_STRING"
        }
      }
    },
    {
      "enabled": true,
      "connected": false,
      "ssid": "iotNetwork2"
    }
  ]
}

```

## *Device Status Properties JSON Requirements*

The device status properties file must have a `.json` extension and must be dropped in the external location you specify. The default location is `<PREDIX_MACHINE_ROOT>/appdata/devicedetail/deviceproperties`.

JSON requirements for the device status file include the following:

- Status files are written when status is updated. The new status file should always overwrite the old status file so only the latest status is read.
- Properties should always contain the full set of properties. Edge Manager services replace the old information with the new information.

- Attributes are arbitrary key-value pairs. Keys (like "Prop1" in the below example) are strings. Values contains a string representation of the value and the data type of the data. Refer to the [Device Details Protobuf Definitions \(page 210\)](#) for a complete list of supported data types.
- The timestamp field must be in RFC3399 format and default to the epoch time if not specified.

The following examples are for reference only. See [Device Details Protobuf Definitions \(page 210\)](#) for the data structure definitions.

The following is an example of a device status properties JSON file (deviceStatusProps.json):

```
{
  "attributes": {
    "Prop2": {
      "value": "2147483647",
      "dataType": "DATATYPE_INT"
    },
    "Prop1": {
      "value": "This is a string",
      "dataType": "DATATYPE_STRING"
    },
    "Prop7": {
      "value": "false",
      "dataType": "DATATYPE_BOOLEAN"
    },
    "Prop8": {
      "value": "XDFDSLKFIERTW*EGJ$%($%*HFKDF",
      "dataType": "DATATYPE_BINARY"
    },
    "Prop9": {
      "value": "2016-10-25T03:54:43.230Z",
      "dataType": "DATATYPE_TIMESTAMP"
    },
    "Prop3": {
      "value": "9223372036854775807",
      "dataType": "DATATYPE_LONG"
    },
    "Prop4": {
      "value": "1.7976931348623157E308",
      "dataType": "DATATYPE_DOUBLE"
    },
    "Prop5": {
      "value": "3.4028235E38",
      "dataType": "DATATYPE_FLOAT"
    },
    "Prop6": {
      "value": "true",
      "dataType": "DATATYPE_BOOLEAN"
    }
  }
}
```

```
}

```

## Software Info JSON Requirements

The software information file must have a `.json` extension and must be dropped in the external location you specify. The default location is `<PREDIX_MACHINE_ROOT>/appdata/devicedetail/softwareinfo`.

JSON requirements for the software information file include the following:

- When the device is first booted and when software is added or updated, the file must be provided.
- The file must contain the full set of installed software. Edge Gateway will replace the old information with the new list.
- The `timestamp` field must be in RFC 3339 format (see the examples below).
- The `type` field can be `UNKNOWN_TYPE` (default), `APPLICATION_TYPE`, `SYSTEM_TYPE`, `CONFIGURATION_TYPE`.
- Only one file is expected. If more than one file is found, the first valid file is used. Extra files will be renamed to `<filename>.extra.<timestamp>`.

The following is an example of a software information JSON file (`softwareInfo.json`):

```
{
  "softwareInfo": [
    {
      "name": "SomeCoolApp",
      "vendor": "Huawei Technologies Co.",
      "type": "APPLICATION_TYPE",
      "version": "1.0.0",
      "installTime": "2016-10-28T03:20:23+00:00",
      "status": "inactive",
      "description": "You wanna run this!",
      "attributes": {
        "Tech Support (phone)": {
          "value": "123-456-7890",
          "dataType": "DATATYPE_STRING"
        },
        "Tech Support (email)": {
          "value": "abcde@huawei.com",
          "dataType": "DATATYPE_STRING"
        },
        "Tech Support (hours)": {
          "value": "24x7",
          "dataType": "DATATYPE_STRING"
        }
      }
    }
  ],
}
```

```

{
  {
    "name": "OpenVPN Client",
    "vendor": "OpenVPN Technologies, Inc.",
    "version": "2.3.12",
    "installTime": "2016-10-25T03:54:43.230Z",
    "status": "active",
    "description": "Securing your traffic"
  },
  {
    "name": "Wind River Linux",
    "vendor": "Wind River",
    "type": "SYSTEM_TYPE",
    "version": "7.0.0.18",
    "installTime": "2016-10-24T18:15:30-05:00",
    "status": "active",
    "description": "Blah"
  }
]
}

```

## Edge Alerts JSON Requirements

There are specific JSON requirements for the edge alerts file.

The edge alerts file must have a `.json` extension and must be dropped in the external location you specify. The default location is `<PREDIX_MACHINE_ROOT>/appdata/devicedetail/edgealerts`.

JSON requirements for the edge alerts file include the following:

- Edge Manager uses `alertType`, `sourceType`, and `source` as keys. When multiple alerts have the same values for these three fields, only the latest is displayed in the Edge Manager UI.
- The `timestamp` field must be in RFC3399 format and default to the epoch time if not specified.

The following example is for reference only. See [Device Details Protobuf Definitions \(page 210\)](#) for the data structure definitions.

The following is an example of an edge alerts JSON file:

```

{
  "edgeAlert": [{
    "alertType": "SIM_USAGE",
    "deviceId": "device1",
    "sourceType": "ALERT_SOURCE_SIM",
    "source": "8991101200003111111",
    "severity": "ALERT_SEVERITY_ERROR",
    "timestamp": "2017-02-09T02:47:57.178Z",
    "description": "SIM usage exceeded 100% of usage plan",
    "status": "ALERT_STATUS_OPEN"
  }, {

```

```

    "alertType": "CELLULAR_STRENGTH",
    "deviceId": "device2",
    "sourceType": "ALERT_SOURCE_DEVICE",
    "source": "Device12345",
    "severity": "ALERT_SEVERITY_WARNING",
    "timestamp": "2017-02-09T02:47:57.181Z",
    "description": "Cellular signal weak",
    "status": "ALERT_STATUS_ACKNOWLEDGED"
  }
}

```

### Related tasks

Viewing and Filtering Alerts ([page](#))

### Related information

([page](#))

## *Device Details Protobuf Definitions*

### Supported Data Types .proto file

The following example `common_value.proto` file describes the supported data types that are common across the device details. The values contain a string representation of the value and the data type of the data.

```

/*
 * Copyright (c) 2017 General Electric Company. All rights reserved.
 *
 * The copyright to the computer software herein is the property of
 * General Electric Company. The software may be used and/or copied only
 * with the written permission of General Electric Company or in accordance
 * with the terms and conditions stipulated in the agreement/contract
 * under which the software has been supplied.
 */

/*
 * Describes the structures of key-value pairs.
 */

syntax = "proto3";

package com.ge.predixmachine.protobuf;

option java_multiple_files = true;
option java_package = "com.ge.predixmachine.datamodel.common";
option java_generate_equals_and_hash = true;

// A generic value in any of the supported datatypes.
message Value
{

```

```

// The value in string format
// Default: Empty string
string value = 1;

// Type of data represented in value field
// Default: DataType.DATATYPE_STRING
DataType data_type = 2;
}

enum DataType
{
// Default
// A string of UTF-8 encoded characters.
// Default: Empty string
DATATYPE_STRING = 0;

// A byte array
// Default: Empty array
DATATYPE_BINARY = 1;

// The boolean data type has only two possible values: true and false
// Default: false
DATATYPE_BOOLEAN = 2;

// 32-bit floating point number
// Default: 0.0f
DATATYPE_FLOAT = 3;

// 64-bit floating point number
// Default: 0.0d
DATATYPE_DOUBLE = 4;

// 32-bit signed two's complement integer
// Minimum value: -2^31
// Maximum value: 2^31-1
// Default: 0
DATATYPE_INT = 5;

// 64-bit signed two's complement integer
// Minimum value: -2^63
// Maximum value: 2^63-1
// Default: 0L
DATATYPE_LONG = 6;

// A point in time represented in RFC3399 format
// Examples: "2016-10-28T03:20:23+00:00", "2016-10-25T03:54:43.230Z",
"2016-10-24T18:15:30-05:00"
// Default: UTC Epoch time ("0001-01-01T00:00:00Z")
DATATYPE_TIMESTAMP = 15;

// Other Proto supported types
// DATATYPE_UINT32
// DATATYPE_UINT64

```

```

// DATATYPE_SINT32
// DATATYPE_SING64
// DATATYPE_FIXED32
// DATATYPE_FIXED64
// DATATYPE_SFIXED32
// DATATYPE_SFIX64
}

```

## Device Info Structures

The following example `edge_device_info.proto` file describes the data structures for device info.

```

/*
 * Copyright (c) 2016 General Electric Company. All rights reserved.
 *
 * The copyright to the computer software herein is the property of
 * General Electric Company. The software may be used and/or copied only
 * with the written permission of General Electric Company or in accordance
 * with the terms and conditions stipulated in the agreement/contract
 * under which the software has been supplied.
 */

/*
 * Describes the structures of Device Info.
 */

syntax = "proto3";

import "google/protobuf/timestamp.proto";
import "common_value.proto";

package com.ge.predixmachine.protobuf;

option java_multiple_files = true;
option java_package = "com.ge.predixmachine.datamodel.gateway";
option java_generate_equals_and_hash = true;

//
// Represents the static information of a device.
// This information is collected on the device and passed to Edge Manager
// via Cloud Gateway.
//
message DeviceInfo
{
    repeated HardwareInfo hardware_info = 1;
    repeated SimInfo sim_info = 2;

    map<string, Value> attributes = 100;
}

//

```

```

// Represents the dynamic information of a device.
// This information is collected on the device and passed to Edge Manager
// via Cloud Gateway.
//
message DeviceStatus
{
    repeated PowerSupplyStatus power_supply_status = 1;
    repeated BluetoothStatus bluetooth_status = 2;
    repeated WifiStatus wifi_status = 3;

    CpuStatus cpu_status = 4;
    MemoryStatus memory_status = 5;
    google.protobuf.Timestamp boot_time = 6;

    repeated NetworkInfo network_info = 7;
    repeated CellularStatus cellular_status = 8;

    repeated DiskStatus disk_status = 9;

    map<string, Value> attributes = 100;
}

//
// Represents the custom information of a device.
// This information is collected on the device and passed to Edge Manager
// via Cloud Gateway.
//
message DeviceProperties
{
    map<string, Value> attributes = 100;
}

//
// A list of vendor-installed software on a device.
// This information is collected on the device and passed to Edge Manager
// via Cloud Gateway.
//
message SoftwareInfoList
{
    repeated SoftwareInfo software_info = 1;
}

// Information of hardware installed on the device.
message HardwareInfo
{
    // Category like cpu, modem, antenna, disk, memory, etc.
    string category = 1;
    // Manufacturer of hardware
    string manufacturer = 2;
    // Model information of hardware
    string model = 3;
    // Firmware information of hardware
    string firmware = 4;
}

```

```
// Other information of hardware
// @Deprecated Use attributes instead.
map<string,string> properties = 100;

// Other information of hardware
// @since 17.1
map<string, Value> attributes = 101;
}

// SIM card information
message SimInfo
{
    string iccid = 1;
    string imei = 2;
    map<string, Value> attributes = 100;
}

// Dynamic status of power supply
message PowerSupplyStatus
{
    string type = 1; // AC, DC, battery, UPS, etc.
    string state = 2;
    int32 percentage_full = 3; // Only for battery & UPS types
    string description = 20;
    map<string, Value> attributes = 100;
}

// Dynamic bluetooth status
message BluetoothStatus
{
    bool enabled = 1;
    bool connected = 2;
    string profile = 3;
    string connected_device = 4;
    map<string, Value> attributes = 100;
}

// Dynamic wifi status
message WifiStatus
{
    bool enabled = 1;
    bool connected = 2;
    string ssid = 3;
    map<string, Value> attributes = 100;
}

// Dynamic cellular status
message CellularStatus
{
    // Identifier of the cellular module
    string id = 1;
}
```

```

// Network mode, for example GSM, WCDMA, TD-SCDMA, LTE, etc.
string network_mode = 3;

// Data in bytes transferred since the beginning of the month
uint64 data_volume = 5;

// Signal strength, for example
// - rssi - Received Signal Strength Indicator (dBm)
// - rsrp - Reference Signal Received Power (dBm)
// - rsrq - Reference Signal Received Quality (dBm)
// - sinr - Signal to Interference plus Noise Ratio (dBm)
// - rscp - Received signal code power (dBm)
// - ecio - Energy per Chip to Interference of Other cell ration (dBm)
map<string, int32> signal_strength = 10;

// Generic properties
map<string, Value> attributes = 100;
}

// Dynamic CPU status
message CpuStatus
{
    double cpu_percent_user = 1;
    // double cpu_percent_nice = 2;
    double cpu_percent_system = 3;
    double cpu_percent_idle = 4;
    // double cpu_percent_iowait = 5;
    // double cpu_percent_irq = 6;
    // double cpu_percent_softirq = 7;
    // double cpu_percent_steal = 8;
    // double cpu_percent_guest = 9;
    // double cpu_percent_guest_nice = 10;

    // Load average over 1, 5, and 15 minutes. The load average is the
    // average number of jobs in the run queue.
    repeated double cpu_load_average = 15;
}

message MemoryStatus
{
    int64 total_bytes = 1;
    int64 free_bytes = 2;
}

message DiskStatus
{
    // Name of disk
    string name = 1;

    // Type of disk
    string type = 2;

    // True if this is the disk running Predix Machine

```

```

    bool machine_disk = 3;

    // Size of disk in bytes.
    int64 total_bytes = 7;

    // Number of unallocated bytes on the disk.
    int64 free_bytes = 8;
}

message NetworkInfo
{
    string name = 1;
    string display_name = 2;
    repeated string ipv4_addresses = 3;
    repeated string ipv6_addresses = 4;
}

// Information of a software package
message SoftwareInfo
{
    string name = 1;
    string vendor = 2;
    SoftwareType type = 3;
    string version = 4;
    google.protobuf.Timestamp install_time = 5;
    string status = 6;
    string description = 20;

    map<string, Value> attributes = 100;
}

// Type of software.
enum SoftwareType
{
    UNKNOWN_TYPE = 0; // Default
    APPLICATION_TYPE = 1;
    SYSTEM_TYPE = 2;
    CONFIGURATION_TYPE = 3;
}

// Information of Predix Machine
message MachineInfo
{
    // Version of Predix machine running on the device.
    string machine_version = 1;

    // Information of the Predix Machine bundles running on the device.
    repeated MachineBundle bundle = 2;

    // Version of the OSGi container running on the device.
    string prosyst_version = 3;

    // Number of days remaining on the OSGi container license.

```

```

    string prosyst_key_expire = 4;

    // Version of the Java running on the device.
    string java_version = 5;

    // Vendor of the Java JDK
    string java_vendor = 6;

    // Information of the Predix Machine components running on the device.
    repeated MachineServiceComponent service = 7;

    // Information of the operating system running on the device.
    OsInfo os_info = 8;
}

message OsInfo
{
    string os_name = 1;
    string os_version = 2;
    string os_arch = 3;
}

message MachineBundle
{
    // Symbolic name of bundle
    string name = 1;
    // Version of the bundle
    string version = 2;
    // State of the bundle
    MachineBundleState state = 3;
}

enum MachineBundleState
{
    UNINSTALLED = 0; // Default
    INSTALLED = 1;
    RESOLVED = 2;
    STARTING = 3;
    STOPPING = 4;
    ACTIVE = 5;
}

message MachineServiceComponent
{
    // Symbolic name of ServiceComponent
    string name = 1;
    // State of the component
    MachineServiceComponentState state = 2;
    // Provided service instances
    string provided_services = 3;
}

enum MachineServiceComponentState

```

```

{
  STATE_UNKNOWN = 0; // Default
  STATE_DISABLED = 1;
  STATE_UNSATISFIED = 2;
  STATE_ACTIVE = 3;
}

////////////////////////////////////
// EDGE ALERT
////////////////////////////////////

//
// Represents the structure of an alert.
// Alerts collected from device or cloud services are propagated to Edge
// Manager.
//
message EdgeAlert
{
  // This determines the handler for the alert
  // Currently supported types:
  // - SIM_USAGE
  // - DEVICE_ONLINE_OFFLINE
  // - OPENVPN_STATUS
  // - CELLULAR_STRENGTH
  string alert_type = 1;

  // Identifier of the device where the alert originates from.
  // Default to empty string.
  string device_id = 2;

  // Type of originator where this alert comes from.
  // Default to ALERT_SOURCE_UNKNOWN if not specified.
  AlertSourceType source_type = 5;

  // Identifier of the source where the alert originates from. (max
  // length: 64)
  // For example, a SIM ID for SIM_USAGE alerts, or OpenVpn client ID for
  // OPENVPN_STATUS alerts, etc.
  string source = 6;

  // Severity of alert.
  // Default to ALERT_SEVERITY_UNKNOWN if not specified.
  AlertSeverity severity = 8;

  // Description of alert. (max length: 255)
  // For example, "SIM usage exceeded 100% of usage plan"
  string description = 10;

  // Timestamp of when this alert is created.
  // Default to the Epoch time if not specified.
  google.protobuf.Timestamp timestamp = 12;
}

```

```

    // Status of alert.
    // Default to ALERT_STATUS_UNKNOWN if not specified.
    AlertStatus status = 15;
}

message EdgeAlertList
{
    repeated EdgeAlert edge_alert = 1;
}

enum AlertSourceType
{
    ALERT_SOURCE_UNKNOWN = 0; // Default
    ALERT_SOURCE_DEVICE = 1;
    ALERT_SOURCE_SIM = 2;
    ALERT_SOURCE_OPENVPN = 3;
}

enum AlertSeverity
{
    ALERT_SEVERITY_UNKNOWN = 0; // Default
    ALERT_SEVERITY_CRITICAL = 2;
    ALERT_SEVERITY_ERROR = 5;
    ALERT_SEVERITY_WARNING = 8;
    ALERT_SEVERITY_INFO = 11;
}

enum AlertStatus
{
    ALERT_STATUS_UNKNOWN = 0; // Default
    ALERT_STATUS_OPEN = 1;
    ALERT_STATUS_ACKNOWLEDGED = 2;
    ALERT_STATUS_CLOSED = 3;
}

```

## *HTTP Client Service*

The HTTP Client service provides a set of APIs to build client-side HTTP/1.1-compliant applications. It is based on the Apache HTTP Client 4.3.5 and exposes the full API of the Apache HTTP Client to the OSGi environment. It also provides a customized helper API to simplify the development of RESTful clients. The Predix Machine HTTP client supports server-side certificate validation.

As an OSGi Service Type, HTTP Client currently registers as either a singleton service or a factory. Three interfaces are supported:

- `IHttpClient` – Exposes functions for simple http connections that are managed by the `PoolingHttpClientManager`.
- `IHttpClientFactory` – For using a private instances of `HttpClient`.

- `IPredixCloudHttpClientFactory` – Constructs an HTTP Client that supports Predix cloud authenticated communication.

## Usage Examples

Add a reference in the member variables of the consuming service's class:

```
private IHttpClient                    httpClient;
```

- Add some dependency injection functions to allow the OSGi framework to inject the `IHttpClient` service instance

```
/**
 * set the Dependency injection of IHttpClient
 *
 * @param httpClient
 *                    the httpClient to set
 */
@Reference
public void setHttpClient(IHttpClient httpClient)
{
               this.httpClient = httpClient;
}

/**
 * clear the Dependency injection of IHttpClient
 *
 * @param httpClient
 *                    the httpClient to clear
 */
public void unsetHttpClient(IHttpClient httpClient)
{
               this.httpClient = null;
}
```

- Set the `HttpClientFactory` service to produce a client that does not share resources with the default client:

```
/**
 * set the Dependency injection of IHttpClient
 *
 * @param httpClientFactory
 *                    the httpClientFactory to set
 */
@Reference
public void setHttpClientFactory(IHttpClientFactory httpClientFactory)
{
               this.httpClientFactory = httpClientFactory;
}
```

```

        this.httpClient = httpClientFactory.createHttpClient();
    }

    /**
     * clear the Dependency injection of IHttpclient
     *
     * @param httpClientFactory
     *         the httpClientFactory to clear
     */
    public void unsetHttpClientFactory(IHttpClientFactory
        httpClientFactory)
    {
        this.httpClientFactory = null;
    }

    /**
     * deactivate component. It is best practice to delete httpClient
     instances created from factories
     *
     */
    @Deactivate
    public void deactivate(ComponentContext ctx)
    {
        if( this.httpClientFactory != null && this.httpClient != null)
        {
            this.httpClientFactory.deleteHttpClient(this.httpClient);
            this.httpClient = null;
        }
    }
}

```

- Use the `IPredixCloudHttpClientFactory` to call REST services in the cloud by using the identity provided in the Predix Cloud Identity Management Service. The client will automatically manage fetching OAuth2 tokens provided by the designated authentication servers in the identity management service. The Predix Cloud version of the service interface is used in the same way as the `HttpClientFactory` interface.

```

    /**
     * set the Dependency injection of IHttpclient
     *
     * @param httpClient
     *         the httpClient to set
     */
    @Reference
    public void setHttpClientFactory(IPredixCloudHttpClientFactory
        httpClientFactory)
    {
        this.httpClientFactory = httpClientFactory;
        this.httpClient = httpClientFactory.createHttpClient();
    }

    /**
     * clear the Dependency injection of IHttpclient

```

```

*
* @param httpClient
*           the httpClient to clear
*/
public void unsetHttpClientFactory(IPredixCloudHttpClientFactory
    httpClientFactory)
{
    this.httpClientFactory = null;
}

/**
 * deactivate component. It is best practice to delete httpClient
 * instances created from factories
 *
 */
@Deactivate
public void deactivate(ComponentContext ctx)
{
    if( this.httpClientFactory != null && this.httpClient != null)
    {
        this.httpClientFactory.deleteHttpClient(this.httpClient);
        this.httpClient = null;
    }
}

```

Use a helper to provide access if necessary:

```

/**
 * get the IHttpClient
 *
 * @return httpClient OSGi registered service.
 */
public IHttpClient getHttpClient()
{
    return this.httpClient;
}

```

To make a connection, use any of the `IHttpClient` API functions that connect to send and/or receive data to/from the server.

```

HttpResponseWrapper response = this.httpClient.get(new URI("<SERVER>" + "/"
    <CONTEXT_PATH>"));

```

## Enabling HTTPS

You can provide your own HTTP/HTTPS context in an HTTP connection, without sharing sensitive configuration information, such as cookie stores, with other applications. By default, the `org.apache.felix.http.enable` property is set to `false` so that only HTTPS is enabled.

 **Note:** See *Security Administration Service: SSL Context and Certificate Management* for more information.

1. Navigate to and open the file `<Predix Machine runtime container location>/security/com.ge.dspmicro.securityadmin.cfg`.
2. Set the value for the following properties:

HTTP Server Property	Description	Value
<code>org.apache.felix.http.enable</code>	HTTP server property.	false
<code>org.apache.felix.http.nio</code>		true
<code>org.apache.felix.http.session.timeout</code>	Allows for the specification of the Session life time, in number of minutes.	5
HTTPS Server Property	Description	Value
<code>org.apache.felix.https.enable</code>	HTTPS server property.	true
<code>org.apache.felix.https.nio</code>		true

## Using HTTP Client APIs

Review the HTTP Client Javadoc API to understand how to implement the HTTP Client service.

1. Navigate to and extract all files in the following file: `<SDK installation location>/docs/apidocs.zip`.
2. Navigate to the Javadoc APIs at `<SDK installation location>/docs/apidocs/index.html /com.ge.dspmicro.httpclient.api`.
3. To enable SSL context for the HTTP Client service, use the `IHttpClient.sslType` method to set the SSL type.

Type	Description
<code>NO_SSL</code>	Allows only HTTP requests.
<code>ALLOW_DEFAULT_CERTS</code>	Allows the use of Java default certificate authorities.

Type	Description
<code>ALLOW_ALL_HOSTNAME_VERIFIER</code>	<p>Allows trust of all certificates without a hostname verifier.</p> <p> <b>Note:</b> This type does not work with client certificate authentication (two-way TLS). To use client certificate authentication without hostname verification, use <code>sslType.ALLOW_DEFAULT_CERTS</code> and <code>SSLConnectionSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER</code> as the second parameter.</p>

## Using the HTTP Client Sample Application

A sample HTTP Client application is provided to illustrate how to use the service.

1. Navigate to `<SDK installation location>/samples/sample-apps.zip` and extract the files.
2. In the `sample-apps/sample` folder, open the `sample-httpclient` application.

 **Note:** See *Building Samples* and *Running Samples in the Predix SDK* for instructions on building and running the sample application.

## Configuring the HTTP Client Service

You can customize your HTTP Client Service. To do so, set values for properties in the HTTP client configuration file.

1. Navigate to and open the configuration file: `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.httpclient.cfg`
2. Set values for the following properties:

Property	Description	Default Value
<code>com.ge.dspmicro.httpclient.connection.maxTotal</code>	Maximum number of allowable open connections.	20
<code>com.ge.dspmicro.httpclient.connection.maxPerRoute</code>	Maximum number of allowable open connections per route.	10
<code>com.ge.dspmicro.httpclient.connection.timeout</code>	Maximum time in milliseconds before the connection times out.	5000

Property	Description	Default Value
<code>com.ge.dspmicro.httpclient.connection.idletimeout</code>	Maximum time in seconds before an idle connection times out.	60
<code>com.ge.dspmicro.httpclient.socket.timeout</code>	Maximum time in milliseconds before a blocked socket times out.	600000

## *HTTP Tunnel*

### HTTP Tunnel

The HTTP Tunnel service facilitates communication of different network protocols through HTTPS. The HTTPS protocol acts as a wrapper for the channel through which tunneled protocols can communicate.

If a network only allows HTTP/HTTPS connections through its firewall, but requires other protocols for communication, you can use the HTTP Tunnel service to communicate using those protocols.

### Example Use Cases

The HTTP Tunnel service supports the following use cases:

- Connecting to Start a Remote Desktop using HTTP Tunnel

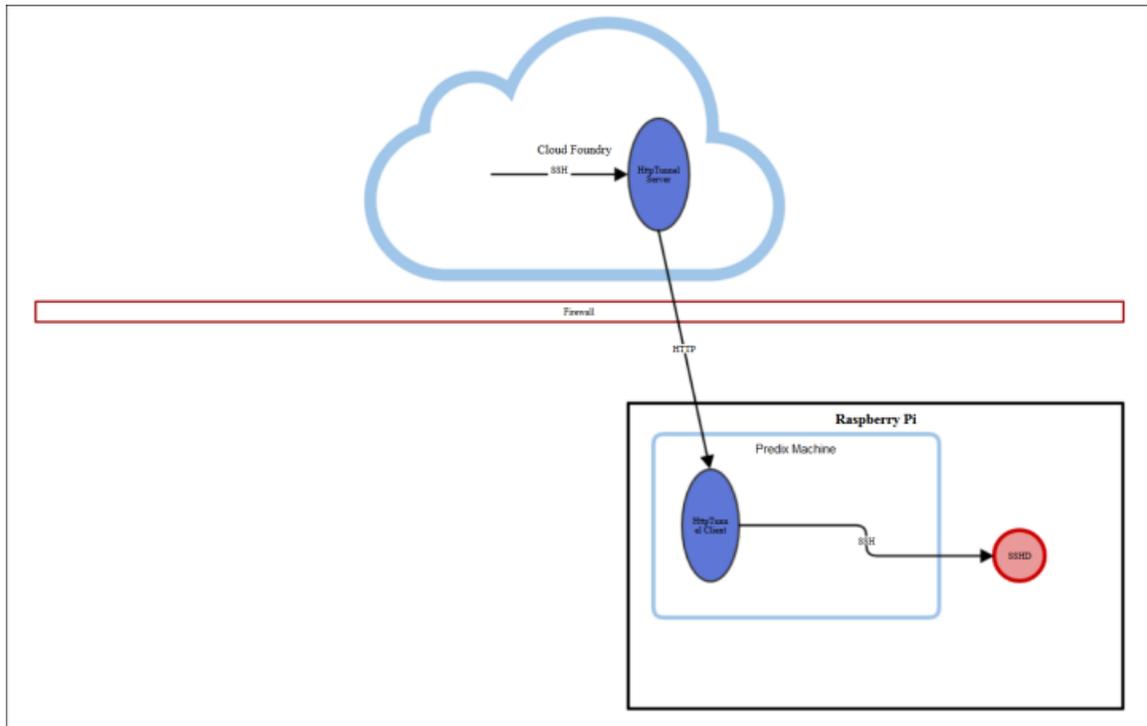
You can remote desktop into a local port and the `HttpTunnel` would tunnel the connection to the remote machine.

- Connecting to a device using SSH.

To connect to a device out in the field, the device must make initial contact with the server. Once the connection is established, you can create an SSH connection to a device through the HTTP Tunnel.

The following diagram illustrates this use case:

Figure: Connecting to a Device Using SSH



## Functionality

The HTTP Tunnel service uses the following client-server model:

- HTTP Tunnel Client – Initiates HTTP Tunnel *connection requests* to the HTTP Tunnel server.
- HTTP Tunnel Server – Receives HTTP Tunnel *connection requests* from the HTTP Tunnel client. It can also initiate HTTP Tunnel *data transfer requests*.

The HTTP Tunnel client always initiates a connection with the HTTP Tunnel server. The server side does not discriminate against its clients, as long as they have permission to access the server. Once the connection is made from the HTTP Tunnel client to the HTTP Tunnel server, the server can initiate a data transfer request.

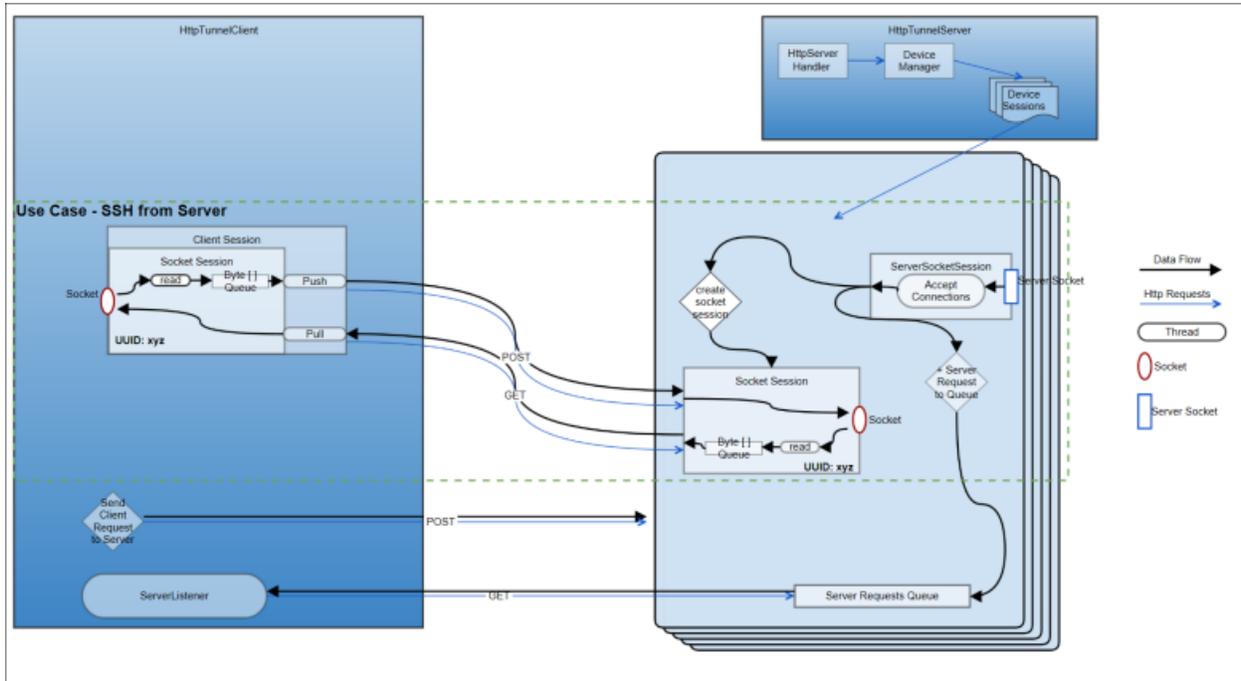
The HTTP Tunnel client uses long polling for the initial connection request. It keeps the connection open for any queued server data transfer requests. As soon as the server initiates a data transfer request, the client acts on the request, the connection is closed, and a new connection is sent to the server for new data transfer requests.

The HTTP Tunnel client also uses long polling to send a `GET` request for data from the server. Once the initial connection has been established, the server and client create sessions to transfer data to each other. One session in the HTTP Tunnel client maps to one session in the HTTP Tunnel server and pulls and pushes data to it. Sessions are created based on which service receives an initial

connection from the user or an application. That service then informs the pairing service to initiate a session with the same id.

The HTTP Tunnel data flow is illustrated in the following diagram:

Figure: Data flow using the HTTP Tunnel service



### Configuring the HTTP Tunnel Client

To use the HTTP Tunnel client, you must configure the SSL Context TrustStore. See [Configuring the Client KeyStore/TrustStore \(page 109\)](#) and set properties for the `com.ge.dspmicro.securityadmin.sslcontext.client.truststore.type`, `com.ge.dspmicro.securityadmin.sslcontext.client.truststore.path`, and `com.ge.dspmicro.securityadmin.sslcontext.client.truststore.password` properties.

You can configure the HTTP Tunnel client to specify the destination host, the maximum number of total connections allowed, and the maximum connection per route. You can also specify the name or type of connection protocol you are using, the protocol port, and the role that the port is playing.

The configuration property values are defined in separate configuration files, one for destination and connection information, and one for protocol and port information.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
------------	-------------	----------

F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>",<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Configure HTTP Tunnel client settings:

a. Navigate to and open the following file: `<Predix Machine runtime container location>/configuration/machine / com.ge.dspmicro.httptunnel.client-[n].config` file.

b. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.httptunnel.client.destinationHost</code>	Route to the HTTP Tunnel server application. (For example, <code>HttpTunnelServer.mycloud.com</code> )	String	N/A	Yes
<code>com.ge.dspmicro.httptunnel.client.name</code>	Name of the HTTP Tunnel client. Must be unique among all HTTP Tunnel Client services in this device.	String	"Default Http Tunnel Client"	Yes
<code>com.ge.dspmicro.httptunnel.client.predixCloud</code>	Indicates whether the HTTP Tunnel Server is located in the Predix cloud ( <code>true</code> ) or located outside of the Predix cloud ( <code>false</code> )	Boolean	B"true"	Yes
<code>com.ge.dspmicro.httptunnel.client.connectTimeout</code>	Indicates the amount of time (in seconds) that the HTTP Tunnel Client will attempt to establish a connection with the server when it first initiates, and when it loses connection. If 0, the HTTP Tunnel Client will always attempt.	Integer	I"0"	Yes

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.http</code>	Maximum size of the data buffer for each session in the HTTP Tunnel Client. Max size of the buffer is in increments of 8Kb of data. (i.e. If the Max size of the buffer is 65536 of 8kb per increment, then the Max Memory size is 512MB)	Integer	I"5000"	Yes
<code>com.ge.dspmicro.http</code>	Indicates the state of the HTTP Tunnel Client on startup.  If <code>true</code> , the tunnel will be on as long as the container is running.  If <code>false</code> , a command must be sent from the Device Manager to start the HTTP Tunnel Client Session.	Boolean	B"false"	Yes

2. Configure HTTP Tunnel client protocol settings:

- a. Navigate to and open the following file: <Predix Machine runtime container location>/configuration/machine/`com.ge.dspmicro.httptunnel.client.protocol-ssh.config`.
- b. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro</code>	Determines which HTTP Tunnel Client this configuration is for.	String	"Default Http Tunnel Client"	Yes
<code>com.ge.dspmicro</code>	Port to open for reading or writing tunneled data.   <b>Note:</b> Will only listen to local host connections when pushing to the tunnel server.	Integer	I"0"	Yes

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.protocol.name</code>	Used to map to the same protocol in the HTTP Tunnel Server. Must be unique among the other protocol configurations running in this container.  For example, you cannot run two SSH Tunnels over the same HTTP Tunnel Client.	String	"SSH"	Yes
<code>com.ge.dspmicro.protocol.name.initiator</code>	Indicates who initiates the Request to tunnel data: Server or Client  If the Client initiates Requests, the Server must have a property file <code>com.ge.dspmicro.httptunnel.server.protocol-[n].cfg</code> with the same <code>protocol.name</code> property as the configuration property in this file so that the server knows where to transfer data it receives.  If the Server initiates a Request, there is no need to include a configuration file in the Server because the Http Tunnel Client tells the Server what protocol it is allowed to Tunnel.	String	"Server"	Yes

## Configuring the HTTP Tunnel Server

You can configure the HTTP Tunnel server to specify the server port listening to the HTTP Tunnel client.

1. Extract all files from the `<SDK installation location>/samples/sample-cloud-apps.zip` file.
2. Access the `tunnelServer` file at `<SDK installation location>/samples/sample-cloud-apps/sample/httptunnel-server`.
3. Set values for the following parameters:

Parameter	Description
<code>KEYSTORE_FILE</code>	The KeyStore file location.
<code>KEYSTORE_TYPE</code>	The KeyStore type.

4. Navigate to and open the following file: `<SDK installation location>/samples/sample-cloud-apps/sample/httptunnel-server/etc/com.ge.dspmicro.httptunnel.server.config`
5. Set values for the following properties:

Property	Description	Type	Default Value	Required
com.ge.dspmicro	The port that listens to the HTTP Client.	Integer	I"8087"	Yes
com.ge.dspmicro	Indicates the amount of time (in seconds) that the HTTP Tunnel Server will wait for a connection to be established by the HTTP Tunnel Client when it first initiates and when it loses connection. If 0, the HTTP Tunnel Server will not timeout.	Integer	I"600"	Yes
com.ge.dspmicro	The maximum size of the data buffer for each session in the HTTP Tunnel Server. Buffer is in increments of 8Kb of data. (For example, if the Max size of the buffer is 65536 of 8kb per increment, then the Max Memory size is 512MB)	Integer	I"5000"	Yes
com.ge.dspmicro	Allows external connections.  Accept external connections when making RDP, SSH, or any protocol requests made to a device.  Setting property to <code>false</code> will only allow connections made from inside of machine running http tunnel server.	Boolean	B"false"	Yes
com.ge.dspmicro	Port mapping for devices listening to Http Tunnel Server requests.  This parameter allows the user to specify a port on the server to listen to connections being tunneled to the Http Tunnel Client.  If a device listening to connections is not listed, a random port will be opened and listed in the <code>&lt;Standalone&gt;/deviceports.json</code> file or <code>&lt;PredixMachineOSGiContainer&gt;/machine/bin/vms/jdk/deviceports.json</code>  The array format is: <pre>[ "&lt;DeviceID&gt;:&lt;Protocol&gt;:&lt;Port&gt;", \   "&lt;DeviceID&gt;:&lt;Protocol&gt;:&lt;Port&gt;", ]</pre>			No

6. Navigate to and open the following file: `<SDK installation location>/samples/sample-cloud-apps/sample/http tunnel-server/etc/com.ge.dspmicro.http tunnel.server.protocol-[n].cfg`

7. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.http.tunnel.protocol</code>	The name of the protocol trying to tunnel.	String		Yes
<code>com.ge.dspmicro.http.tunnel.open.port</code>	The port to open to tunnel the named protocol.	Integer	1"	Yes
<code>com.ge.dspmicro.http.tunnel.write.destination.host</code>	Destination to write data when received from client. If left blank, server will write to localhost	String	"localhost"	No

### Configuring Predix Cloud Identity Management Service for HTTP Tunnel

You can configure the Predix Cloud Identity Management Service for the HTTP tunnel either by setting values in the `.config` file, or by using Predix Cloud Enrollment in the Technician Console, which populates the property values.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to and open the following file: `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.predixcloud.identity.config`.
2. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.predixcloud.oauth.url</code>	The Predix cloud URL of an OAuth2 authorization endpoint. This is the UAA URL for the technician to log into the cloud.	String		Yes (Only when connecting to the HTTP Tunnel Server in the Predix cloud)

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.predixcloud.identity.uaa.enrollment.endpoint</code>	<b>Predix cloud</b> enrollment endpoint URL. This URL is called once when creating credentials during enrollment.	URL String		Yes (Only when connecting to the HTTP Tunnel Server in the Predix cloud)
<code>com.ge.dspmicro.predixcloud.identity.uaa.token.url</code>	<b>The Predix cloud</b> User Account and Authentication (UAA) endpoint for getting a token using <code>client_credentials</code> grant type.	URL String		Yes (If not trying to connect to the cloud, any valid URL will work. For example, http://localhost.)
<code>com.ge.dspmicro.predixcloud.identity.uaa.clientid</code>	<b>The client ID</b> assigned to the device. If this property has a value, this device is enrolled . If it is blank, the device is not enrolled.	String		Yes
<code>com.ge.dspmicro.predixcloud.identity.uaa.clientid</code>	<b>Predix device</b> identity. The Client ID is used to identify the device when trying to connect to the server. This should be a unique ID so that the server can identify the different HTTP Tunnel Clients making requests.	String		No

## *HTTP Proxy Settings Overview*

You can specify HTTP proxy settings using the Proxy Configuration service to store and provide the proxy settings for a Predix Machine-enabled device in a single location, which also allows you to specify destinations that do not require proxies

### Apache Proxy Configuration Service

The Apache Proxy Configuration service stores and provides the proxy settings for a Predix Machine-enabled device in a single location.

This relieves interested services from having to implement proxy settings on their own configuration files.

Because services use the proxy information in a variety of ways, values for the proxy host name, proxy port, and proxy user name and password are provided in the raw form in the `org.apache.http.osgi.services.ProxyConfiguration` interface.

- `String getHostname()`
- `int getPort()`
- `String getUsername()`
- `String getPassword()`
- `boolean isEnabled()`
- `String[] getProxyExceptions();`

## Example Use Cases

The following use cases provide examples of the Apache Proxy Configuration service.

1. Predix Machine's HTTP Client needs to make a proxied connection to external servers. Instead of implementing proxy settings in its configuration file, it consumes the Proxy Configuration service running in the Predix Machine runtime container, which also provides the settings.
2. Predix Machine's WebSocket Client needs to make a proxied connection to external servers. Instead of implementing proxy settings in its configuration file, it consumes the Proxy Configuration service running in the Predix Machine runtime container, which also provides the settings. The container then applies the settings to its `org.eclipse.jetty.websocket.client` fragment.

## Dependencies

Maven dependencies and an OSGi import are required to consume this service:

- ◦ The following Maven dependencies are required to consume the Proxy Configuration service:

```
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient-osgi</artifactId>
  <version>4.4.1</version>
  <scope>provided</scope>
</dependency>
```

- The following OSGi imports are required in the consuming bundle:

```
Import-Package: org.apache.http.osgi.services;version=[4.4,4.5),
```

## Configuring the Apache Proxy Configuration Settings

At minimum, you must provide proxy host information when configuring the Apache Proxy Configuration service.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is `property=<type>["<value1>","<value2>"]`. For example, array of integer `property=I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to `<Predix Machine runtime container location>/configuration/machine`
2. Open the `org.apache.http.proxyconfigurator-[n].config` file and set values for the following properties:

Property	Description	Type	Required	Default Value
<code>proxy.host</code>	The network name or IP address of the proxy host.	String	Yes	
<code>proxy.port</code>	The port used when connecting to the proxy host.	Integer	No	<code>I"8080"</code>
<code>proxy.enabled</code>	Identifies if the proxy settings are enabled. This only serves as a hint to the connecting services set by the admin. It does not specify or reflect the actual server's availability	Boolean	No	<code>B"false"</code>

Property	Description	Type	Required	Default Value
<code>proxy.exceptions</code>	<p>A list of server names or IP addresses that do not need a proxy for access.</p> <p>Acceptable formats include:</p> <ul style="list-style-type: none"> <li>• IPv4 addresses</li> <li>• Full hostnames (For example, myapp.grc-apps.svc.ice.ge.com)</li> <li>• Domains. For example, .ge.com. Domains must begin with a period.</li> </ul>	<p>String Array</p> <p> <b>Note:</b> Array format is <code>property=&lt;type&gt;[ "&lt;value1&gt;", "&lt;value2&gt;" ]</code>. For example, array of integer-property=<code>["1", "2", "3"]</code> A backslash may be used to break up the line for clarity.</p>	No	<code>["localhost", "127.0.0.1", ]</code>
<code>proxy.user</code>	The user name for the proxy server, if required.	String	No	
<code>proxy.password</code>	The password for the proxy server, if required.	String	No	

### Obtaining the Apache Proxy Configuration Service

To obtain the Apache Proxy Configuration service, inject it using declarative services.

The following example shows how to inject the service using declarative services:

```
import org.apache.http.osgi.services.ProxyConfiguration;

...

@Reference
public void setService(ProxyConfiguration proxyConfiguration)
{
    //set service here
}
```

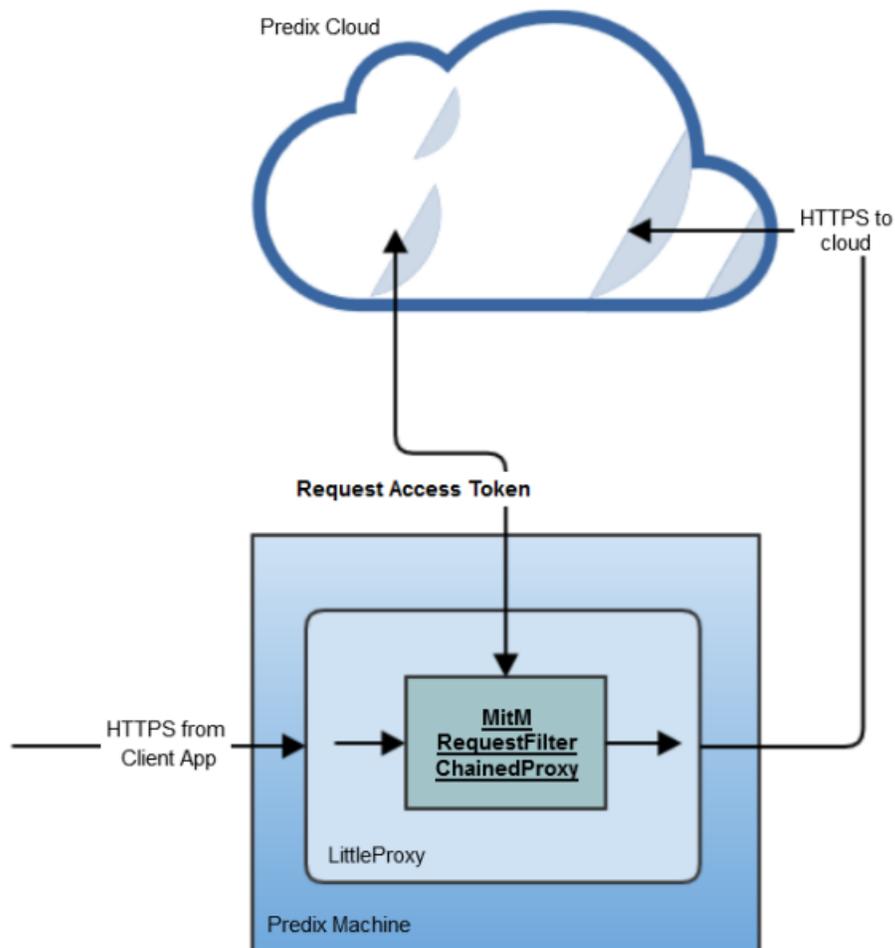
### *Little Proxy*

Applications that run inside the Predix Machine runtime container can communicate with the Predix cloud because Predix Machine adds authentication to the requests going to the cloud. Run the Little Proxy service in Predix Machine so that trusted applications that run outside of the Predix Machine container can communicate with the cloud by proxying requests.

The Little Proxy service facilitates proxy requests using the following workflow:

1. A trusted application that is running on Predix Machine, or outside of the container on the device, sends an HTTPS request to Predix Machine and the Little Proxy service.
2. The Little Proxy service requests an access token from the Predix cloud.
3. The Little Proxy service adds the token to the proxy request and forwards it to the cloud or any upstream proxy.

The following diagram illustrates this workflow:



## Configuring the Little Proxy Service

In an Agent or Debug container type, you can configure the Little Proxy service to specify ports, connection timeout, and scopes.

1. Navigate to `<Predix Machine runtime container location>/configuration/machine`

2. Open the `com.ge.dspmicro.littleproxy.config` file.
3. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.littleproxy.port</code>	Specifies the port that listens for incoming connections.	Integer	"8089"	Yes
<code>com.ge.dspmicro.littleproxy.idleconnectiontimeout</code>	Specifies the number of seconds that lapse until you disconnect idle connections.	Integer	"90"	No
<code>com.ge.dspmicro.littleproxy.connectiontimeout</code>	Specifies the number of milliseconds to wait for an upstream server connection.	Integer	"40000"	No
<code>com.ge.dspmicro.littleproxy.scopes</code>	Specifies scopes for requesting a token.	String Array	(Blank)	No

## WebSocket Client Service

The WebSocket Client service is a component of the Cloud Gateway. It enables applications to establish a WebSocket connection with a WebSocket server endpoint. The Predix Machine implementation exposes the WebSocket Client through the Java WebSocket API (JSR-356). This allows applications running on Predix Machine to be WebSocket-provider agnostic.

### Example Use Cases

The WebSocket Client Service supports the following use cases:

- A service attempts to connect to a `WebSocket` endpoint to send and receive data. The component first gets an instance of the client container, which implements the `WebSocketContainer` interface through a dependency injection. The service then calls a `connectToServer()` method and passes a `ClientEndpoint`-annotated `WebSocket` class to the method.
- A service attempts to connect to a `WebSocket` endpoint to transfer/receive data. The component first gets an instance of the client container, which implements the `WebSocketContainer` interface through dependency injection. It then calls a `connectToServer()` method and passes a `ClientEndpoint`-annotated `WebSocket` object instance to the method. This method allows

the consumer to register a `WebSocket ClientEndpoint` with a configured state as opposed to an uninstantiated class.

## SSL Context

The `WebSocket Client` uses the global `SSLContext` set in the Security Admin configuration file at `<Predix Machine runtime container location>/security` to enable `WebSockets` over `TLS`. After being configured, the client automatically establishes a `TLS` connection when connecting to a `WSS` URI.

## Dependencies

Maven dependencies and an `OSGi` import package are required to consume the service.

- The following Maven dependency is required:

```
<dependency>
  <groupId>javax.websocket</groupId>
  <artifactId>javax.websocket-api</artifactId>
  <version>1.0</version>
</dependency>
```

- The following `OSGi` import is required to consume the bundle:

```
Import-Package: javax.websocket;version="[1.0,2)"
```

## Obtaining the WebSocket Client Service

Inject the `WebSocketContainer` service using declarative services.

The following example shows how to inject the `WebSocketContainer` service:

```
import javax.websocket.WebSocketContainer;
...
@Reference
public void setWebSocketContainer(WebSocketContainer container)
{
    //set client container
}
```

## Using the WebSocket Client Service

The Java Community Process for Java `WebSockets` provides a convenient and provider-agnostic way for consumers to write `WebSocket` code. The API allows you to annotate a class as a `Client/Server` endpoint and annotate respective methods with the commonly used `WebSocket` callback convention of `onOpen/onMessage/onClose`.

Use the following table as a reference for class or method annotations.

Callback Method	Corresponding Event	Example
<code>ClientEndpoint</code>	N/A	<pre> @ClientEndpoint public class EchoClient { ... } </pre>
<code>onClose</code>	A connection has closed.	<pre> @OnClose public void close(Session session,                   CloseReason reason) { } </pre>
<code>onError</code>	An error has occurred with a connection.	<pre> @OnError public void error(Session session,                   Throwable error) { } </pre>
<code>onMessage</code>	A message has arrived.	<pre> @OnMessage public void message (Session session,                     String msg) { } </pre> <p>Additional supported data types:</p> <ul style="list-style-type: none"> <li>• Binary messages</li> <li>• PONG messages, pertaining to the WebSocket connection</li> </ul>
<code>OnOpen</code>	A connection has opened.	<pre> @OnOpen public void open(Session session,                  EndpointConfig conf) { } </pre>

The following example shows an annotated client following the JSR-356 standard. The example contains the minimum `OnOpen` and `OnMessage` callbacks. To use the WebSocket Client service, the consumer must implement an annotated class.

```

@ClientEndpoint
public class BroadcastClientEndpoint
{
    // called when session is opened
    @OnOpen
    public void onOpen(Session session)
    {
        session.getBasicRemote().sendText("Hello!");
    }

    // Receive string data through the open sessions.
    @OnMessage

```

```

public void onMessage(String data, Session session)
{
    System.out.println("Received message " + data);
}
}

```

## Using the WebSocket Client Sample Application

A sample WebSocket application is provided to illustrate how to use the service.

1. Navigate to <SDK installation location>/samples/sample-apps.zip and extract the files.
2. In the sample-apps/sample folder, open the sample-websocketclient application.

 **Note:** See *Building Samples* and *Running Samples in the Predix SDK* for instructions on building and running the sample application.

## Configuring Web Service Security

1. Navigate to <Predix Machine runtime container location>/security.
2. Open the Security Administration configuration file, `com.ge.dspmicro.securityadmin.cfg`.
3. Configure the following property:

Property	Description	Default Value
<code>com.ge.dspmicro.securityadmin.httpsURLConnection.noHostnameVerifier</code>	Sets the default for <code>URLConnection</code> to <code>noHostnameVerifier</code> . This property should only be used for a DEBUG container	false

## Mobile Gateway Services

### WebSocket Server Overview

The WebSocket Server is a component of the Mobile Gateway. It enables applications to host a WebSocket server endpoint. The endpoints can be implemented using the Java Websocket API (JSR-356). This allows applications running on Predix Machine to be WebSocket-provider agnostic.

For an application to host a WebSocket server endpoint, it obtains a reference to the `ServerContainer` interface and passes it a callback handler class annotated with `@ServerEndpoint`. Other WebSocket clients can now connect to this endpoint.

## Dependencies

Maven dependencies and an OSGi import package are required to consume the WebSocket Server.

- The following Maven dependency is required in the `pom.xml` file.

```
<dependency>
  <groupId>javax.websocket</groupId>
  <artifactId>javax.websocket-api</artifactId>
  <version>1.0</version>
</dependency>
```

- The following OSGi import is required to consume the bundle:

```
Import-Package: javax.websocket;version="[1.0,2)"
```

## Setting SSL Context

The WebSocket Client uses the global `SSLContext` set in the Security Admin configurations to enable WebSockets over Transport Layer Security (TLS). Once configured, the server can listen for and successfully establish TLS connections using the WebSocket Secure port configured in the properties. The server can also request/require client certificates to enable two-way TLS.

## WebSocket Server Annotations

The Java Community Process for Java WebSockets allows a convenient and provider-agnostic way for consumers to write WebSocket code. The API allows a developer to annotate a class as a Client or Server endpoint and annotate respective methods with commonly-used WebSocket callback convention: `onOpen/onMessage/onClose`.

The Java WebSocket API for the `ServerContainer` interface is found at: <http://docs.oracle.com/javaee/7/api/javax/websocket/server/ServerContainer.html>.

A sample usage application is found at `<SDK installation location>/samples/sample-apps/sample/sample-websocketserver` (after you unzip the `sample-apps.zip` file).

The following table describes the possible types of annotations used in server (or client) endpoints.

Callback Method	Corresponding Event	Example
<code>ServerEndpoint</code> (annotation type)	N/A	<pre>@ServerEndpoint(value = "/echo") public class EchoServer { ... }</pre>
<code>onClose</code>	A connection has closed.	<pre>@OnClose public void close(Session session,                   CloseReason reason) { }</pre>
<code>onError</code>	A error has occurred with a connection.	<pre>@OnError public void error(Session session,                   Throwable error) { }</pre>
<code>onMessage</code>	A message has arrived.	<pre>@OnMessage public void message (Session session,                     String msg) { }</pre> <p>Additional supported data types:</p> <ul style="list-style-type: none"> <li>• Binary messages</li> </ul> <p> <b>Note:</b> Does not support <code>PongMessage</code>, but can be extended with the use of <code>Encoder</code> and <code>Decoder</code> interfaces.</p>
<code>OnOpen</code>	A connection has opened.	<pre>@OnOpen public void open(Session session,                  EndpointConfig conf) { }</pre>

## Example Annotated Endpoint

The following example shows an annotated server following the JSR-356 standard. The example contains the bare minimum `OnOpen` and `OnMessage` callbacks. To define a `WebSocket` endpoint, the consumer developer must implement an annotated class.

```
@ServerEndpoint(value = "/echo") //URI for this endpoint
public class EchoServer
{
    // called when session is opened
    @OnOpen
    public void onOpen(Session session)
    {
        session.getBasicRemote().sendText("Hello!");
    }

    // Receives text data. Returns whatever is received to the sender
```

```

@OnMessage
public String onMessage(String data, Session session)
{
    return data;
}
}

```

## Limitations

Neither the `ServerContainer` API nor the Jetty API allows for the removal of endpoints once they have already been added. The only way to remove WebSocket endpoints is to restart the WebSocket Server bundle.

### Obtaining the WebSocket Server

There are several ways to obtain a service in OSGi. You can use dependency injection in declarative services. To obtain the service directly from the OSGi service registry, use the same `ServerContainer` interface. After you get the `ServerContainer`, you can call `addEndpoint` with the class that has the endpoint annotations to add a new WebSocket server URL.

Inject the service using declarative services.

Example of how to inject the WebSocket Server container.

```

import javax.websocket.server.ServerContainer;

...

@Reference

public void setServerContainer(ServerContainer server)
{
    //set ServerContainer here
}

```

### Using the WebSocket Server Sample Application

A sample application is provided to show you how to use the WebSocket Server.

1. Navigate to `<SDK installation location>/sample/sample-apps.zip` and extract the files `<SDK installation location>/sample-apps`.
2. Open the `sample-websocketserver` application.

 **Note:** See *Building Samples* and *Running Samples in the Predix SDK* for instructions on building and running the sample application.

## Configuring the WebSocket Server

The properties from the configuration file can be used to set the global default settings for WebSocket connections. These settings will be used by default when opening new connections; however, each individual connection can be configured separately using the API.

When you format values for properties in `.config` files (as opposed to `.cfg` files), use the type character followed by a quoted string representation of value. For example, a Boolean property=`B"true"`. Lowercase type character implies primitives. The type can be omitted for a string. The types and corresponding type characters are shown in the following table:

T = String	I = Integer	L = Long
F = Float	D = Double	X = Byte
S = Short	C = Character	B = Boolean

Array format is property=`<type>["<value1>","<value2>"]`. For example, array of integer property=`I["1", "2", "3"]`. A backslash may be used to break up the line for clarity.

1. Navigate to and open the following file: `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.wsserver.config`.
2. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.wsserver.contextPath</code>	The context path for the WebSocket servlet, also known as a base URL. All endpoints will have this URL prepended to their URL.	string	/	no
<code>com.ge.dspmicro.wsserver.timeout</code>	Idle timeout in milliseconds.	integer	I"3000"	no
<code>com.ge.dspmicro.wsserver.ws</code>	WebSocket server port configuration.  DISABLED - Disable WebSocket server port.  CUSTOM - Use the port specified in <code>customPort</code> property.	enum	"DISABLED"	yes

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.wsserver.ws.customPort</code>	Port value to use with a WebSocket server port configuration set to CUSTOM.	int	I"8183"	no
<code>com.ge.dspmicro.wss</code>	WebSocket Secure server port configuration.  DISABLED - Disable WebSocket server port  CUSTOM - Use the port specified in the <code>customPort</code> property.	enum	"DISABLED"	YES
<code>com.ge.dspmicro.wss.customPort</code>	Port value to use with a WebSocket secure server port configuration set to CUSTOM.	INT	I"8444"	no
<code>com.ge.dspmicro.wsserver.wantClientCertificate</code>	Request client certificate if supported by the client.	boolean	B"false"	no
<code>com.ge.dspmicro.wsserver.needClientCertificate</code>	Require client certificate.	boolean	B"false"	no

# Predix Machine Containerization

## *Containerization Overview*

### *About Containers*

Containers consist of an entire runtime environment: an application along with all of its dependencies, libraries, other binaries, and configuration files needed to run the application, all bundled into one package.

Containers differ from virtualization in that a virtual machine includes an entire operating system, as well as the application. A container is also much smaller in size than a virtual machine, whose own entire operating system, might be several gigabytes in size.

You can now run Predix Machine in a Docker container. For more information about Docker, see <https://www.docker.com>.

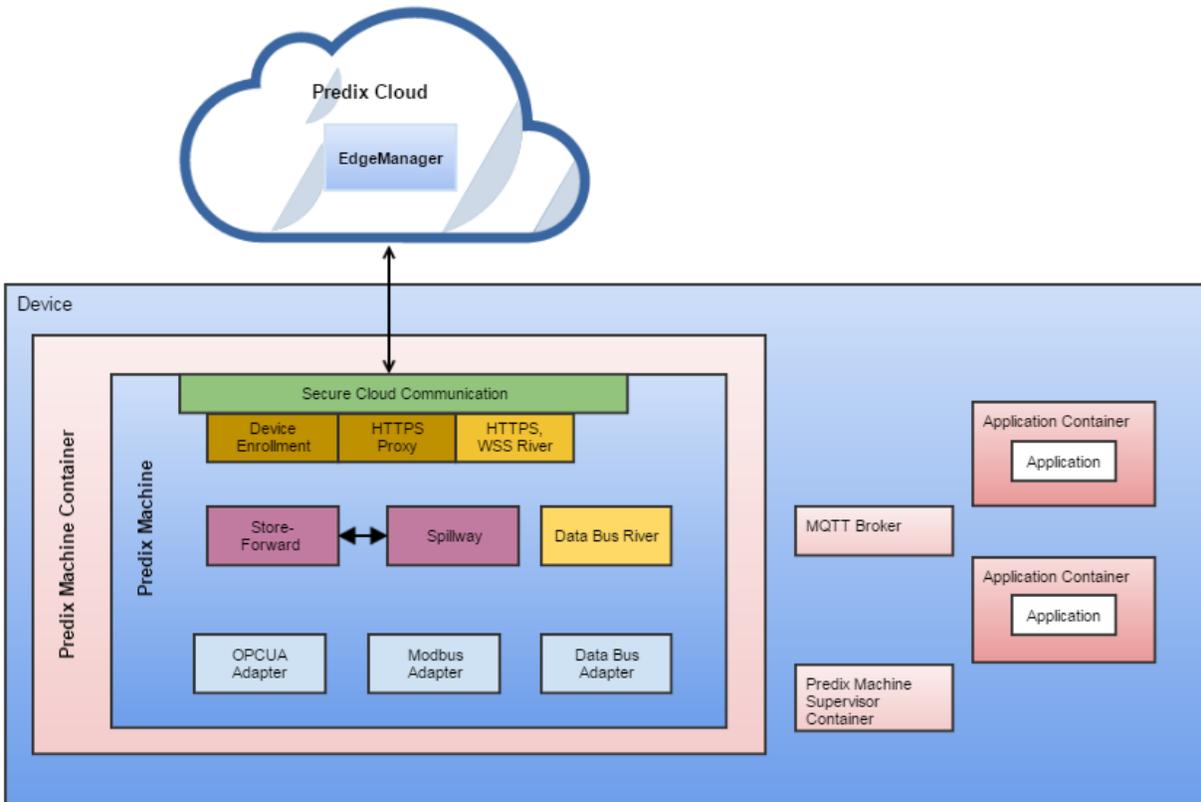
## Predix Machine Containerization Overview

Designing and building a Dockerized solution to create, deploy, and run applications in the Predix Machine container is recommended. This ensures the application will run on other machines regardless of customized settings on those machines. Docker containerization ensures consistency across multiple development and release cycles.

The Predix Machine containerized environment consists of the Predix Machine container as a Docker container, as well as other Docker containers on which applications reside. Communication between the Predix Machine Docker container and other application containers occurs through the Data Bus and the MQTT Broker.

A Predix Machine Supervisor manages containers in the Dockerized environment.

The containerized Predix Machine environment is illustrated in the following image:



To see examples of how data flows in a containerized environment, see [About Data Flow in a Containerized Environment](#) (page ).

For instructions on creating Predix Machine as a Docker image, or converting an existing Predix Machine container into a Docker image, see [Generating a Predix Machine as a Docker Image](#).

[using Command Line Scripts \(page 279\)](#) and [Generating a Docker Image from a Predix Machine Runtime Container \(page 281\)](#).

## *Memory Footprint for Docker Containers*

The following table shows the memory footprint of each of the Predix Machine Docker containers on Ubuntu 14.04.

<b>Container</b>	<b>Memory usage while container is running</b>	<b>Image Size</b>
Bootstrap Loader	~1MB	13.63MB
Predix Machine	~200MB	186.90MB
Mosquitto	~20MB	15.42MB
C++ SDK Sample Subscriber	~1MB	10.12MB
C++ SDK Sample Publisher	~1MB	10.12MB

## *Predix Edge SDK*

### *Predix Edge SDK Overview*

The edge SDK is for developing applications in supported languages that run in Docker containers and communicate with the Docker container-based Predix Machine through the Data or Management Bus.

Predix Machine Management Bus provides command and configuration handling so commands and configurations can go to edge SDK Docker container clients. Data Bus is an MQTT client that facilitates communications between containerized applications within an edge device.

#### **Related concepts**

[Data Bus Overview \(page 255\)](#)

[Management Bus \(page 265\)](#)

### *Downloading the Predix Machine edge SDK*

You can download a Predix Machine edge SDK for C++ without a runtime, for C++ debug development environment, or for Java development.

Use the edge SDK to develop applications in various languages that run in Docker containers and communicate with the Docker container based Predix Machine through the Data Bus.

1. Access Artifactory at the following URL: <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/edgesdk/>
2. To access the C++ edge SDK downloads, click the `predixmachine-edgesdk-cpp` folder.

 **Note:**

Artifacts are provided to support different runtime architectures (ARM and x86). Select the download that makes sense from the following options:

- `armhf-arm7` – Raspberry PI3 Architecture
  - `x86_64` – 64-bit x86 architecture
- a. For the C++ edge SDK without a runtime, choose one of the following:
    - `predixmachine-edgesdk-cpp-alpine-linux-armhf-release-shared/`
    - `predixmachine-edgesdk-cpp-alpine-linux-x86_64-release-shared/`
  - b. For debug development environment for C++ edge SDK, choose one of the following folders:
    - `predixmachine-edgesdk-cpp-rootfs-alpine-armhf/`
    - `predixmachine-edgesdk-cpp-rootfs-alpine-x86_64/`

3. For the Java edge SDK, click the `predixmachine-edgesdk-java/` folder.

## *Predix Machine Edge SDK Directory Structure*

When extracted, the `predixmachine-edgesdk-cpp` file creates the following directory structure:

Directory	Description
include	Contains the following folders: <ul style="list-style-type: none"> <li>• <code>google</code></li> <li>• <code>libcalg-1.0</code></li> <li>• <code>protobuf-c</code></li> <li>• <code>rapidjson</code></li> </ul> Contains the following files: <ul style="list-style-type: none"> <li>• <code>edge_data.pb.h</code></li> <li>• <code>IDatabusClient.h</code></li> <li>• <code>msgq.h</code></li> <li>• <code>PDataValue.h</code></li> </ul>

Directory	Description
share	Contains the following folders: <ul style="list-style-type: none"> <li>• doc</li> <li>• samples</li> </ul>

When extracted, the `predixmachine-edgesdk-java` file creates the following directory structure:

Directory	Description
docs	Contains the <code>apidocs.zip</code> file, which, when extracted contains the Javadoc APIs.
jars	Contains the JAR files for the samples.
license	Contains the license files.
samples	Contains <ul style="list-style-type: none"> <li>• edge-container</li> <li>• sample-container-app</li> <li>• sample-edge-databus</li> <li>• sample-edge-managementbus</li> </ul>

## Accessing the Edge SDK Samples

The edge SDK allows you to develop applications, in various languages, which run in Docker containers and communicate with the Docker container-based Predix Machine through the Data or Management Bus.

1. For the Java samples, navigate to <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/edgesdk/predixmachine-edgesdk-java/17.2.1/>.
  - a. Download and extract `predixmachine-edgesdk-java`.
  - b. Navigate to `<Edge SDK installation location>/samples` and extract the files from `sample-edge-apps.zip`.
  - c. Navigate to the folder where you extracted the files.  
The sample folder includes:

Sample	Description
edge-container	
sample-edge-databus	Standalone Java application that demonstrates how to use the Data Bus service.

Sample	Description
sample-edge-managementbus	Standalone Java application that demonstrates how to use the Management Bus service.

Follow the instructions in the `README.txt` file included with the samples for information about how to build and run the samples.

2. For C++ samples, navigate to <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/edgesdk/predixmachine-edgesdk-cpp/>.

a. You can review the sample applications without installing the entire C++ SDK by downloading and extracting the files from the `getting-started.zip`, which contains all the sample application source and headers for each application.

Read the `sdk-user-documentation.txt` file for information about how to configure and run the samples. The `sdk-user-documentation.txt` file contains the following information:

- An overview of each application and the messaging used
- How to create the required configuration files for the applications
- How to deploy and run a sample Docker container
- A detailed discussion on how to test the applications

The `samples` folder contains:

Sample	Description
SampleDatabusPublisher	Demonstrates how to publish data on certain topics to Data Bus.
SampleDatabusSubscriber	Demonstrates how to subscribe to topics and receive data on those topics from the Data Bus.
SampleEdgeAnalyticsComponentSubscriber	Demonstrates how to subscribe to, and process, Edge Analytics Components using the Management Bus.
SampleEdgeCommandSubscriber	Demonstrates how to subscribe to, and process, Edge Commands using the Management Bus.
SampleEdgeConfigSubscriber	Demonstrates how to subscribe to, and process, Edge Configurations using the Management Bus.
SampleEdgeDataPublisher	Demonstrates how to publish Edge Data on certain topics to the Data Bus.
SampleEdgeDataSubscriber	Demonstrates how to subscribe to topics and receive Edge Data on that topic using the Data Bus.

## *Data Flow in a Containerized Environment*

### *About Data Flow in a Containerized Environment*

The Data Bus allows the flow and exchange of machine data between Predix Machine and other applications in a containerized environment.

The Data Bus facilitates:

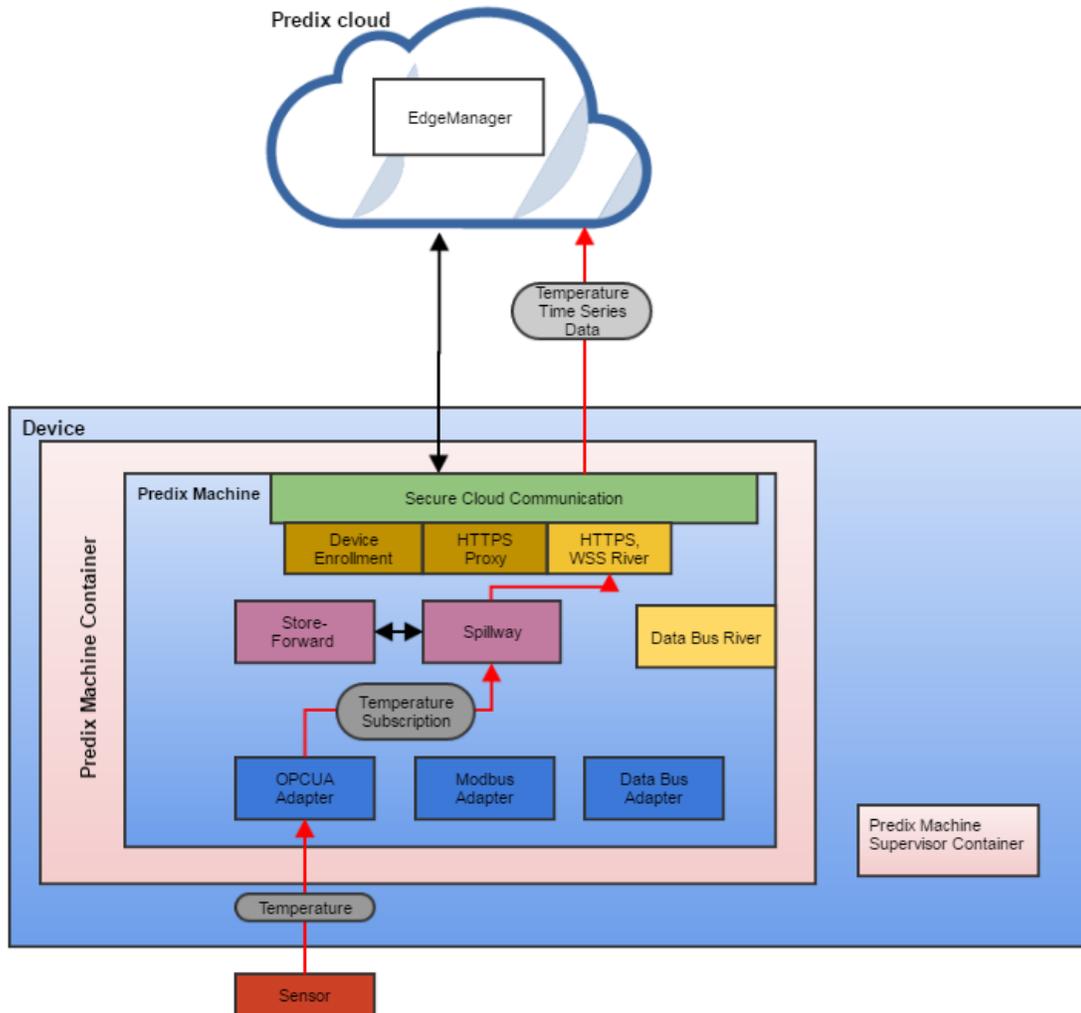
- Publishing machine data that is ingested by machine adapters to other applications through the Data Bus River.
- Ingesting data. Data that is published by other applications for transmission to the cloud through the HTTPS, WebSocket, or other rivers is ingested through the Data Bus Adapter.
- Exchanging of data between applications using the Databus API. The data on the wire is in the serialized PDataValue format used by Predix Machine.

### **Data Flow Use Cases**

Examples of how machine data can be transmitted from a sensor to the Predix Machine container, through the MQTT broker to containerized applications, and ultimately to the Predix cloud are listed below:

- In a typical data flow, Predix Machine data that is ingested by the OPC-UA adapter is transmitted to the Time Series service through the WebSocket River.

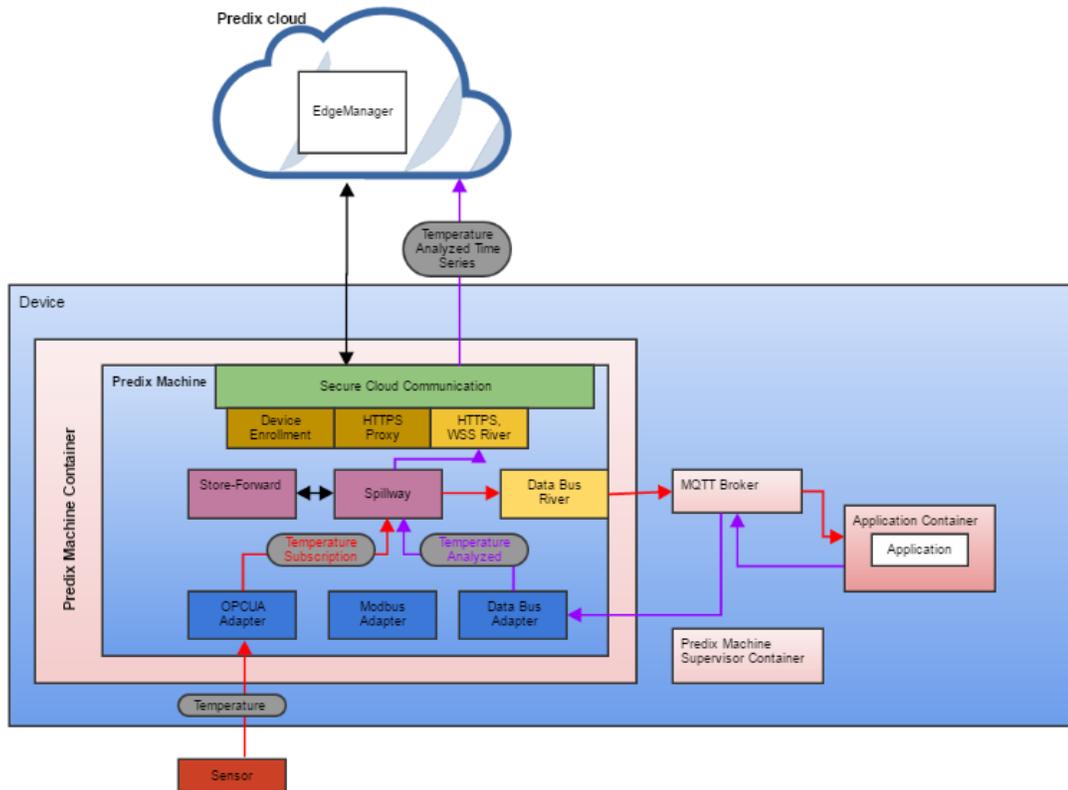
This flow is illustrated in the following image:



• In a containerized environment:

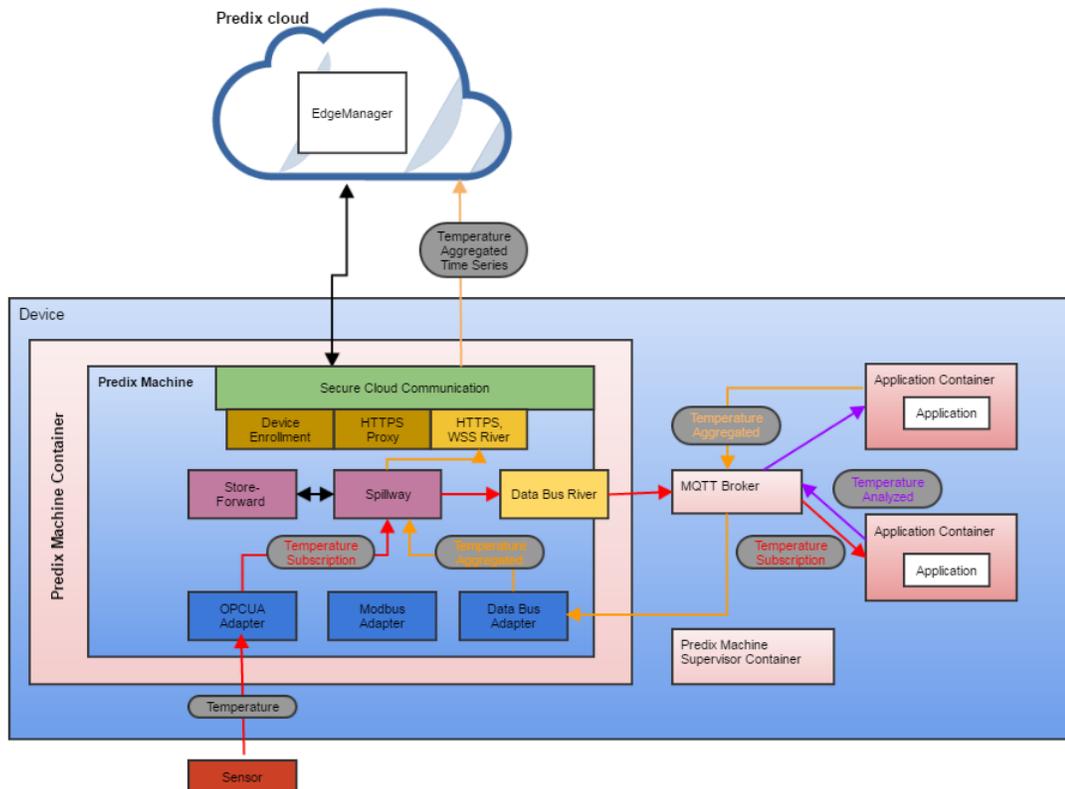
1. Temperature data that is ingested by the OPC-UA adapter is published on the topic `Temperature_Subscription` through the Data Bus River.
2. The `Temperature_Subscription` topic is consumed by an application and processed. The processed data is published as `Temperature_Analyzed` by the application.
3. The Data Bus Adapter consumes the `Temperature_Analyzed` and transmits it to the Time Series service through the WebSocket River.

This flow is illustrated in the following image:



- In another containerized environment example:
  1. Temperature data that is ingested by the OPC-UA Adapter is published on the topic `Temperature_Subscription` through the Data Bus River.
  2. `Temperature_Subscription` is consumed by an application and processed. The processed data is published as `Temperature_Analyzed` by the application.
  3. Another application consumes `Temperature_Analyzed`, performs further analysis, and publishes the results as `Temperature_Aggregated`.
  4. The Data Bus Adapter consumes `Temperature_Aggregated` and transmits it to the Time Series service through the WebSocket River.

This flow is illustrated in the following image:



## Data Bus Overview

Data Bus is an MQTT client that facilitates communications between containerized applications within an edge device.

The applications, which can be written in Java or C++, communicate through a unified set of topics and data schema.

Data Bus supports the following types of data.

### EdgeData

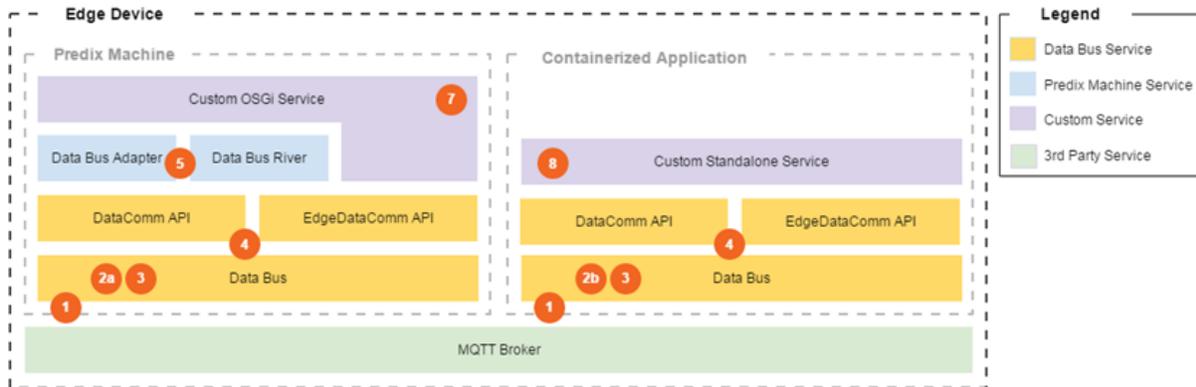
A data structure defined with Protobuf that extends the functionality of `PDataValue` and occupies less space when serialized.

GE recommends that you use `EdgeData` for development, especially if you are developing for the cloud. `EdgeData` provides features not supported by `PDataValue`, such as binary data and array type.

### PDataValue/Data

Data collected from sensors. See [Data Communication \(page 256\)](#) for more information.

## Data Bus Design



1. Data Bus is an MQTT client.
2. Packaging:
  - a. Data Bus can run as OSGi services in a Predix Machine container.
  - b. Data Bus can also be packaged into Java/C++ libraries for standalone application development.
3. To configure Data Bus, provide connection information for the MQTT broker.
4. Data Bus provides three services—DataBus, DataComm API, and EdgeDataComm API.
5. Data Bus Machine Adapter and Data Bus River are Predix Machine services built on top of the two DataComm APIs.
  - a. Data Bus Machine Adapter - Subscribes to data on Data Bus. See [Data Bus Machine Adapter \(page 262\)](#) for more information.
  - b. Data Bus River - Publishes data to Data Bus. See [Data Bus River \(page 263\)](#) for more information.
6. Data Routing:
  - a. You can route data from various Machine adapters to Data Bus River, or from Data Bus Machine Adapter to other rivers by configuring spillways, the same way data is routed from other adapters to rivers.
  - b. Spillway should not be configured to route data from Data Bus Adapter to Data Bus River. See [Hoover Service \(page 139\)](#) for more information.
7. Custom OSGi service accesses Data Bus through DataComm API, EdgeDataComm APIs, or Data Bus Adapter and River.
8. Standalone application accesses Data Bus through DataComm or EdgeDataComm APIs.

## Data Communication

The EdgeDataComm and DataComm API allows communication and transfer of data between Predix Machine and Dockerized applications.

## Data

Data can be:

- Ingress data collected from sensor streams through the Machine Gateway.
- Data that flows through egress channels like HTTP, MQTT, and WebSocket.
- Application-to-application-data.

## Topics

All data is communicated within a root topic. Note the following information and limitations regarding the `topicTag`:

API	Root topic	Topics
edgeDataComm	<code>edgeData/&lt;topicTag&gt;</code>	<ul style="list-style-type: none"> <li>• <code>topicTag</code> is a branch that can be used for filtering.</li> <li>• <code>topicTag</code> can only contain the following characters: A-Z, a-z, 0-9, and <code>_</code>.</li> </ul>
IDataComm	<code>data/&lt;topicTag&gt;</code>	<ul style="list-style-type: none"> <li>• <code>topicTag</code> is a branch that can be used for filtering.</li> <li>• <code>topicTag</code> can only contain the following characters: A-Z, a-z, 0-9, and <code>_</code>.</li> </ul>

## Use Cases

Ingress and egress data flow examples are provided.

- Ingress Data Flow Example
  1. An analytic application subscribes to a topic named `data/Temperature_Subscription`.
  2. Predix Machine is configured with a Spillway that forwards data from the `Temperature_Subscription` topic to the Data Bus River.
  3. Predix Machine receives data through the Modbus Adapter.
  4. The Spillways sends the data on to the Data Bus River.
  5. The Data Bus River publishes data to the `data/Temperature_Subscription` topic.
- Egress Data Flow Example
  1. Predix Machine is configured with a Spillway that forwards data from the Data Bus Adapter to the WebSocket River service.
  2. The Data Bus Adapter subscribes to the `data/Temperature_Analyzed` topic.
  3. An analytic application publishes data to the `data/Temperature_Analyzed` topic.
  4. The Data Bus Adapter receives the data.

5. The Spillway forwards the data to the WebSocket River.

## *Obtaining the Data Bus Service*

1. To obtain the Edge API, inject the `IEdgeDataComm` service using declarative services.

```
private IEdgeDataComm edgeDataComm;

/**
 * Dependency injection of IEdgeDataComm.
 *
 * @param edgeDataComm Communication handler for EdgeData messages.
 */
@Reference
public void setEdgeDataComm(IEdgeDataComm edgeDataComm)
{
    this.edgeDataComm = edgeDataComm;
}

/**
 * Clears dependency injection of IEdgeDataComm.
 *
 * @param edgeDataComm Communication handler for EdgeData messages.
 */
public void unsetEdgeDataComm(@SuppressWarnings("hiding") IEdgeDataComm
    edgeDataComm)
{
    this.edgeDataComm = null;
}
```

2. To obtain the Data API for OSGi applications, inject the `IDataComm` service using declarative services.

Example of how to inject the `IDataComm` service:

```
private IDataComm dataComm;

/**
 * Dependency injection of IDataComm.
 *
 * @param dataComm Communication handler for data plane messages.
 */
@Reference
public void setDataComm(IDataComm dataComm)
{
    this.dataComm = dataComm;
}

/**
 * Clears dependency injection of IDataComm.
 *
 */
```

```

 * @param dataComm Communication handler for data plane messages.
 */
public void unsetDataComm(@SuppressWarnings("hiding") IDataComm
    dataComm)
{
    this.dataComm = null;
}

```

## *Data Bus Consumer Configuration*

The Data Bus service requires some Maven dependencies and OSGI imports.

### **Maven Dependencies**

The following Maven dependencies are required to consume this service:

For the Databus API:

```

<!-- Databus API -->
<dependency>
    <groupId>com.ge.dspmicro</groupId>
    <artifactId>databus-api</artifactId>
    <version>{Predix_Machine_version}</version>
</dependency>

```

For the EdgeDataComm API and EdgeData:

```

<!-- If using EdgeDataComm API and EdgeData -->
<dependency>
    <groupId>com.ge.predixmachine</groupId>
    <artifactId>protobuf-models</artifactId>
    <version>{Predix_Machine_version}</version>
</dependency>
<dependency>
    <groupId>com.google.protobuf</groupId>
    <artifactId>protobuf-java</artifactId>
    <version>3.0.0</version>
</dependency>

```

For the DataComm API and PDataValue:

```

<!-- If using DataComm API and PDataValue -->
<dependency>
    <groupId>com.ge.dspmicro</groupId>
    <artifactId>machinegateway-api</artifactId>
    <version>{Predix_Machine_version}</version>
</dependency>
<dependency>
    <groupId>com.ge.dspmicro</groupId>
    <artifactId>device-common</artifactId>

```

```
<version>{Predix_Machine_version}</version>
</dependency>
```

## OSGI Imports

The following OSGI imports are required for consuming the bundle:

```
Import-Packages:com.ge.dspmicro.databus.api.common;version="[2.0,3.0)",
com.ge.dspmicro.databus.api.data;version="[2.0,3.0)",
com.ge.dspmicro.machinegateway.types;version="[1.4,2.0)",
com.ge.predixmachine.datamodel.datacomm;version="[1.0,2.0)",
com.google.protobuf;version="[3.0,4.0)",
com.google.protobuf.util;version="[3.0,4.0)",
```

## Configuring Data Bus

For OSGI applications, you can configure the Data Bus to specify the ID of the Docker application and the URI of the MQTT broker that handles messaging within the device.

Install Mosquitto (an MQTT broker).

1. Navigate to <Predix Machine runtime container location>/configuration/machine.
2. Open the com.ge.dspmicro.databus.mqtt.config file.
3. Set values for the following properties:

Property	Description	Type	Format	Default Value	Required
com.ge.dspmicro.databus.app.id	Unique identifier for the Docker application.	String	Alphanumeric characters only. Will be converted to lowercase.	None	Yes
com.ge.dspmicro.databus.mqtt.broker.uri	URI of the MQTT broker that handles messaging within the device.	String	tcp://hostname:port  <b>Important:</b> The MQTT Broker container uses port 1883 on localhost. If you set any another value here, you cannot register with the broker.	None	Yes
com.ge.dspmicro.databus.mqtt.broker.alias	Alias for the Data Bus to the MQTT broker.	String	N/A	None	Generated

Property	Description	Type	Format	Default Value	Required
com.ge.dspmicro.databus.mqtt.application.id	Unique identifier for the Docker application.	String	Alphanumeric characters only.	None	Yes
com.ge.dspmicro.databus.mqtt.connector.url	URL of the MQTT broker that handles messaging within the device.	String	tcp://hostname:port  <b>Important:</b> The MQTT Broker container uses port 1883 on localhost. If you set any other value here, you cannot register with the broker.	None	Yes

## Using the Data Bus Sample Applications

Java and C++ sample applications are provided to illustrate how to use the service. Java applications include one that uses an OSGi framework, and one standalone that does not. The C++ sample application is also a standalone application that does not use the OSGi framework.

1. To access the OSGi Java Sample:
  - a. Navigate to <SDK installation location>/samples/sample-apps.zip and extract the files.
  - b. In the sample-apps/samples folder, open the sample-databus application.
2. To access the standalone Java sample:
  - a. Navigate to <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/edgesdk/predixmachine-edgesdk-java/17.2.1/> and extract the files.
  - b. In the samples/sample-edge-apps/sample/sample-edge-databus folder, open the sample-edge-databus application.
3. To access the C++ sample:
  - a. Navigate to <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/edgesdk/predixmachine-edgesdk-cpp/> and download the package with the appropriate architecture.
  - b. In the samples/sample-edge-apps folder, open the sample-edge-databus application.

## Using the Data C++ APIs

Review the C++ APIs to understand how to implement Data communication in C++ in the Data Bus.

1. Navigate to <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/edgesdk/predixmachine-edgesdk-cpp/> folder and extract files.

2. Open the doc folder to review C++ APIs.

## *Using the Data Communication Java APIs*

Review the `com.ge.dspmicro.databus.api` to understand how to implement Data communication in Java in the Data Bus.

1. Navigate to <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/edgesdk/predixmachine-edgesdk-java/17.2.1/docs/apidocs/index.html>
2. Open the `com.ge.dspmicro.databus.api`.

## *Data Bus Machine Adapter*

The Data Bus Machine Adapter uses the existing Machine Gateway Adapter infrastructure to get data from the Data Bus. The Machine Gateway Adapter interface allows the Data Bus Machine adapter to connect to a river service through a Spillway.

Using this adapter, Dockerized applications can process data and push the data back to the WebSocket River for uploading to the cloud. For example, if a Docker application receives and processes OPC-UA data, it processes the data, and then puts it back on the Data Bus in a topic. The Data Bus Adapter picks up the data and makes it available to subscribers. A Spillway will listen to a subscription of this data and use a River to send it to the Time Series service to the cloud.

**⚠ Important:** While the Data Bus Machine Adapter and the Data Bus River are both components of the Data Bus, the two should not be connected together through a spillway. This would result in an infinite loop.

## **Dependencies**

Maven dependencies and an OSGi import package are required to consume the service:

- The following Maven dependency is required:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>machineadapter-databus</artifactId>
  <version>17.1.0</version>
</dependency>
```

- The following OSGi import is required in the consuming bundle, but only if you are using the Machine Adapter Data Bus constants:

```
Import-Package:
  com.ge.dspmicro.machineadapter.databus;version="[1.0,2)"
```

## Configuring the Data Bus Machine Adapter

The Data Bus Machine adapter is configured as a factory service component. A configuration file of service ID `com.ge.dspmicro.machineadapter.databus-[n].config` must be present, where `n` is a unique integer per instance.

1. Navigate to, and open, `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.databus-[n].config`.
2. Set values for the following properties.

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.machineadapter.databusname</code>	A unique name that identifies this adapter instance.	String	n/a	Yes
<code>com.ge.dspmicro.machineadapter.databusdescription</code>	A human-readable description of the adapter.	String	n/a	No
<code>com.ge.dspmicro.machineadapter.databustopics</code>	An array of strings that represent the databus topics for which to create subscriptions.	String array	n/a	Yes

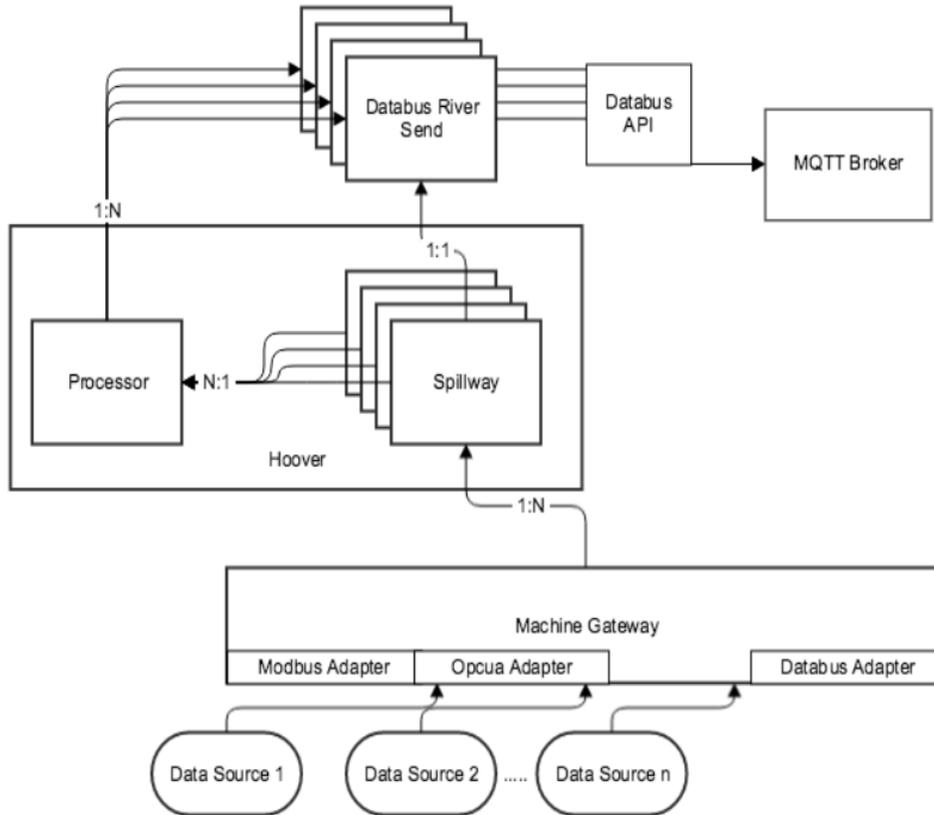
## Data Bus River

You can transmit data from Predix Machine to other applications on the same device using the Data Bus River. For example, send data that is ingested by any of your machine adapters to other applications, such as analytics, running on the same device.

### Functionality and Data Flow

The Data Bus River service receives data from machine adapters and publishes it as a serialized list of `PDataValues` on the data bus using the Data Bus API. The subscription name for the data is the topic for publication.

The data flow from the data source through the adapter to the Hoover spillway and processor to the Data Bus API is shown in the following figure.



## Dependencies

Maven dependencies and an OSGi import are required to consume the Data Bus River.

- The following Maven dependencies are required:

```
<dependency>
  <groupId>com.ge.dspmicro</groupId>
  <artifactId>databusriver-send</artifactId>
  <version>17.2.x</version>
</dependency>
```

- The following OSGi imports are required in the consuming bundle:

```
Import-Package: com.ge.dspmicro.databusriver.send.api;version="[1.0,2)"
```

## *Obtaining the Data Bus River Service*

Inject the service using declarative services.  
An example on how to inject the service follows:

```
import com.ge.dspmicro.mqttriver.send.api.IMqttRiverSend;
...
@Reference
public void setService(IDatabusRiverSend service)
{
    //set service here
}
```

## Configuring the Data Bus River

1. Navigate to <Predix Machine runtime container location>/configuration/machine.
2. Open the `com.ge.dspmicro.databusriver.send.config` file.
3. Set values for the following properties:

Property	Description	Type	Default Value	Required
<code>com.ge.dspmicro.databusriver.send.river.name</code>	A human-readable name for the river. This is used to link the Data Bus River to other services, such as the Spillway.	String	n/a	Yes
<code>com.ge.dspmicro.databusriver.send.defaultTopic</code>	A UTF-8 string. This is used only if the properties passed to the Data Bus River do not include the <code>subscription</code> property.	String	n/a	No

## Management Bus

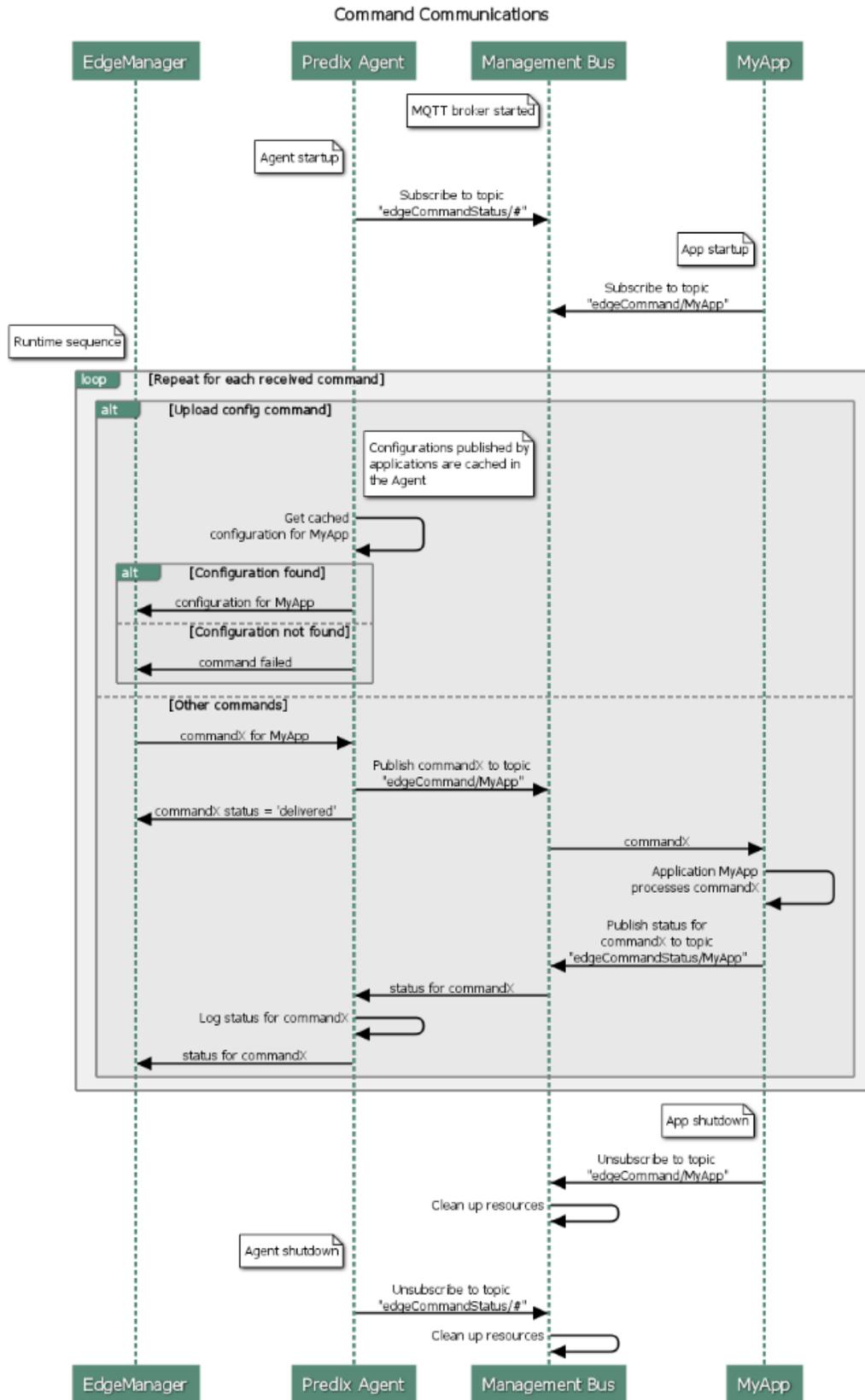
Use Management Bus with containerized applications to leverage the power of Predix Machine to communicate administrative events with Edge Manager,

Predix Machine Management Bus service allows you to develop an application that runs in its own container. You can add your own business logic and use any language for your application (as long

as you follow the messaging protocol). This allows your custom applications to exchange commands and configurations with Predix Machine, without restricting you to using a particular language.

## *Command Communications*

The following diagram shows the execution flow for command communications.



## Startup

The following describes the command communication flow for startup:

1. Predix Agent starts up and subscribes to command statuses published by all applications (topic `edgeCommandStatus/<appid>`).
2. MyApp starts up and subscribes to relevant commands (topic `edgeCommand/<appId>`).

## New Commands Pushed from Edge Manager

The following describes the command communication flow for new commands pushed from Predix Edge Manager:

1. Agent receives an Upload Configuration command for MyApp from Predix Edge Manager.
  - a. The Agent checks cached configurations for MyApp configuration and uploads it to Predix Edge Manager.
2. Other commands:
  - a. Predix Agent publishes the command to the `edgeCommand/<appId>` topic.
  - b. Predix Agent responds to Predix Edge Manager with a status of "delivered."
  - c. MyApp receives and processes the command.
  - d. MyApp publishes the command status to the `edgeCommandStatus/<appId>` topic.
  - e. Predix Agent receives the command status and logs it.
3. Predix Agent submits the command status to Predix Edge Manager.

## Shutdown

The following shows the command communication flow for shutdown:

1. MyApp unsubscribes from edge commands.
2. Management Bus performs cleanup, as necessary.
3. Predix Agent unsubscribes from all edge command statuses.
4. Management Bus performs cleanup, as necessary.

 **Note:** Commands are uni-directional, which means they go only from Predix Agent to applications. Commands do not flow application-to-agent or application-to-application.

Command statuses are uni-directional, which means commands go only from applications to Predix Agent. Command statuses do not flow agent-to-application or application-to-application.

## Topics

Topic Name	Payload	Sender	Recipient
edgeCommand/<appId>	Command for specific application	Predix Agent	applications
edgeCommandStatus/<appId>	Command execution status	applications	Predix Agent

The following shows the payload format for the command and command status:

```

/*
 * Copyright (c) 2016 General Electric Company. All rights reserved.
 *
 * The copyright to the computer software herein is the property of
 * General Electric Company. The software may be used and/or copied only
 * with the written permission of General Electric Company or in accordance
 * with the terms and conditions stipulated in the agreement/contract
 * under which the software has been supplied.
 */

/*
 * Describes the structures of Predix Machine Data downloaded from the
 * cloud.
 */

syntax = "proto3";

package com.ge.predixmachine.protobuf;

import "google/protobuf/timestamp.proto";
import "task_info.proto";

option java_multiple_files = true;
option java_package = "com.ge.predixmachine.datamodel.commandcomm";
option java_generate_equals_and_hash = true;

//
// Represents the Command that the edge device will process
//
message EdgeCommand
{
    reserved 3;
    reserved "app_name";

    // The Command ID
    string id = 1;

    // The Command Handler Type
    string command_handler_type = 2;

    // The Command Name
    string command = 4;
}

```

```

// URL to upload output
string url = 5;

// The parameters for this command
map<string, string> params = 6;
}

//
// Represents the Command Status for a previous executed task from Cloud
// Edge Gateway Service
// Ensure that the TaskStatus is set. If not set, the default TaskStatus
// will be used,
// which may cause misinterpretation of the status.
//
message CommandStatus
{
// The ID of the Command sent down from the cloud
string task_id = 1;

// A Status defined in the TaskStatus
TaskStatus status = 2;

// A Status Message
string status_message = 3;

// A Detailed Status Message
string status_detailed_message = 4;

// An output for the Edge Command
string output = 5;

// The command start time
google.protobuf.Timestamp start_time = 6;

// The command end time
google.protobuf.Timestamp end_time = 7;
}

```

## Using the Command Communication APIs

Review the Command Communication Javadoc APIs (`CommandComm` and `CommandListener`) to understand how to implement command communication.

Navigate to and open the following API: `<Predix_Machine_SDK installation location>/docs/apidocs/index.html/com.ge.dspmicro.managementbus.api.command`.

## *Configuration Communications*

## Startup

The following describes the configuration communications flow for startup:

1. Predix Agent starts up and subscribes to configuration statuses published by all applications (topic `edgeConfigStatus/<appid>`).
2. MyApp starts up and subscribes to relevant configurations (topic `edgeConfig/<appId>`).
3. MyApp publishes the startup configuration to Predix Agent (topic `edgeConfigStatus/<appId>`).
4. Predix Agent receives the startup configuration and caches it for later use.

## New configurations Pushed from Edge Manager

The following describes the configuration communication flow for new configurations pushed from Predix Edge Manager:

1. Predix Agent receives a new configuration from Predix Edge Manager.
2. Predix Agent publishes the new configuration to topic `edgeConfig/<appId>`.
3. MyApp receives a new configuration.
4. MyApp updates its configuration and restarts as necessary.
5. MyApp publishes the updated configuration to topic `edgeConfigStatus/<appId>`.
6. Predix Agent receives the configurations and updates its cache.

## Shutdown

The following shows the configuration communication flow for shutdown:

1. MyApp unsubscribes from edge configurations.
2. Management Bus performs cleanup, as necessary.
3. Predix Agent unsubscribes from all edge configuration statuses.
4. Management Bus performs cleanup, as necessary.

## Topics

Topic Name	Payload	Sender	Recipient
<code>edgeConfig/&lt;appId&gt;</code>	New configurations for specific application	Predix Agent	applications
<code>edgeConfigStatus/&lt;appId&gt;</code>	Configuration status and latest configuration used by application	applications	Predix Agent

The following shows the payload format for configuration and configuration status communications:

```
/*
```

```

* Copyright (c) 2016 General Electric Company. All rights reserved.
*
* The copyright to the computer software herein is the property of
* General Electric Company. The software may be used and/or copied only
* with the written permission of General Electric Company or in accordance
* with the terms and conditions stipulated in the agreement/contract
* under which the software has been supplied.
*/

/*
* Describes the structures used in communication of Predix Machine
Configurations.
*/

syntax = "proto3";

package com.ge.predixmachine.protobuf;

import "edge_package.proto";

option java_multiple_files = true;
option java_package = "com.ge.predixmachine.datamodel.configcomm";
option java_generate_equals_and_hash = true;

//
// Represents the configuration of the application.
// This is typically new configuration deployed from Edge Manager for the
// application.
//
message EdgeConfig
{
    // Identifier for the configuration update task.
    // This is used to tie the task to the ConfigStatus.
    string task_id = 1;

    // Binary archive of application configuration.
    bytes configuration = 5;
}

//
// Represents the status of a configuration update operation.
// This is typically published by containerized applications to acknowledge
// a configuration update operation
// and notify Predix Machine agent with its latest configuration.
//
message ConfigStatus
{
    // Status of configuration update operation.
    PackageStatus package_status = 1;

    // Latest configuration of the application.
    // This may or may not be the same as the configuration received by the
    // application.

```

```
bytes configuration = 5;
}
```

## Using the Configuration Communication APIs

Review the Command Communication Javadoc APIs (`CommandComm` and `CommandListener`) to understand how to implement command communication.

Navigate to and open the following API: `<Predix_Machine_SDK_installation_location>/docs/apidocs/index.html/com.ge.dspmicro.managementbus.api.command`.

## *Accessing the Management Bus Sample Application*

A standalone Java application is provided to illustrate how to use the Management Bus service.

1. Navigate to <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/edgesdk/predixmachine-edgesdk-java/17.2.1/> and extract the files.
2. Navigate to `samples/sample-edge-apps.zip` and extract the files.
3. The Java application sample is in the `sample-edge-managementbus` folder.

## *Download and Run Containerization Software*

### *Downloading Docker Images*

Download the Docker container components that allow you to use Predix Machine in a containerization environment.

To use Predix Machine in a complete containerization environment, you must download and install the following Docker container components, based on your environment:

- Predix Machine Agent for Docker/containerized environment
- Bootstrap Loader
- MQTT Broker

**!** **Important:** Do not download any of the images from `https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/platforms`. These images are not supported on Predix Machine 17.1.x and above.

Access Artifactory at the following URL <https://artifactory.predix.io/artifactory/webapp/#/artifacts/browse/tree/General/PREDIX-EXT/predix-machine-package/docker-images> and login with your Predix account credentials.

 **Note:** Be sure to download version 1.3.0 of the base docker image.

Click the folder to download the corresponding container.

Folder	Description
<ul style="list-style-type: none"> <li>• <code>predixmachine-agent-armhf-&lt;jre_version&gt;/</code>, or</li> <li>• <code>predixmachine-agent-x86_64-&lt;jre_version&gt;/</code></li> </ul>	Predix Machine Agent container for the containerization environment.
<ul style="list-style-type: none"> <li>• <code>predixmachine-agent-armhf-&lt;jre_version&gt;-openvpn/</code></li> <li>• <code>predixmachine-agent-x86_64-&lt;jre_version&gt;-openvpn/</code></li> </ul>	Predix Machine Agent container that includes OpenVPN.
<ul style="list-style-type: none"> <li>• <code>predixmachine-agent-debug-armhf-&lt;jre_version&gt;/</code></li> <li>• <code>predixmachine-agent-debug-x86_64-&lt;jre_version&gt;/</code></li> </ul>	Predix Machine Agent Debug container for the containerization environment.
<ul style="list-style-type: none"> <li>• <code>predixmachine-agent-debug-armhf-&lt;jre_version&gt;-openvpn/</code></li> <li>• <code>predixmachine-agent-debug-x86_64-&lt;jre_version&gt;-openvpn/</code></li> </ul>	Predix Machine Agent Debug container that includes OpenVPN.
<ul style="list-style-type: none"> <li>• <code>predixmachine-bootstrap-alpine-armhf/</code></li> <li>• <code>predixmachine-bootstrap-alpine-x86_64/</code></li> </ul>	Bootstrap container for loading containers into the Docker environment.
<ul style="list-style-type: none"> <li>• <code>predixmachine-alpine-armhf/</code></li> <li>• <code>predixmachine-alpine-x86_64/</code></li> </ul>	Base Docker images for Linux Alpine to use with C++ edge SDK.
<ul style="list-style-type: none"> <li>• <code>predixmachine-openjdk-jre-armhf/</code></li> <li>• <code>predixmachine-openjdk-jre-x86_64/</code></li> </ul>	Base Docker images for Linux Alpine with OpenJDK to use with the Java edge SDK.
<ul style="list-style-type: none"> <li>• <code>predixmachine-mosquitto-armhf/</code></li> <li>• <code>predixmachine-mosquitto-x86_64/</code></li> </ul>	Docker image for the MQTT broker Mosquitto.

Folder	Description
<ul style="list-style-type: none"> <li>• <code>predixmachine-edgesdk-cpp-base-alpine-armhf/</code></li> <li>• <code>predixmachine-edgesdk-cpp-base-alpine-x86_64/</code></li> </ul>	Base Data Bus image for C++.

**Related concepts**

Accessing Artifactory Downloads ([page](#) )

## Downloading the Docker Quickstart App

The Docker quickstart app provides an application that is running within a Docker container.

1. Access Artifactory at the following URL <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/docker-quickstart/> and login with your Predix account credentials.
2. Download the `docker-quickstart-<version>.zip` file.
3. Unzip and extract all of the files from the ZIP file.  
A `start-bootstrap` folder is created in the location where you extracted the files.
4. Follow the instructions in the `<location of docker-quickstart>/start-bootstrap/README.txt` file to run the Dockerized app.

**Related concepts**

Accessing Artifactory Downloads ([page](#) )

## Preparing a Mac OS Device to Run the Containerized Predix Machine

1. Download Docker for Mac 1.12.x or later.
2. Download the architecture-appropriate variants of the following files:
  - Predix Machine bootstrap docker image: `bootstrap-alpine-<architecture>-<version>.tar.gz`.
  - For the Predix Machine Agent Docker image: `predixmachine-agent-<architecture>-<version>.tar.gz`.

 **Note:** If you have customized Predix Machine, you can replace this with your own containerized version that you generated with the `GenerateContainers` and `DockerizeContainer` scripts.

  - Mosquito Messaging Broker Docker image: `predimachine-mosquito-<architecture>-<version>.tar.gz`

- Docker images for any other applications, such as ones created with the Predix Machine edge SDK, provided as Docker images archived as tar.gz.

### 3. Load the bootstrap image into Docker's cache.

```
docker load -i predixmachine-bootstrap-alpine-<architecture>-<version>.tar.gz
```

### 4. Create a directory to mount to the bootstrap container.

This directory will be used by the bootstrap to manage the directory paths of the containers it manages.

### 5. Create and start the bootstrap container, which is located in <predixmachinesdk-installation\_location>/utilities/containers/bootstrap/:

```
bash start_bootstrap.sh -d <bootstrap mount directory>
```

**Note:** Containers do not start in host mode, as Docker for Mac does not support this as of this release. The Mosquitto container will expose port 1883 and the Predix Machine container will expose port 8443 through port binding. Only connections from within the host machine (127.0.0.1) will be allowed to connect to these ports.

### 6. Copy the docker image archives for Mosquitto to the directory monitored by bootstrap.

```
cp predixmachine-mosquitto-x86_64-<version>.tar.gz <bootstrap mount directory>
```

Bootstrap will load the images and start the MQTT container.

**Note:** If you use Finder, you must copy the file. Docker on Mac does not register a "move" event of the file.

### 7. Verify that Mosquitto images were loaded and the MQTT container started:

```
docker images
docker ps
```

The output of the Docker commands should, after some delay, list the Bootstrap and the MQTT container.

### 8. Go into the MQTT Docker container:

```
docker exec -it MQTT /bin/sh
```

### 9. Edit the /etc/mosquitto/mosquitto.conf file in the MQTT container to allow connections from other containers:

```
vi /etc/mosquitto/mosquitto.conf
```

- a. Comment out the last line of the file, "bind\_address localhost", by adding a # at the beginning of the line.
- b. Save and exit the file.
- c. Enter the following command to exit from the Docker container:

```
exit
```

This is necessary because Docker for Mac does not yet support host mode. This means connections from other containers will not be seen as incoming from 'localhost' to the MQTT container. Instead, connections to the MQTT container are only allowed from within the host machine (i.e. the host itself or other Docker containers running on the host).

10. Copy the docker image archives for Predix Machine and any other custom applications to the directory monitored by bootstrap.

```
cp predixmachine-agent-x86_64-<version>.tar.gz <bootstrap mount
directory>
```

Bootstrap will load the images and start containers. The process may take a little while, including time for the polling interval, time to load images, and start containers.

 **Note:** If you use finder, you must copy the file. Docker on Mac does not register a "move" event of the file.

11. Verify that all images were loaded and the containers have started:

```
docker images
docker ps
```

12. For Predix Machine to communicate using MQTT, the following default configuration files must be edited if used:

- com.ge.dspmicro.managementbus.mqtt.config
- com.ge.dspmicro.databus.mqtt.config

- a. Since the containers are not running in host mode, the 'localhost' value in the MQTT connection URL must be replaced with the IP Address of the MQTT container:

```
docker inspect MQTT | grep IPAddress
```

 **Note:** Enter the command as shown above. It is case-sensitive and must be properly capitalized.

- b. Copy the IP Address and replace the 'localhost' values in the configuration files. This can be done by accessing the <bootstrap mount directory>/containers/Predix\_Machine/configuration/machine/ folder from the host or by accessing the /data/configuration/machine folder from within the Predix\_Machine Docker container.
- c. Restart the Predix Machine container for the changes to take effect:

```
docker restart Predix_Machine
```

 **Note:** Any configuration updates that are done for testing must include this IP Address in these configuration files or you must restart the MQTT container for every configuration update so the Predix Machine container can successfully talk to the MQTT container.

13. Restart the MQTT container for the changes to take effect:

```
docker restart MQTT
```

## *Preparing a Linux Device to Run the Containerized Predix Machine*

Download all Containerization software and components for your development environment as described in the *Download and Run Containerization Software* section.

Perform the following steps to start the containerized Predix Machine environment.

1. Download Docker 1.12.x or later.
2. Download the architecture-appropriate files to the device:
  - Predix Machine bootstrap docker image: `predixmachine-bootstrap-alpine-<architecture>-<version>.tar.gz`.
  - For the Predix Machine Agent Docker image: `predixmachine-agent-<architecture>-<version>.tar.gz`.

 **Note:** If you have customized Predix Machine, you can replace this with your own containerized version that you generated with the `GenerateContainers` and `DockerizeContainer` scripts.

- Mosquitto Messaging Broker Docker image: `predixmachine-mosquitto-<architecture>-<version>.tar.gz`
  - Docker images for any other applications, such as ones created with the Predix Machine edge SDK, provided as Docker images archived as `tar.gz`.
3. Load the bootstrap image into Docker's cache.

```
docker load -i predixmachine-bootstrap-alpine-<architecture>-<version>.tar.gz
```

4. Create a directory to mount to the bootstrap container.

This directory will be used by the bootstrap to manage the directory paths of the containers it manages.

5. Create and start the bootstrap container, which is located in <predixmachinesdk-installation\_location>/utilities/containers/bootstrap/:

```
bash start_bootstrap.sh -d <bootstrap mount directory>
```

Containers will start in host mode.

6. Copy the docker image archives for Predix Machine, Mosquitto, and any other custom applications to the directory monitored by bootstrap.

Bootstrap loads the images and starts containers. The process may take a little while, including time for the polling interval, time to load images, and start containers.

Ubuntu:

```
cp predixmachine-agent-x86_64-<version>.tar.gz predixmachine-mosquitto-x86_64-<version>.tar.gz <bootstrap mount directory>
```

Pi 3:

```
cp predixmachine-agent-armhf-<version>.tar.gz predixmachine-mosquitto-armhf-<version>.tar.gz <bootstrap mount directory>
```

7. Verify that the images were loaded and that the containers have started:

```
docker images
docker ps
```

The output of the Docker commands should, after some delay, list the Bootstrap, Predix Machine, Mosquitto, and any other custom containers that were introduced.

## *Generate Docker Images*

### *Generating a Predix Machine as a Docker Image using Command Line Scripts*

- Download Predix Machine from <https://artifactory.predix.io/artifactory/webapp/#/artifacts/browse/tree/General/PREDIX-EXT/predix-machine-package/predixmachinesdk/17.2.5/predixmachinesdk-17.2.5.zip>.

- Download Eclipse with PDE runtime plug-ins; for example Eclipse IDE for Java EE Developers. This download should remain in the .zip or tar.gz format.
- Ensure that you have Maven 3.1 or higher installed. On a command line interface, type `mvn -version` to verify the version.
- Install the latest version of Docker for Predix Machine version 17.2.x and later and all other Linux requirements as listed at <https://www.docker.com>.

You can generate a Predix Machine as a Docker image using a script similar to the one you can use to generate a runtime container.

## Docker Build Options

Creating a Docker image using an existing Predix machine uses the `GenerateContainers.sh` command, as follows:

```
GenerateContainers.sh [options]
```

For example:

- `sh GenerateContainers.sh -e ~/eclipse-jee-mars-4.5.0-macosx-cocoa-x86_64.tar.gz -c PROV sh GenerateContainers.`
- `sh -e $ECLIPSE_PATH -c CUSTOM /Users/user1/myMachine.img`
- `sh GenerateContainers.sh -e $ECLIPSE_PATH -c AGENT -d --docker_host myDockerHost --arch x86_64 --http_proxy http://proxy-src.research.ge.com:8080 --https_proxy http://proxy-src.research.ge.com:8080 --no_proxy localhost,127.0.0.1,*.ge.com`

`GenerateContainers.sh` uses the following options:

Required:

- `-e <ECLIPSE_PATH>` (Required): Path of downloaded Eclipse archive

Options:

- `-c <CONTAINER_TYPE>`: Type of Predix Machine container to create.
  - `AGENT`: Predix Machine with agent feature for Docker support. This container is required for running in the Dockerized model to use with the edge SDK.
  - `AGENT_DEBUG`: Predix Machine Debug with agent feature for Docker support.
  - `PROV`: Predix Machine Provision (includes only the JAR bundles that support provisioning).
  - `DEBUG`: Predix Machine Debug with Predix Machine Web Console.
  - `TECH`: A Technician Console image.
  - `CONN`: Predix Machine Provides connectivity support (VPN).
  - `TUNNEL`: A Predix Machine container with HTTP tunneling.
  - `CUSTOM <image file full path>`: A Predix Machine container using a custom image you created in Eclipse.

- [not specified]: Predix Machine default container.
- `-d`: Creates Predix Machine as a Docker image with the following options:
  - `--docker_host <DOCKER_HOST>`: Name of Docker host to use.
  - `--arch <ARCHITECTURE>`: The target architecture for the Docker image. The default: `x86_64`
  - `--ftp_proxy <FTP_PROXY_SERVER>`: FTP proxy server setting for Dockerized Predix Machine
  - `--http_proxy <PROXY_SERVER>`: HTTP proxy server setting for Dockerized Predix Machine
  - `--https_proxy <PROXY_SERVER>`: HTTPS proxy server setting for Dockerized Predix Machine
  - `--no_proxy <PROXY_EXCEPTIONS>`: A set of comma-separated domains that do not use the proxy

1. Open a terminal window.

2. From the command line, navigate to the `<Predix Machine SDK download location>/predixmachinesdk-17.2.x/utilities/containers` folder.

3. Run the following command:

```
GenerateContainers.sh -e <full path and name of downloaded Eclipse.tar.gz file> -c <container type> -d <Docker build option>
```

For example:

```
GenerateContainers.sh -e /home/eclipse-jee-mars-SR2-linux-gtk-x86_64.tar.gz -c AGENT
```

The script creates a `PredixMachine-agent-17.2.x` container in the `<Predix Machine SDK download location>/predixmachinesdk-17.2.x/utilities/containers` folder.

 **Note:** So that you can update your Predix Machine Docker container in the future, make sure to use the Docker image named `PredixMachine`. If a Predix Machine Docker image by a different name is pushed down to a device that is already running Predix Machine, errors occur. Predix Machine Docker images bearing a name other than `PredixMachine` are not recognized by Predix Machine Supervisor and cannot be started.

## Related concepts

Accessing Artifactory Downloads ([page](#))

## *Generating a Docker Image from a Predix Machine Runtime Container*

- Locate the Predix Machine runtime container for which you want to create a Docker image ("Dockerize")
- Install the latest version of Docker for Predix Machine 17.2.x and later and all other Linux requirements as listed at <https://www.docker.com>.

You can create a Docker image from an *existing* Predix Machine runtime container by using the `Dockerize.bat` or `Dockerize.sh` script and specifying build options, including specifying the container name.

You can use the following options when you build your Docker image.

- `-m`: (Required) The path of Predix Machine for which Docker image is created.
  - `--docker_host`: The Name of the Docker host.
  - `--container_name`: Meaningful name reflective of the Predix Machine container. For example, 'provision' for the provisioning container. It forms part of the Docker image tag. Defaults to 'default'.
-  **Important:** Do not create containers with any of the following names:
- `predixmachine`
  - `bootstrap`
- `--tar_name`: Base name of the resulting TAR file.
  - `--arch <ARCHITECTURE>`: The target architecture for the Docker image. Default: `x86_64`.
  - `--ftp_proxy <FTP_PROXY_SERVER>`: FTP proxy server setting for Dockerized Predix Machine.
  - `--http_proxy <PROXY_SERVER>`: HTTP proxy server setting for Dockerized Predix Machine.
  - `--https_proxy <PROXY_SERVER>`: HTTPS proxy server setting for Dockerized Predix Machine.
  - `--no_proxy <PROXY_EXCEPTIONS>`: A set of comma-separated domains that do not go through the proxy.

1. Open a terminal window.
2. In the command line, navigate to the `<Predix Machine SDK download location>/predixmachinesdk-17.2.x/utilities/containers` folder.
3. Run the following command:

```
DockerizeContainer.sh -m <path of Predix Machine for which Docker image is created> ---
docker_host <Docker host name> --container_name <container name> tar_name <tar file
name> --arch <Architecture> --http_proxy <HTTP proxy server setting for Dockerized Predix
Machine runtime container> --https_proxy <HTTPS proxy server setting for Dockerized Predix
Machine runtime container> --no_proxy <Comma-separated domains that do not go through
proxy>
```

For example:

```
DockerizeContainers.sh -m c:/MyPredixMachine --docker_host myDockerHost --
http_proxy http://my.proxy.com:8080 --https_proxy http://my.proxy.com:8080 no_proxy
"localhost,127.0.0.1*.my.com" DockerizeContainer.sh -m c:/MyPredixMachine --docker_host
myDockerHost --container_name provision --http_proxy http://my.proxy.com:8080 --
https_proxy http://my.proxy.com:8080 --no_proxy "localhost,127.0.0.1*.my.com"
```

The script creates the Predix Machine Docker image in the <Predix Machine SDK download location>/predixmachinesdk-17.2.x/utilities/containers folder.

## Troubleshoot Predix Machine

Use this document to help identify and resolve problems you may encounter.

### *Troubleshoot Eclipse*

You may encounter the following issues when using the Predix Machine SDK in Eclipse to generate or run Predix Machine.

#### *Eclipse container generation error: Help for java.exe parameters appears*

After you generate a Predix Machine runtime container using Eclipse and try to start the container, Help for java.exe parameters appears.

#### **Cause**

When you set your `JAVA_HOME` environment variable, you added either a trailing backslash or quotes.

#### **Solution**

Remove the trailing backslash or quotes from your `JAVA_HOME` environment variable.

#### *Eclipse container generation error: Application error*

When you attempt to generate a Predix Machine container using a script, the following error appears:

```
!ENTRY org.eclipse.osgi 4 0 2015-03-17 15:20:37.855 !MESSAGE Application
error !STACK 1 java.lang.RuntimeException: Application
"com.prosyst.tools.builder.core.generator" could not be found in the
registry. The
applications available are:
```

## Cause

You downloaded an Eclipse version without Plug-in Development Environment (PDE) support.

## Solution

Download Eclipse Luna, Kepler (for SDK versions 16.2.2 and lower), or Mars or Neon packages (for SDK version 16.3.x and higher) for Java EE Developers.

### *Eclipse error: Bundle is not found in the target platform*

When you attempt to export an SDK image to generate a Predix Machine container, the following error appears:

```
Bundle <bundle name> is not found in the target platform.
```

## Cause

The bundle that is named in the error message is no longer part of the SDK. It has been removed or renamed.

The Predix Machine SDK does not automatically replace or remove renamed bundles.

## Solution

Bundles that were included in a prior SDK version but have been removed or renamed in a newer version appear with a warning in the bundles list.

Remove the bundles that have been removed from the container and replace the renamed bundles with new versions.

### *Eclipse debug error: Failed to attach agent for Predix Machine*

When you debug a Predix Machine container from Eclipse, the following message appears:

```
Failed to attach agent for Predix Machine
```

## Cause

This is a timing issue. The Eclipse debugger tries to attach, but if the machine speed and the container size slows the container start, it may fail on the first attempt.

## Solution

Wait until the container has fully started and click **Try Again**.

### *Eclipse error: Container that includes OPC-UA bundles fails and runs out of memory*

In Eclipse, when you profile a container that includes the `com.ge.dspmicro.opcua-server` bundle, the profile action fails and you run out of memory.

## Cause

The OPC-UA bundle creates a large number of temporary objects per second. For each of those temporary objects, the profiler stores a lot of system information. The stack trace, where objects are created, uses the largest amount of memory.

## Solution

1. Increase the application heap from 1GB to 2GB for Eclipse. (-Xmx2024m).
2. Lower the mBProfiler memory usage.
  - a. In Eclipse, select **Run > Profile Configurations**.
  - b. In **Profile Configurations**, select your **OSGi Runtime** image.
  - c. Select the **mBProfiler** tab.
  - d. Lower the value in the **Maximum stack trace depth** field.

### *Javadoc Error: This element has neither an attached source nor attached Javadoc and hence no Javadoc can be found.*

## Condition

When trying to view a mouseover/Tool tip for JavaDocs in Eclipse the following message appears:

Note: This element neither has attached source nor attached Javadoc and hence no Javadoc can be found.

## Cause

Maven Dependencies are not set.

## Solution

1. In your Predix Machine SDK download location, unzip the `docs/apidocs.zip` file and extract all files.

2. In Eclipse, open the **Package Explorer** view.

The Properties dialog box appears.

3. In the **Maven Dependencies** section, locate the JAR file for which you want to view the Javadoc information, right-click, and choose **Properties**.
4. Click **Javadoc location**.
5. For the **Javadoc location path** field, click **Browse**.
6. Navigate to the Predix Machine SDK download location/docs, select the **apidocs** folder, and click **Validate**.
7. Repeat this procedure for each JAR file.

## *Troubleshoot Predix Machine*

Try these solutions if you encounter Predix Machine errors.

### *Predix Machine Build Errors*

When building Predix Machine samples, you may receive these types of errors:

```
Unresolvable build extension: Plugin
  org.apache.felix:maven-bundle-plugin:3.0.1 or one of its dependencies
  could not be resolved:
  Failed to read artifact descriptor for org.apache.felix:maven-bundle-
  plugin:jar:3.0.1 @
```

```
[ERROR] Some problems were encountered while processing the POMs: [WARNING]
  'version' contains an expression but should be a constant...
```

```
[ERROR] Unknown packaging: bundle @
  com.ge.dspmicro:sample-configuration:${dspmicro.sample-apps.version},
  <sample location...>\pom.xml, line 12,
  column 13 [WARNING] 'version' contains an expression but should be a
  constant.
```

```
Caused by: org.eclipse.aether.resolution.ArtifactResolutionException: Could
  not transfer artifact com.ge.dspmicro:gitrepository:pom:16.1.0 from/to
  artifactory.repo
  **(https://devcloud.swcoe.ge.com/artifactory/predix-virtual)**:
```

### **Cause**

You have not properly configured your proxy or Maven settings in the `<user directory>.m2/settings.xml` file.

## Solution

1. Generate an API key.
  - a. Log into <https://artifactory.predix.io>.
  - b. Select your user name.
  - c. Enter your password.
  - d. Click **Unlock** to populate the API Key field.
  - e. Copy the key.
2. Define the proxy settings for Maven based on your site needs.

 **Note:** Ask your network administrator if you have questions about your proxy requirements.

- a. Add proxy settings in the `<user directory>.m2/settings.xml` file:

 **Note:** You may also have to set proxy settings for the HTTPS protocol.

```
<proxy>
  <id>optional</id>
  <active>true</active>
  <protocol>http</protocol>
  <username>proxyuser</username>
  <password>proxypass</password>
  <host>proxy.host.com</host>
  <port>80</port>
  <nonProxyHosts>*.host.com|localhost</nonProxyHosts>
</proxy>
```

- b. Add a server entry with the following information. (Use the API key copied in Step 1.)

```
<server>
  <id>artifactory.external</id>
  <username>predix cloud login</username>
  <password>{encrypted password - API key}</password>
</server>
```

## "Lock" File Error

If you encounter a "lock" file error, try the following solution.

This issue may be encountered in the following scenarios:

- If Predix Machine is installed as a service, or starts automatically when the machine starts, Predix Machine does not start, but no error is thrown, so it is difficult to debug.
- When the device powers on, if you start Predix Machine directly from the command line by running the `start_predixmachine.sh` script, you see the following error:

```
Predix Machine lock file exists at /PredixMachine-connectivity-  
<version>/security/lock.  
Either another instance of Predix Machine is running, or the last  
instance was shutdown incorrectly.
```

## Cause

By default, Predix Machine creates a "lock" file under the <PredixMachine\_Home>/security/ folder, which remains if the machine is powered off or not shut down properly.

## Solution

There are two possible solutions if you encounter this problem. The first one is recommended because it prevents the issue from occurring in the first place.

- Set the PREDIXMACHINELOCK environment variable to point to /tmp or /var/run/predixmachine so that the "lock" file that is created in the <PredixMachine\_Home>/security folder is automatically cleared when the machine is restarted or not shut down properly.
- Delete the "lock" file and then re-run the `start_predixmachine.sh` script.

## *Troubleshoot Device Enrollment*

You may encounter the following issues when enrolling devices.

### *Enrollment error: User Cannot Access Enrollment Login*

Troubleshoot the enrollment error where the user cannot access the enrollment login.

## Problem

If a user clicks **Enroll**, the **Login** page does not appear.

## Cause

This issue may be caused by the following:

- UAA URL is incorrect.
- User's browser has incorrect proxy settings.
- User is already logged into Edge Manager.

## Solution

Try the following solutions:

- Verify the UAA URL as the root UAA URL, and that it does not include the path to the token endpoint.
- Verify that the user can access other sites. If not, check the proxy configuration.
- No resolution is required unless a different user needs to perform enrollment. If so, log out of Edge Manager before enrolling.

### *Enrollment error: Request for Authorization was Invalid*

When enrolling a device, the following error is received:

```
The request for authorization was invalid.
```

## Cause

Possible causes include the following:

- The device ID does not exist.
- The activation code is incorrect.

## Solution

Try the following solutions to resolve the issue:

- Create the device in Edge Manager.
- Make sure that the correct Edge Manager activation code is used.

### *Enrollment error: Could not perform enrollment because user <username> is not authorized to enroll this device*

User is not authorized to enroll a device.

In the Technician Console, the following error message is received:

```
Could not perform enrollment because user <Username> is not authorized to enroll this device.
```

## Cause

The user is not assigned to this device in Edge Manager.

## Solution

The administrator must assign the user to the device in Edge Manager.

### *Enrollment error: Could not get token for this client*

In the Technician Console, the user/technician gets the following error message:

```
Could not get token for this client
```

## Cause

Possible causes include:

- Predix Machine proxy settings are not correct.
- The user is not assigned to the Technician role.

## Solutions

Try the following solutions to resolve the issue:

- The administrator should verify the Predix Machine settings on the **Configuration** tab in the Web Console.
- The administrator should verify that the user is assigned to the Technician role.
- If you use a proxy server, set HTTP Client proxies.
  1. In the Predix Machine Web Console, click **OSGI > Configuration**.
  2. Click the **Apache HttpClient** OSGI bundle in the Name column. For example, **CM\_GENERATED\_PID.0**.
  3. Select the **proxy.enabled.name** check box.
  4. In the **proxy.host.name** box, enter the proxy host.
  5. In the **proxy.port.name** box, enter the proxy port number.
  6. Complete the **proxy.password.name**, and **proxy.exceptions.name** box and click **Save**.

### *Enrollment error: Assigned technician cannot enroll device*

If you are testing the Technician Console as a technician and attempt to enroll a device for which the technician is not authorized, the login fails. If you then use the same machine and browser to attempt to enroll the device as the correctly-assigned technician, the enrollment fails.

## Cause

The User Access and Authentication (UAA) service cookies for the incorrect technician login persist in the browser.

## Solution

Delete the UAA cookies from the browser to allow the assigned technician to enroll the device.

### *Enrollment error: Deleted device still has Reachable status*

If you delete an enrolled device that has an Online status in Edge Manager, the device remains in an Online status, and you cannot create another device with the same name or ID.

## Cause

Each successful enrollment request sent from the device to the cloud contains a UAA token. This token is valid for 15 minutes.

## Solution

Wait for at least 15 minutes before adding a new device with the same name or ID.

## Verification

After 15 minutes, create a device with the same name/ID.

### *Enrollment error: Docker image timeout*

When deploying a Docker image to an enrolled device in Edge Manager, you may receive timeout errors, as shown in the following examples:

```
Time expired, stop looking for watchdog fw result
file
```

```
HTTP request error. java.net.SocketTimeoutException: Read timed
out
```

## Cause

The Docker image might be very large (for example, greater than 100MB) for a low-memory or low-consumption device like Raspberry Pi and so requires more time.

## Solution

1. Change the package expiration timeout setting.
  - a. Navigate to <Predix Machine runtime container location>/configuration/machine and open the `com.proximity.osgiagent.impl.DeviceService.cfg` file.
  - b. Change the value of the `package.expiration.timeout` property to be higher than the default of `1000`.
2. Change the HTTP Client Socket timeout expiration setting.
  - a. Navigate to <Predix Machine runtime container location>/configuration/machine and open the `com.ge.dspmicro.httpclient.config` file.
  - b. Change the value of the `com.ge.dspmicro.httpclient.socket.timeout` property to be higher than the default of `I"600000"`.

### Note:

If this happens when updating an enrolled Predix Machine Agent container:

1. Remove the existing Predix Machine Agent container.
2. Reinstall the container.
3. Re-enroll the device.
4. Change the timeout configurations as noted above.
5. Push updates to the Predix Machine Agent from the cloud.

## *Enrollment error: Authentication failure*

When attempting to enroll a Predix Machine Agent container in Edge Manager, the following error appears:

```
HTTP request
    error. com.ge.dspmicro.httpclient.api.HttpWithStatusCodeException:
    Could not
    authenticate using provided credentials. The OAuth2 endpoint
    responded with status code
    401.
```

## Cause

Either your certificate enrollment shared secret is not correct, or the Network Time Protocol (NTP) is not set correctly.

## Solution

Use either of the following steps to resolve the issue.

- If your shared secret is not correct, correct the secret associated with your certificate.
- Set the correct NTP date on your device. The following example shows how to set the NTP date on a Raspberry Pi.
  1. On your device, open the `vi/etc/system/ timesyncd.conf` file.
  2. Uncomment `NTP` and add an NTP Server. For example:

```
NTP=ntp.ubuntu.com
```

3. Set the system for NTP to synchronize the time.

```
timedatectl set-ntp true
```

4. Re-start the Time Sync service.

## Verification

1. Determine the time status using either of the following ways:

- ```
systemctl status systemd-timesyncd
```

The result should indicate `Synchronized to time server <ntp or ubuntu IP>(ntp.ubuntu.com)`

- ```
timedatectl status
```

Verify that the `NTP synchronized` property value is `yes`.

2. Verify the date is correct.

```
date
```

## *Credentials do not Work After Device Enrollment*

### Errors After Apparently Successful Device Enrollment

This issue happens on Predix Machine 17.2.0 and later running on Windows.

After apparently successfully enrolling a device with Predix Machine, Predix Machine restarts and then does not accept the login credentials used to enroll the device, and the device does not appear "online" in Edge Manager.

You may also see the following errors:

- **ERROR 1:** Failed to construct SSLContext
- **ERROR 2:** Failed to load keystore file

### Cause

Predix Machine 17.2 does not support Windows in a production environment.

The SDK environment under Windows is supported for building Predix Machine containers, but in order to connect and enroll with the cloud you must run the `bin/start_predixmachine.sh` script, which starts the yeti process, which is required for device enrollment.

### Solution

You can use the Docker solution on Windows for testing with the cloud.

## *Troubleshoot Synchronization Issues*

### *DeviceId does not match caller security context*

Within Edge Manager, Predix Machine goes offline, with error logs indicating **UnauthorizedDeviceIdException**. The sync call to Edge Manager fails with Http Status Code 403.

A detailed error message indicates **DeviceId does not match caller security context**.

### Cause

The value of the `DeviceId` field is incorrect.

### Solution

Fix the `DeviceId` in the field `com.ge.dspmicro.predixcloud.identity.deviceid` within the file `$PREDIX_HOME/configuration/machine/com.ge.dspmicro.predixcloud.identity.config`.

 **Note:** `Deviceid` is a case-sensitive field.

## *Troubleshoot Web Console Errors*

You can encounter the following issues when using the Predix Machine Web Console.

### *Web console error: Configuration properties are cleared when container restarts*

When you have set configuration properties in the Predix Machine Web Console and then run a sample, the configuration properties are lost when Predix Machine restarts.

#### **Cause**

The browser cache is cleared when the container restarts.

#### **Solution**

Set your configuration in the sample CONFIG file corresponding to the sample you are trying to run. For example, if you are running the Subscription Machine Adapter sample, set the configuration properties in the `<SDK installation location>/samples/sample-apps/sample/configuration/machine/com.ge.dspmicro.sample.subscriptionmachineadapter.config` file.

### *Web console error: ModbusIOException*

The following exception appears on the Predix Machine Web Console:

```
net.wimpi.modbus.ModbusIOException: Premature end of stream (Header truncated)
```

#### **Cause**

The Modbus Machine Adapter caches the TCP/IP connection for performance purposes after the first successful communication. The subscription `updateInterval` in the Modbus Machine Adapter XML configuration file is so long that the device has closed the cached connection.

The Modbus Machine Adapter creates a new connection and tries a second time when the cached connection is stable, so the data is fetched successfully despite the exception.

#### **Solution**

Increase the connection timeout setting on the device, or reduce the subscription `updateInterval` in the Modbus Machine Adapter XML configuration file.

## *Troubleshoot Container Errors*

You may encounter the following issues when running or starting the Predix Machine container.

### *Container error: Unable to start Predix Machine container*

You cannot start a Predix Machine container.

#### **Cause**

Your limited license has expired. You did not enroll your device before the limited license expired for the SDK version you used to generate your container.

#### **Remedy**

Download the latest version of the Predix Machine SDK and enroll your device before the limited SDK license expires.

### *Container error: Java keytool not found*

When you try to run the Predix Machine container, the following error message appears:

```
[predix@localhost predix]$ ./predixmachine Java keytool not found. Exiting.
```

#### **Cause**

The Predix Machine start script uses the keytool command to generate a default key when it starts. The keytool must be part of your system PATH so that it can be located by the startup scrip.

#### **Solution**

Download the JDK (not the JRE) from <http://openjdk.java.net/install/> or [Oracle](#). After downloading the JDK, add the SDK bin directory to your system PATH.

### *Container error: Active bundle is listed as inactive*

Your bundle is active, but when you list the service using the scr.list command, your service is listed as inactive.

## Possible Causes

- A configuration file is missing.
- Items in the service are unsatisfied.
- Incorrect entries in the `permission.perm` file inside your bundle.

## Possible Solutions

- Add the required service `id=-1` configuration file to the service.
- If service ID is not `-1`, review the service details using the `(scr.component{service#})` command. Review the list of items that are not satisfied and address those issues.
- If the list of unsatisfied items lists no reasons but the service is inactive, it may be a permissions issue.
  - If you are using a `permissions.perm` file in the bundle they may not be correct. You can either remove the `permissions.perm` file or add all permissions: `(java.security.AllPermission "*" "*")` and retry.
  - If this fixes the service, then enable the `mbs.debug=10` line in the Predix Machine container location/`configuration/machine/predix.prs` file to display debug information in the console for resolving the permission issue.

## *Permission security exception error: access denied*

The following permission security exception error appears:

```
java.security.AccessControlException: access denied ("java.io.FilePermission"
"." "read")
```

## Cause

The Predix Machine thread that is calling a solution callback requires permission to perform the operation. A Predix Machine bundle (which has restricted permissions) is attempting to execute code in the callback that exceeds its permission set. You can implement the callback code in a thread managed by the bundle, or you can wrap the code in the access code in the example noted in the solution below.

## Solution

Add `AccessController.doPrivileged()` to the callback method in your solution application:

```
protected void callback_method(final SomeObject event)
{
    AccessController.doPrivileged(new PrivilegedAction<Void>()
    {
        @Override
```

```

        public Void run()
        {
            handleEvent(event);
            return null;
        }
    });
}

```

## *Modbus Adapter error: SocketTimeoutException*

The following exception appears from the Modbus Adapter:

```

java.net.SocketTimeoutException: Read timed out
    at java.net.SocketInputStream.socketRead0(Native Method)
    at java.net.SocketInputStream.read(SocketInputStream.java:152)
    at java.net.SocketInputStream.read(SocketInputStream.java:122)
    at java.io.BufferedInputStream.fill(BufferedInputStream.java:235)
    at java.io.BufferedInputStream.read1(BufferedInputStream.java:275)
    at java.io.BufferedInputStream.read(BufferedInputStream.java:334)
    at java.io.DataInputStream.read(DataInputStream.java:149)
    at
    net.wimpi.modbus.io.ModbusTCPTransport.readResponse(ModbusTCPTransport.java:171)
    at
    net.wimpi.modbus.io.ModbusTCPTransaction.execute(ModbusTCPTransaction.java:193)
    at
    com.ge.dspmicro.machineadapter.modbus.impl.ModbusCommunication.read(ModbusCommunication
    at
    com.ge.dspmicro.machineadapter.modbus.impl.ModbusCommunication.read(ModbusCommunication
    at
    com.ge.dspmicro.machineadapter.modbus.impl.ModbusMachineAdapterImpl
    $SubscriptionWorker.run(ModbusMachineAdapterImpl.java:458)
    at java.util.concurrent.Executors
    $RunnableAdapter.call(Executors.java:471)
    at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:304)
    at java.util.concurrent.ScheduledThreadPoolExecutor
    $ScheduledFutureTask.access$301(ScheduledThreadPoolExecutor.java:178)
    at java.util.concurrent.ScheduledThreadPoolExecutor
    $ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:293)
    at
    java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor
    $Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:745)

```

## Cause

The `unit id` or `register address` specified in the Modbus Adapter XML configuration file is invalid. During each read interval, the Modbus Adapter reads each invalid node twice. If several nodes are invalid, this requires more time to complete.

## Remedy

Make sure the `unit id` or `register address` specified in the XML configuration file `<Predix Machine runtime container location>/configuration/machine/com.ge.dspmicro.machineadapter.modbus-[n].xml` is valid on the device.

 **Note:** See [Configuring the Modbus Machine Adapter \(page 145\)](#).

## *Troubleshoot Container Upgrades*

You can encounter the following issues when upgrading Predix Machine containers.

### *Script Overwrite of Predix.PRS file*

When performing a Predix Machine upgrade, as described in [Upgrading a Predix Machine Container \(page 69\)](#), certain configuration files are whitelisted and are not overwritten during the upgrade. This does not include the `predix.prs` file, which is overwritten in the upgrade.

## Cause

This is by design, but refer to the Remedy for possible workarounds.

## Remedy

Consider some of the following options to prevent overwriting the `predix.prs` file during upgrade.

- Make the preconfigured `predix.prs` part of your update package. If you upload the same `predix.prs` file, it replaces itself and does not interfere with your current configuration. Additionally, if you need to change configurations, you can edit the `predix.prs` in your update package through Edge Manager and push it to your device.
- If you do not need to update the `predix.prs`, don't include it in your update package. The updater overlays configurations, so if there is no new `predix.prs` file to overwrite with, it will not overwrite.

- Alternatively, add the `predix.prs` to the whitelist of the configuration install script for your device. These scripts are editable through Predix Edge Manager or can be edited before uploading the package.

## Release Notes

### *Predix Machine Release Notes Version 17.2.5*

These are the resolved and known issues for Predix Machine 17.2.5.

#### **Enhancements**

This release contains the following enhancements:

##### **Time Series Data Ingestion Through MQTT Adapter**

Predix Machine can now ingest data in the Time Series format through the MQTT protocol adapter for transmitting to the Time Series service via the Websocket River. This allows your applications to include custom attributes in the Time Series data without having to create custom processors.

#### **Resolved Issues**

This release contains the following resolved issues:

##### **Eventhub River Unable to Process BAD Quality PData**

Predix Machine Event Hub River was not able to process PData with the attribute quality BAD. This issue has been fixed.

##### **OpenVPN/On Demand Events Binding Failures**

Predix Machine OpenVPN management occasionally failed to bind to the IPv6 address on the tun0 interface, causing ODE not to work as expected. This issue has been fixed.

##### **Websocket River Cannot Handle Extra Fields**

Predix Machine Websocket River was not able to handle extra fields in the response from the Time Series service. This issue has been fixed.

### *Predix Machine Release Notes Version 17.2.4*

These are the resolved and known issues for Predix Machine 17.2.4.

## Enhancements

This release contains the following enhancements:

### **Extended Life for Root Certificates**

The life has been extended for the included required root certificates for Predix Machine 17.2.4.

## Resolved Issues

This release contains the following resolved issues:

### **Modbus Adapter Fails to Read COIL Boolean Data**

The issue where the Predix Machine Modbus Adapter was failing to read COIL boolean data has been fixed.

### **Global ipv6 not Found**

The issue where the global ipv6 was not found for the OpenVPN Network Interface, which caused an error to be thrown, has been fixed.

### **OpenVPN Management Event Listener Blocks on Socket Read**

The issue where sometimes, on slow network connections or slow devices, the management event listener received a connection request but no data, causing the connection to hang until the connection was closed has been resolved.

### **Certificate Renewal Failure**

The issue where if a device was turned off, or went offline for some reason, when the device woke up after the certificate had expired, certificate renewal failed, has been fixed.

## Known Issues

This release has the following known issues:

### **Device ID Converted to Lower Case When Added to Edge Manager**

When a device is added to Edge Manager, the deviceId is converted to all lower case. So, for example, if you add a device called "CamelCase," it is converted to "camelcase" and stored that way in the database. This can cause issues in cases where an API is case-sensitive.

### **Space in Predix Machine Adapter or Spillway Causes Error**

When the Predix Machine adapter's or spillway's subscription name contains a space (for example, "Analytics Subscription"), the data is read and sent to spillway/StoreForward. When reading from StoreForward and sending the data to the Data Bus River, this subscription name is, by default, used as the topic name for sending data to the MQTT broker.

However, topic names are invalid if they contain spaces, so sending the data through the Data Bus River fails with an error similar to the following:

```
2017-12-01 02:01:09,457[pool-8-thread-1] |WARN|
com.ge.dspmicro.hoover.impl.spillway.SpillwayImpl|132-
com.ge.dspmicro.hoover-impl-<version>|
Spillway Transfer Failed.
Details: "Status for transfer UUID "<UUID>" in River
"DataBusRiverForOPCUA".
State: FAILED
Status Code: -1
Status Message: Transfer failed: Invalid topic tag "Analytics
Subscription". Make sure topic tag contains only A-Z, a-z, _,
0-9.
```

**Workaround:** Use only A-Z, a-z, \_, 0-9 for Predix Machine adapter's or spillway's subscription names.

### Move Event not Propagated on Mac

On Mac, when moving Docker images to the bootstrap mount directory, the "move" event is not propagated through to the bootstrap container, which causes bootstrap to ignore any files moved to its mounted directory.

Additionally, when copying Docker images, the "copy" event does not always propagate, which results in bootstrap not always picking up copied files.

### Docker Images

When Docker bootstrap creates the application container (for example, an analytics container), it creates a folder for it under the mounted directory `container/<folder>`, which may later be used by the application container).

When bootstrap receives a Delete Application Container command from Edge Manager, it deletes the application container, but does not delete the corresponding folder. If the application container is deployed again, a new folder will be created for the new container. Thus, if deleting the application container and deploying a new application container is done numerous times, many extraneous folders will remain on `<mounted dir>/containers/<folder>`.

This may cause out of storage problems for a device that is used for a long time out in the field.

### WebSocket River failed to send data

Occasionally, WebSocket River will fail to send data when the network switches from no proxy to a network that uses proxy.

**Workaround:** If this happens, you must restart the WebSocket River bundle.

### Predix Machine Agent container upgrade

Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#).

**Workaround:** You can upgrade by pushing a new Docker image through Edge Manager.

### OPC-UA Adapter exceptions

In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcua.security.mode` property to `BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## Export Control

Predix Machine is classified for export control and uses ECN 5D002.

## *Predix Machine Version 17.2.3*

These are the enhancements, and resolved and known issues for Predix Machine 17.2.3.

### Enhancements

This release contains the following enhancements:

#### Store and Forward Database File Size Limit

In previous versions of Predix Machine (17.2, 17.2.1, 17.2.2), the following behavior was noted:

 **Note:**

The Store and Forward service may not stop writing precisely at the specified limit. Tape data stores are created with 4KB initially and its size doubles every time it runs out of space. Store and Forward allows the doubling as long as the size is below the specified limit. For example, if the `dbSizeLimitMB` is set to 3MB, and the current data store size is at 2MB, expansion will happen, pushing its size to 4MB. The next time it runs out of space, expansion will not happen because the size is already above the set `dbSizeLimitMB`.

 **Note:**

Once the Tape data store reaches or exceeds the set `dbSizeLimitMB`, the Store and Forward service stops writing to the store and does not resume until all the data in the store is forwarded. This behavior is different from the previous H2 implementation, which resumed writes as soon as the store size dropped below the `dbWriteResumeLimitMB`.

This behavior no longer applies in Predix Machine version 17.2.3, however, it is strongly recommended that you set the `com.ge.dspmicro.storeforward.dbSizeLimit` parameter to a reasonable value based on the storage space available on the device, so the device does not run out of space in case connectivity is lost for extended periods of time.

See [Configuring the Data Store and Forward Service \(page 125\)](#).

### **Task Statuses are Immediately Uploaded to the Cloud**

The `com.ge.dspmicro.cloud.gateway.uploadTaskStatusOnSubmitEnabled` property in the `com.ge.dspmicro.cloud.gateway.config` file is now set to "true" by default. This means that any task statuses are immediately uploaded to the cloud when submitted to the Cloud Gateway service and that Predix Machine retrieves the next pending tasks when it uploads the task status.

## **Resolved Issues**

The following issues were resolved in this release:

### **Enrollment URL not Reset After Failure**

The issue where the `com.ge.dspmicro.predixcloud.identity.tenantid` parameter was not getting reset in the `com.ge.dspmicro.predixcloud.identity.config` file when an enrollment URL was incorrect, which caused enrollment to fail, is now fixed.

### **RunJournalCtl Hangs with Large Output**

The issue that caused the Java Process to become deadlocked in cases where the `Journalctl` system command was used without filtering, which produced very large output and resulted in an uncaught `OutOfMemoryError` is now fixed.

## **Known Issues**

This release has the following known issues:

### **Device ID Converted to Lower Case When Added to Edge Manager**

When a device is added to Edge Manager, the `deviceId` is converted to all lower case. So, for example, if you add a device called "CamelCase," it is converted to "camelcase" and stored that way in the database. This can cause issues in cases where an API is case-sensitive.

### **Space in Predix Machine Adapter or Spillway Causes Error**

When the Predix Machine adapter's or spillway's subscription name contains a space (for example, "Analytics Subscription"), the data is read and sent to spillway/StoreForward. When reading from StoreForward and sending the data to the Data Bus River, this subscription name is, by default, used as the topic name for sending data to the MQTT broker.

However, topic names are invalid if they contain spaces, so sending the data through the Data Bus River fails with an error similar to the following:

```
2017-12-01 02:01:09,457[pool-8-thread-1] |WARN|
com.ge.dspmicro.hoover.impl.spillway.SpillwayImpl|132-
com.ge.dspmicro.hoover-impl-<version>|
Spillway Transfer Failed.
Details: "Status for transfer UUID "<UUID>" in River
"DataBusRiverForOPCUA".
State: FAILED
Status Code: -1
Status Message: Transfer failed: Invalid topic tag "Analytics
Subscription". Make sure topic tag contains only A-Z, a-z, _,
0-9.
```

**Workaround:** Use only A-Z, a-z, \_, 0-9 for Predix Machine adapter's or spillway's subscription names.

### Move Event not Propagated on Mac

On Mac, when moving Docker images to the bootstrap mount directory, the "move" event is not propagated through to the bootstrap container, which causes bootstrap to ignore any files moved to its mounted directory.

Additionally, when copying Docker images, the "copy" event does not always propagate, which results in bootstrap not always picking up copied files.

### Docker Images

When Docker bootstrap creates the application container (for example, an analytics container), it creates a folder for it under the mounted directory `container/<folder>`, which may later be used by the application container).

When bootstrap receives a Delete Application Container command from Edge Manager, it deletes the application container, but does not delete the corresponding folder. If the application container is deployed again, a new folder will be created for the new container. Thus, if deleting the application container and deploying a new application container is done numerous times, many extraneous folders will remain on `<mounted dir>/containers/<folder>`.

This may cause out of storage problems for a device that is used for a long time out in the field.

### WebSocket River failed to send data

Occasionally, WebSocket River will fail to send data when the network switches from no proxy to a network that uses proxy.

**Workaround:** If this happens, you must restart the WebSocket River bundle.

### Predix Machine Agent container upgrade

Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#).

**Workaround:** You can upgrade by pushing a new Docker image through Edge Manager.

### Third-party bundle migration

Third-party bundles have been migrated to later versions. Open your image in the Eclipse editor for migrating bundles to later versions if required.

### OPC-UA Adapter exceptions

In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcua.security.mode` property to `BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## Export Control

Predix Machine is classified for export control and uses ECN 5D002.

## *Predix Machine Version 17.2.2*

These are the enhancements, and resolved and known issues for Predix Machine 17.2.2.

### Enhancements

This release contains the following enhancements:

#### New Commands

New commands to help the administrator troubleshoot different issues have been added. The new commands include:

- `run_dpkg` – Lists all packages installed on the device using system command `dpkg`.
  -  **Note:** Only for Linux devices that support `dpkg`.
- `run_ifconfig` – Runs the system command `ifconfig` on the device and returns the command output.
  -  **Note:** This is only for devices that support `ifconfig`.
- `run_top` – Runs the system command `top` on the device for one iteration.
  -  **Note:** This is only for devices that support `top`.

- `run_journalctl` – Queries the default system journal on the device using system command `journalctl` according to the filtering and formatting values specified in the command parameters.

 **Note:** Only for Linux devices that support `journalctl`.

- `refresh_device_detail` – Sends all device detail information in the next sync. This command overrides the Cloud Gateway's polling interval (`com.ge.dspmicro.cloud.gateway.pollingInterval`) and schedules a sync immediately.

See the [Commands and Command Formats \(page 96\)](#) documentation for details about these new commands.

## Resolved Issues

This release contains the following resolved issues:

### WebSocket River Memory Leak

The issue where a memory leak was observed in WebSocket River when connections were repeatedly opened and closed has been resolved.

### Problem Re-enrolling After Upgrading

The problem experienced when re-enrolling and resetting the Predix Machine container after upgrading from 17.1.x to 17.2.x has been resolved.

### Store and Forward Memory Leak

The issue where a memory leak was observed in the Store and Forward service has been resolved.

 **Note:** This applies to the Tape implementation only.

### WebSocketRiverSend Threads Deadlocking

The issue where `WebSocketRiverSend` threads were deadlocking, due to one thread trying to close the `httpClient` connection, due to an exception, and another thread trying to execute an `https` request has been resolved.

### Problem when Connecting Predix Machine Through Proxy

The issue where there was a problem connecting Predix Machine through a proxy with NTLM authentication has been resolved by reverting back to the previous versions of the `httpClient` libraries.

## Known Issues

This release has the following known issues:

### Space in Predix Machine Adapter or Spillway Causes Error

When the Predix Machine adapter's or spillway's subscription name contains a space (for example, "Analytics Subscription"), the data is read and sent to spillway/StoreForward. When reading from StoreForward and sending the data to the Data Bus River, this subscription name is, by default, used as the topic name for sending data to the MQTT broker.

However, topic names are invalid if they contain spaces, so sending the data through the Data Bus River fails with an error similar to the following:

```
2017-12-01 02:01:09,457[pool-8-thread-1]|WARN|
com.ge.dspmicro.hoover.impl.spillway.SpillwayImpl|132-
com.ge.dspmicro.hoover-impl-<version>|
Spillway Transfer Failed.
Details: "Status for transfer UUID "<UUID>" in River
"DataBusRiverForOPCUA".
State: FAILED
Status Code: -1
Status Message: Transfer failed: Invalid topic tag "Analytics
Subscription". Make sure topic tag contains only A-Z, a-z, _,
0-9.
```

**Workaround:** Use only A-Z, a-z, \_, 0-9 for Predix Machine adapter's or spillway's subscription names.

### Move Event not Propagated on Mac

On Mac, when moving Docker images to the bootstrap mount directory, the "move" event is not propagated through to the bootstrap container, which causes bootstrap to ignore any files moved to its mounted directory.

Additionally, when copying Docker images, the "copy" event does not always propagate, which results in bootstrap not always picking up copied files.

### Docker Images

When Docker bootstrap creates the application container (for example, an analytics container), it creates a folder for it under the mounted directory `container/<folder>`, which may later be used by the application container).

When bootstrap receives a Delete Application Container command from Edge Manager, it deletes the application container, but does not delete the corresponding folder. If the application container is deployed again, a new folder will be created for the new container. Thus, if deleting the application container and deploying a new application container is done numerous times, many extraneous folders will remain on `<mounted dir>/containers/<folder>`.

This may cause out of storage problems for a device that is used for a long time out in the field.

### WebSocket River failed to send data

Occasionally, WebSocket River will fail to send data when the network switches from no proxy to a network that uses proxy.

**Workaround:** If this happens, you must restart the WebSocket River bundle.

### **Predix Machine Agent container upgrade**

Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#).

**Workaround:** You can upgrade by pushing a new Docker image through Edge Manager.

### **Third-party bundle migration**

Third-party bundles have been migrated to later versions. Open your image in the Eclipse editor for migrating bundles to later versions if required.

### **OPC-UA Adapter exceptions**

In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcua.security.mode` property to `BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## **Export Control**

Predix Machine is classified for export control and uses ECN 5D002.

## *Predix Machine Version 17.2.1*

These are the enhancements, and known and resolved issues for Predix Machine 17.2.1.

### **Resolved Issues**

This release contains the following resolved issues.

#### **OpenVPN in Docker**

The issue where if a host has two IPV6 addresses (Local and Global), Predix Machine may ignore the Global IPV6, which causes the On-demand Event service to not work is resolved.

#### **Enrollment Failing when securityadmin.config property set to false**

The issue where when the `com.ge.dspmicro.securityadmin.passwords.encrypt` property in the `com.ge.dspmicro.securityadmin.config` file was set to `false`, Predix Machine enrollment failed is resolved. Previously, the device status changed from "Create" to "Offline" but could not update to "Online" right after enrollment.

## OPCUA Multiple Subscriptions but Only one Topic

The issue where OPCUA would have multiple subscriptions but only one topic on the database has been fixed.

## Saving Application Configuration Causing Error

The issue where an error occurred in saving the configuration of an application in Predix Machine 17.1.3 has been fixed.

## Known Issues

This release has the following known issues:

### Space in Predix Machine Adapter or Spillway Causes Error

When the Predix Machine adapter's or spillway's subscription name contains a space (for example, "Analytics Subscription"), the data is read and sent to spillway/StoreForward. When reading from StoreForward and sending the data to the Data Bus River, this subscription name is, by default, used as the topic name for sending data to the MQTT broker.

However, topic names are invalid if they contain spaces, so sending the data through the Data Bus River fails with an error similar to the following:

```
2017-12-01 02:01:09,457[pool-8-thread-1]|WARN|
com.ge.dspmicro.hoover.impl.spillway.SpillwayImpl|132-
com.ge.dspmicro.hoover-impl-<version>|
Spillway Transfer Failed.
Details: "Status for transfer UUID "<UUID>" in River
  "DataBusRiverForOPCUA".
State: FAILED
Status Code: -1
Status Message: Transfer failed: Invalid topic tag "Analytics
  Subscription". Make sure topic tag contains only A-Z, a-z, _,
  0-9.
```

**Workaround:** Use only A-Z, a-z, \_, 0-9 for Predix Machine adapter's or spillway's subscription names.

### Move Event not Propagated on Mac

On Mac, when moving Docker images to the bootstrap mount directory, the "move" event is not propagated through to the bootstrap container, which causes bootstrap to ignore any files moved to its mounted directory.

Additionally, when copying Docker images, the "copy" event does not always propagate, which results in bootstrap not always picking up copied files.

## Docker Images

When Docker bootstrap creates the application container (for example, an analytics container), it creates a folder for it under the mounted directory `container/<folder>`, which may later be used by the application container).

When bootstrap receives a Delete Application Container command from Edge Manager, it deletes the application container, but does not delete the corresponding folder. If the application container is deployed again, a new folder will be created for the new container. Thus, if deleting the application container and deploying a new application container is done numerous times, many extraneous folders will remain on `<mounted_dir>/containers/<folder>`.

This may cause out of storage problems for a device that is used for a long time out in the field.

### **WebSocket River failed to send data**

Occasionally, WebSocket River will fail to send data when the network switches from no proxy to a network that uses proxy.

**Workaround:** If this happens, you must restart the WebSocket River bundle.

### **Predix Machine Agent container upgrade**

Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#).

**Workaround:** You can upgrade by pushing a new Docker image through Edge Manager.

### **Third-party bundle migration**

Third-party bundles have been migrated to later versions. Open your image in the Eclipse editor for migrating bundles to later versions if required.

### **OPC-UA Adapter exceptions**

In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcua.security.mode` property to `BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## **Export Control**

Predix Machine is classified for export control and uses ECN 5D002.

## *Predix Machine Version 17.2.0*

These are the new features, enhancements, and known and resolved issues for Predix Machine 17.2.0.

## Predix Machine 17.2.0 New Features

This release contains the following new features.

### Event Hub River Service

Predix Machine Event Hub River service provides connectivity between Predix Machine and the Event Hub service. Event Hub is a publish-subscribe service that is built to be secure, massively scalable, fault-tolerant, and language-agnostic. Event Hub can ingest streaming data from anywhere and send it to the cloud for processing.

See [Event Hub River Service \(page 85\)](#).

### Data Store and Forward uses Tape

Tape is a collection of queue-related classes that provides reliable persistence of data.

See [Data Store and Forward \(page 122\)](#).

### On-Demand Event Service

The On-demand Event service enables bi-directional connectivity between Predix Machine-powered devices and Predix Edge Manager. To use this feature, you must either install OpenVPN, or download the Docker image that includes OpenVPN. See [Downloading Docker Images \(page 273\)](#).

## Enhancements

This release contains the following enhancements.

### Compare Containers to see Updated JAR Files

When Predix Machine is updated to a new version, only JAR files that are also updated change versions. There is a program available in `<Predix Machine SDK download location>/predixmachinesdk-17.2.x/utilities/CompareContainer` that will compare two machine folders and build a download package, with updated JAR files, for upgrading from one to the other, or for adding components to an existing container.

### Web Server Jetty Libraries Upgraded

The Web Server Jetty libraries have been upgraded to the latest versions.

### Default Polling Interval Increased

The default polling interval for the `com.ge.dspmicro.cloud.gateway.pollingInterval` property has been increased from 30 seconds to 300 seconds (5 minutes). See [Configuring Cloud Gateway \(page 187\)](#).

## Resolved Issues

This release contains the following resolved issues.

### **Predix Machine does not reconnect to Mosquitto**

Restarting a Mosquitto (MQTT) Docker container causes the Predix Machine Docker container to lose the connection to MQTT.

**Workaround:** Restart the Predix Machine Docker container to restore the connection.

## Deprecated

### **H2 Database**

H2 database has been deprecated in Predix Machine 17.2. There is no migration path if you have data stored in an H2 database, but you can continue to use H2. By default, the Data Store and Forward service is now configured to use Tape, and it is recommended that you switch to Tape.

### **HTTP River Service**

The HTTP River Service is replaced with the Event Hub Service.

See [Event Hub River Service \(page 85\)](#).

### **Windows Support**

Windows is no longer a supported operating systems for production environments.

### **JDK 7**

JDK 7 is no longer supported. Upgrade to JDK 8.

## Known Issues

This release has the following known issues:

### **Move Event not Propagated on Mac**

On Mac, when moving Docker images to the bootstrap mount directory, the "move" event is not propagated through to the bootstrap container, which causes bootstrap to ignore any files moved to its mounted directory.

Additionally, when copying Docker images, the "copy" event does not always propagate, which results in bootstrap not always picking up copied files.

### **OpenVPN in Docker**

If a host has two IPV6 addresses (Local and Global), Predix Machine may ignore the Global IPV6, which causes the On-demand Event service to not work.

**Workaround:** Disable the Local IPv6 address so there is no conflict.

## Docker Images

When Docker bootstrap creates the application container (for example, an analytics container), it creates a folder for it under the mounted directory `container/<folder>`, which may later be used by the application container).

When bootstrap receives a Delete Application Container command from Edge Manager, it deletes the application container, but does not delete the corresponding folder. If the application container is deployed again, a new folder will be created for the new container. Thus, if deleting the application container and deploying a new application container is done numerous times, many extraneous folders will remain on `<mounted dir>/containers/<folder>`.

This may cause out of storage problems for a device that is used for a long time out in the field.

## WebSocket River failed to send data

Occasionally, WebSocket River will fail to send data when the network switches from no proxy to a network that uses proxy.

**Workaround:** If this happens, you must restart the WebSocket River bundle.

## Predix Machine Agent container upgrade

Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#).

**Workaround:** You can upgrade by pushing a new Docker image through Edge Manager.

## Third-party bundle migration

Third-party bundles have been migrated to later versions. Open your image in the Eclipse editor for migrating bundles to later versions if required.

## OPC-UA Adapter exceptions

In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcua.security.mode` property to `BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## Export Control

Predix Machine is classified for export control and uses ECN 5D002.

## *Predix Machine Version 17.1.3*

These are the changes, enhancements, and known and resolved issues for Predix Machine 17.1.3.

You can download the new packages for Predix Machine 17.1.3 from: <https://artifactory.predix.io/artifactory/PREDIX-EXT/predix-machine-package/predixmachinesdk/17.1.3/predixmachinesdk-17.1.3.zip>.

## Important Announcements

### Removal of OAuth Enrollment for Edgemanager and Predix Machine

Due to a recently identified security issue with OAuth enrollment, support for OAuth-based device enrollment in Edge Manager is being discontinued effective **August 22, 2017**. If you already have devices in Edge Manager that were originally enrolled using OAuth authentication, they will continue to function normally. However, if you have devices added to (but not yet enrolled in) Edge Manager with OAuth as the specified method of enrollment, you will be required to delete those devices.

You then have to add the devices to US-West Edge Manager, using [certificate enrollment](#).

In preparation for US-East retirement on September 15, 2017, you will not be able to add new devices to US-East Edge Manager tenants starting August 22nd due to a security issue.

Please contact [Predix Support](#) if you have more questions or concerns.

### US-East ASV Edge Manager Retirement

Effective September 15, 2017, the Edge Manager instance in US-East will be retired. In preparation for the retirement, you will not be able to add new devices to Edge Manager tenants starting August 22nd due to a security issue.

Please contact [Predix Support](#) if you have questions, or follow these [instructions](#) to get access to US-West EdgeManger.

## Enhancements

This release has the following enhancements.

### New Spillway configuration property

There is a new (optional) configuration property in `com.ge.dspmicro.hoover.spillway-0.config` to set the length of time, in seconds, the Spillway waits for a response from Data River.

See [Configuring the Hoover Service \(page 141\)](#).

### New Modbus configuration property

There is a new (optional) configuration property in `com.ge.dspmicro.machineadapter.modbus-0.config` to allow Modbus Adapter to read a block of tags, or nodes, from the server in a single request instead of a request per tag.

See [Configuring the Modbus Machine Adapter \(page 145\)](#).

### **New script to clear Predix Machine setup**

There is a new utility script (`ResetContainer.sh`) available in the `<Predix Machine runtime container location>/security` folder for resetting a Predix Machine container to a pre-run state. This will clear generated keystores, password, encrypted fields, as well as remove enrollment information.

## **Resolved Issues**

These are the resolved issues included in Predix Machine 17.1.3.

### **Exception thrown when switching a WebSocket River parameter**

The issue where switching the `com.ge.dspmicro.websocketriver.send.tag.augment` WebSocket River parameter from "false" to "true" caused the following exception:

```
Exception while augmenting tag, reverting to original tagName  
xxxxxxx. ExceptionL null);
```

is fixed.

### **Docker script not always executable**

The issue where the `docker_start_predixmachine.sh` script was not always set to be executable from the Predix Machine SDK has been resolved.

### **Building sample-apps fails**

The issue where building the sample-apps fails because the build is attempting to pull from an inaccessible Artifactory location has been fixed.

### **Error logging added in OPA-UA Adapter write**

Previously, no exceptions were thrown in the write back function for OPC-UA for error conditions, which caused some exceptions to be missed. Error logging has been added to catch error condition exceptions.

### **Number of VPN client logs is reduced**

Previously, if the VPN service was stopped, exceptions were generated every three seconds. This exception is now managed and a meaningful message is returned.

## **Known Issues**

This release has the following known issues:

### **Predix Machine does not reconnect to Mosquitto**

Restarting a Mosquitto (MQTT) Docker container causes the Predix Machine Docker container to lose the connection to MQTT.

**Workaround:** Restart the Predix Machine Docker container to restore the connection.

### **WebSocket River failed to send data**

Occasionally, WebSocket River will fail to send data when the network switches from no proxy to a network that uses proxy.

**Workaround:** If this happens, you must restart the WebSocket River bundle.

### **Predix Machine Agent container upgrade**

Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#).

**Workaround:** You can upgrade by pushing a new Docker image through Edge Manager.

### **Upgrading devices from Predix Machine 16.4.x to 17.x.x**

For devices you want to upgrade from Predix Machine 16.4.x to 17.1.x, a specific `install.sh` script is required and is not part of the generated container.

**Workaround:** Contact Predix Machine product management to request this install script.

### **Third-party bundle migration**

Third-party bundles have been migrated to later versions. Open your image in the Eclipse editor for migrating bundles to later versions if required.

### **OPC-UA Adapter exceptions**

In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcua.security.mode` property to `BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## **Licensing**

You receive a non-production license key, which expires on 1/31/2019, when you download the Predix Machine SDK 17.1.3. To obtain a production license key, contact Predix Machine product management.

## **Export Control**

Predix Machine is classified for export control and uses ECN 5D002.

## *Predix Machine Version 17.1.2*

These are the enhancements, and known and resolved issues for Predix Machine 17.1.2.

## Enhancements

### Configure Modbus for handling null, or no data

There is a new property in the `com.ge.dspmicro.machineadapter.modbus-0.config` file that allows you to configure the way Modbus handles null, or no data.

Property	Description	Type	Default Value
<code>com.ge.dspmicro.machineadapter.modbus.nullDataResponseBehavior</code>	When set to <code>false</code> , an exception is thrown on null data.  If set to <code>true</code> , null datapoints are retained and the quality is set to "BAD."	Boolean	false

## Resolved Issues

### Exceptions on slower devices

The issue where the Device Details service did not map all the component service states, which caused exceptions on slower devices, has been resolved.

### VPN log not rolled

The issue where the VPN log is not rolled after reaching the size and time interval threshold has been resolved.

### get\_log command sent from Edge Manager failing with error

On the Mac OS, the Desktop folder is capitalized, and the `get_log` command was changing the case of the letter "D" in Desktop to lowercase "d", which caused an error, similar to the following:

```
2017-05-17 13:52:04,233[pool-3-thread-1] |INFO|
com.ge.dspmicro.cloud.gateway.impl.GatewayService|70-
com.ge.dspmicro.cloud-gateway-17.1.1|Status received with
message: "Log path ".../desktop/Demo/predixmachinesdk-17.1.1/
utilities/containers/PredixMachine-provision-17.1.1/logs/
machine/machine.log" not allowed.
```

This has been fixed.

### Start time not displayed when Docker container is updated

The issue where the start time was not displayed in Edge Manager when a Docker container is updated and deployed is resolved.

### Deployment timing out

The issue where sometimes deployment status was not sent to Edge Manager, making the deployment time out has been resolved.

### Error in Machine log after OAuth enrollment

The issue where the following error was logged after OAuth enrollment has been resolved.

```

May 16 16:25:24 <device> start_predixmachine.sh[22070]: #0
INFO    > Bundle with id #62
        com.ge.dspmicro.predix-connectivity, <version> was
        started.May 16 16:25:24 <device> start_predixmachine.sh[22070]:
2017-05-16 16:25:24,008[FW Buff
        Logger]|INFO|com.prosyst.mbs.system.bundle|22-
com.prosyst.mbs.system.bundle-1.0.0|Bundle with
        id #62 com.ge.dspmicro.predix-connectivity, <version> was
        started.May 16 16:25:24 <device> start_predixmachine.sh[22070]:
2017-05-16 16:25:24,329[FW Buff
        Logger]|INFO|com.prosyst.mbs.system.bundle|22-
com.prosyst.mbs.system.bundle-1.0.0|Bundle with
        id #63 com.ge.dspmicro.predixcloud-identity, <version> was
        started.May 16 16:25:24 <device> start_predixmachine.sh[22070]:
#0 INFO    > Bundle with id #63
        com.ge.dspmicro.predixcloud-identity, <version> was
        started.May 16 16:25:26 <device> start_predixmachine.sh[22070]:
2017-05-16 16:25:26,081[EventAdmin
        Asynch Priority Dispatcher 49 (Bundle
        6)]|WARN|com.prosyst.mbs.osgi.eventadmin|6-
com.prosyst.mbs.osgi.eventadmin-1.1.1001|[Event
        Admin] Error delivering event to

        com.ge.dspmicro.cloud.gateway.impl.GatewayService@bddb7fMay 16
16:25:26 <device> start_predixmachine.sh[22070]:
        java.lang.NullPointerExceptionMay 16 16:25:26 <device>
start_predixmachine.sh[22070]: at

        com.ge.dspmicro.cloud.gateway.impl.GatewayService.handleEvent(GatewayService.java:
16 16:25:26 <device> start_predixmachine.sh[22070]: at

        com.prosyst.mbs.impl.services.event.HandlerWrapper.deliverEvent(HandlerWrapper.java:
16 16:25:26 <device> start_predixmachine.sh[22070]: at

        com.prosyst.mbs.impl.services.event.EventAdminImpl.deliverEvent(EventAdminImpl.java:
16 16:25:26 <device> start_predixmachine.sh[22070]: at

        com.prosyst.mbs.impl.services.event.ASynchQueue.run(ASynchQueue.java:102)May
16 16:25:26 <device> start_predixmachine.sh[22070]: at

        com.prosyst.util.impl.tpt.threadpool.PEA.run(PEA.java:11)May 16
16:25:26 <device> start_predixmachine.sh[22070]: at
        java.security.AccessController.doPrivileged(Native
        Method)May 16 16:25:26 <device> start_predixmachine.sh[22070]:
at

```

```

com.prosyst.util.impl.tpt.threadpool.ExecutorImpl.run(ExecutorImpl.java:194)
[4:com.prosyst.mbs.core.threads:1.1.0]May 16 16:25:26
<device> start_predixmachine.sh[22070]: 2017-05-16
16:25:26,263[Component
  Resolve Thread (Bundle
  28)]|INFO|com.prosyst.mbs.osgi.eventadmin|6-
com.prosyst.mbs.osgi.eventadmin-1.1.1001|[Event
  Admin] Priority Handler registered:
com.ge.dspmicro.device.techconsole Service Registration
  Registration state is: registered. ID: 79 Ranking: 0 with
  priority
  49

```

## Known Issues

This release has the following known issues:

- Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#). You can upgrade by pushing a new Docker image through Edge Manager.
- For devices you want to upgrade from Predix Machine 16.4.x to 17.1.x, a specific `install.sh` script is required and is not part of the generated container. Contact Predix Machine product management to request this install script.
- Third-party bundles have been migrated to later versions. Open your image in the Eclipse editor for migrating bundles to later versions if required.
- In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcu.security.mode` property to `BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## Export Control

Predix Machine is classified for export control and uses ECN 5D002.

## *Predix Machine Version 17.1.1*

These are the enhancements, and known and resolved issues for Predix Machine 17.1.1.

### Enhancements

The following enhancements are in Predix Machine 17.1.1:

#### **Third Party software for serial communication support**

The jamod modbus library has been updated to j2mod. To implement the new libraries:

1. Go to <http://central.maven.org/maven2/com/fazecast/jSerialComm/1.3.11/jSerialComm-1.3.11.jar> and download the tested version of the `jSerialComm` jar library.
2. Place the `jSerialComm-1.3.11.jar` library in the Predix Machine container at `<PredixMachineDirectory>/machine/lib/jserialcomm`.

 **Note:** Create the `jserialcomm` directory if it does not already exist.

Various native libraries are embedded in the `jserialcomm` jar and will be picked up as needed. Linux `x86_64` has been tested and verified to work.

 **Note:** The Windows platform is not supported at this time.

### **Bootstrap starts the Predix Machine container in debug mode**

Now, when a Predix Machine container is downloaded or copied to the bootstrap mount directory, the bootstrap starts in debug mode if the name of the container matches the string `*_debug*`. The bootstrap includes `MACHINE_DEBUG_ENABLED` and `PREDIX_MACHINE_NOCHANGE_PW` properties, set to `true` to indicate to Predix Machine that it should start in debug mode.

Predix Machine also exposes port 8000 for debug connections. These debug settings allow Eclipse to attach to the Java VM for debugging. Additionally, a password change upon first login to the Predix Machine Web Console is not required.

### **The bootstrap is always running**

The bootstrap is always running, unless it is stopped manually by using Docker commands.

### **Bootstrap logs the installation**

Predix Machine bootstrap now logs Docker container installations. When Predix Machine finishes writing the log file, it uploads the log file to the `<Predix_Machine_Container_Location>/appdata/packageframework` directory.

### **Bootstrap container can be updated**

When deploying a new bootstrap container from Edge Manager or from the bootstrap mounted directory, the bootstrap updates itself. The installation watcher script in the bootstrap checks to see if there are any pending downloads to transfer to the bootstrap installer.

### **Alpine base images updated**

The Alpine base images have been updated from version 3.3 to version 3.5.

## Resolved Issues

The following issues are resolved in this release:

### **Predix Machine Running on Docker**

The issue where running Predix Machine on Docker on a Raspberry Pi device did not work as expected is resolved.

The issue where running Predix Machine on Docker using OpenJDK 8 did not work as expected has been resolved.

The issue where the Docker bootstrap could not start Predix Machine in some cases has been resolved.

### **Predix Machine goes into a restarting loop**

The issue where, at times, Predix Machine restarted, and went into a restarting loop, before returning a task acknowledging status to the cloud is resolved.

### **Devices not saving keys/properties**

The issue where devices were not saving created keystores upon startup after a power failure is resolved.

### **Error when sending a payload of greater than 8 K**

The issue that was causing an error when a payload greater than 8 KB, with added attributes, was sent to WebSocket river has been resolved.

 **Note:** For data ingestion, the Time Series service has a 512 KB limit on the JSON payload.

### **OAuth syncs immediately after enrollment**

The issue where it took a while for an enrolled device to turn to an "online" status on the cloud after OAuth device enrollment has been resolved. Now, OAuth is synchronized immediately after enrolling the device with Predix Machine.

### **Spillway creates large number of threads**

A problem was fixed in the Spillway threading implementation, which prevents Predix Machine from running out of memory when there is a high load.

## Known Issues

This release has the following known issues:

- Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#). You can upgrade by pushing a new Docker image through Edge Manager.

- For devices you want to upgrade from Predix Machine 16.4.x to 17.1.x, a specific `install.sh` script is required and is not part of the generated container. Contact Predix Machine product management to request this install script.
- Third-party bundles have been migrated to later versions. Open your image in the Eclipse editor for migrating bundles to later versions if required.
- In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcua.security.mode` property to `BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## Export Control

Predix Machine is classified for export control and uses ECN 5D002.

## *Predix Machine Version 17.1*

These are the new features, enhancements, and known and resolved issues for Predix Machine 17.1.

 **Note:** US West users must use Predix Machine version 17.1.0 and later with Edge Manager.

## New Features

This release contains the following new features:

### Management Bus

Predix Machine Management Bus allows your custom applications to exchange commands and configurations with Predix Machine, without restricting you to using a particular language.

For more information, see [Management Bus \(page 265\)](#).

### Package Handler

Predix Machine Package Handler service allows you to download and install custom packages using your own installer, or if you are downloading large packages of information from Edge Manager to the device.

For more information, see [Package Handler Service \(page 190\)](#).

### Predix Connectivity VPN

Predix Connectivity provides a service for managing a VPN client to provide a secure connection to desired server destinations. The VPN service is integrated with Edge Manager, so that you can manage VPN on devices using Edge Manager.

A new `LogsHandler` interface is available so that you can upload a specific log file to Edge Manager using the `GetLog` command, as well as retrieve the file names of logs that can be given to the `GetLog` command.

For more information, see [Predix Machine VPN Service \(page 55\)](#).

## Enhancements

This release contains the following enhancements:

### Device Detail Service

The information provided by the Device Detail service has been enhanced to include Bluetooth and WiFi connection status; device information properties; device status properties; device properties; and edge alerts.

There is now support for generic fields in `proto` in Predix Machine to send to the cloud for device details.

The location for where the Device Detail service picks up vendor-provided JSON files is configurable.

See [Device Detail Service \(page 196\)](#)

## Known Issues

This release has the following known issues:

### Predix Machine running in a Docker container does not work as expected

Predix Machine running in a Docker container does not work as expected in the following cases:

- If you are using OpenJDK 8.

**Workaround:** Use OpenJDK 7.

- If Predix Machine is running in a Docker container on a Raspberry Pi device.

- Do not update a Predix Machine Agent container using the method described in [Upgrading a Predix Machine Container \(page 69\)](#). You can upgrade by pushing a new Docker image through Edge Manager.
- For devices you want to upgrade from Predix Machine 16.4.x to 17.1.x, a specific `install.sh` script is required and is not part of the generated container. Contact Predix Machine product management to request this install script.
- Third-party bundles have been migrated to later versions. Open your image in the Eclipse editor for migrating bundles to later versions if required.
- In the OPC-UA Adapter, if you set the value of the `com.ge.dspmicro.machineadapter.opcua.security.mode` property to

`BASIC256_SIGN` or `BASIC256_SIGN_ENCRYPT`, exceptions occur and you cannot establish a security channel.

## Resolved Issues

The issue where certain characters got converted (URI encoded) by the time they reached the WebSocket service, which caused issues with Time Series, has been resolved. For instance, a space was converted to '%20', a backslash to '%5C' and so on, and '%' is not an acceptable character for a Time Series attribute. Time Series attributes can contain only alphanumeric characters, periods (.), forward slashes (/), dashes (-), and underscores (\_).

For example, now `opcua-//abcd%20mno:p%5Cxyz-d%pqrs%20hijkl` is converted to `opcua-//abcd-mno-p-xyz-d-pqrs-hijkl`.

## Export Control

Predix Machine is classified for export control and uses ECN 5D002.

# Predix Machine Data Collection Configuration

## *Ingesting Data Using Modbus Adapter*

### *Configuration Guidelines for Predix Machine and Modbus Adapter*

These are the recommendations for how to configure Predix Machine (adapters, spillways, rivers) to send data using the WebSocket river.

A series of benchmark tests were run on a representative device with different numbers of data sources and spillway river combinations. Metrics were collected from each test to determine the optimal configuration for sending data to the Predix Time Series service using Predix Machine WebSocket River. The WebSocket River provides connectivity between a Predix Machine-enabled device and the Predix Time Series service in the Predix cloud. For more information about the WebSocket River service, see [WebSocket River \(page 79\)](#).

## Environment and Setup

The following setup was used for testing:

### Device

Dell 500 Gateway with the following configuration:

- 8 GB RAM
- 2 CPU 500 Hz

- Ubuntu 15.04
- Java 1.8

 **Note:** Heap configuration was left up to the JVM. The JVM makes intelligent choices about memory requirements based on the class of the server Predix Machine is installed on, which in turn is determined by the total available memory, the number of CPUs, and platform architecture (32-bit or 64-bit).

By default, the initial heap size is set to 128699584 and the maximum heap size is set to 2059193344.

## **Predix Machine**

The following Predix Machine components were used:

- Modbus Adapter
- Spillway
- Store and Forward
- WebSocket River

## **Modbus Simulator (diagslave)**

Used to simulate modbus data sources.

## **Simulator Instances**

Multiple instances of the simulator were run on VMs, which were on the same LAN as the test device.

## **Latency**

Latency of 40 ms was simulated for each modbus simulator to mirror a realistic scenario.

## **Modbus Adapter**

Each modbus adapter instance was configured to pull 22 values per second from the modbus simulator.

The 22 values constitute one subscription, and are transmitted by the WebSocket river to the Time Series service in a single payload.

## **Summary**

- Key metrics such as latency can be used to define a workable configuration for sensor data collection, followed by benchmark tests in the real or simulated environment to validate and fine tune the configuration
- Predix Machine was able to support the following configuration on the test device (Dell 5500 Gateway), while leaving CPU and memory resources to spare, which indicates that further scaling may be possible:

- 60 modbus adapter instances, each pulling 22 data points every second, with a total of 1,320 data points per second.
- 15 **Spillway > Store and Forward > WebSocket River** channels, each subscribing to data from four adapters.

## Factors That Influence the Configuration

The following factors can affect the data collection configuration:

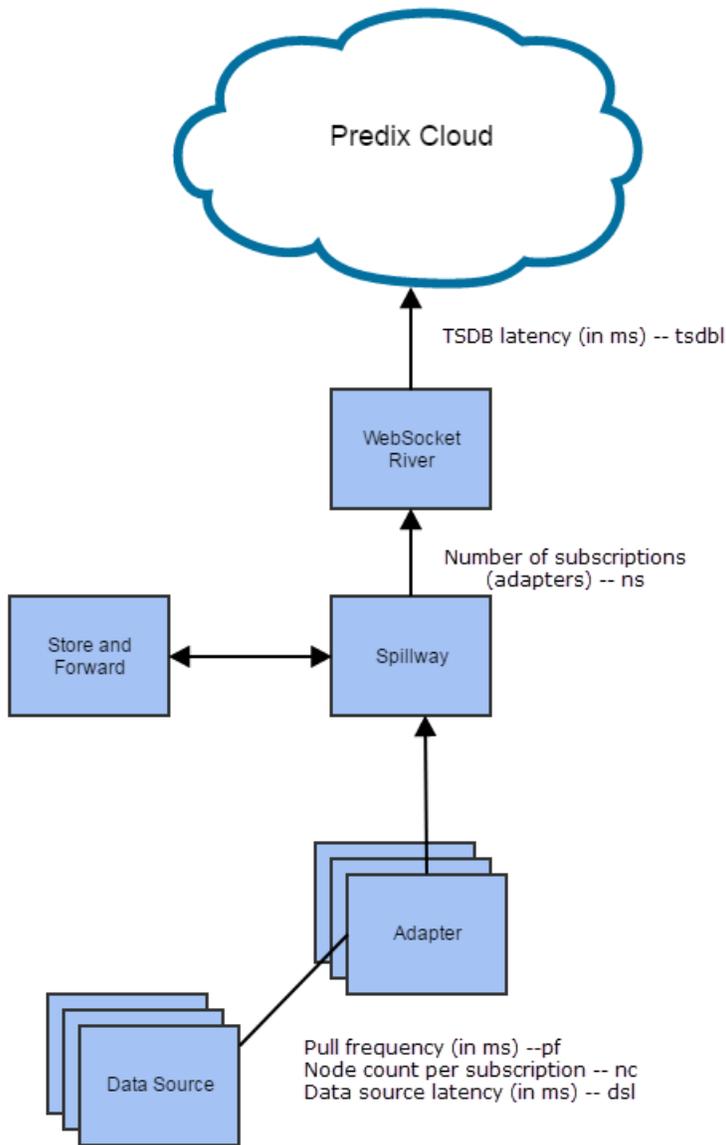
- Choice of adapter
- Number of tags or nodes
- Subscription or pull mode
- Collection interval
- Whether the Data Store and Forward service is used
- Latency between the adapter and data source
- Latency between the device and cloud service
- Network bandwidth between device and cloud service
- Compute resources on the device, such as RAM, number of CPUs, and CPU speed

## Configuration Conditions

The following image shows the data collection configuration for Predix Machine.

 **Note:** The following conditions must be met for your configuration:

- $\text{tsdbl} * \text{ns} < \text{pf}$
- $\text{dsl} * \text{nc} < \text{pf}$



## What to Observe

While testing a specific data collection configuration, monitor the following conditions to ensure acceptable performance.

### Size of Store Forward Database

This should remain stable at all times and should not exhibit growth. Predix Machine logs the size of the database and the number of records. The number of records should be roughly the same at each compression, and ideally small in size (<10).

### Time Series Database

Look for consistent volume of data/unit of time and low jitter or variance in the data point timestamps. For example, at a polling rate of 2 seconds, ensure that there are close to 1800 samples for each data point per hour, and that they are approximately 2000 ms apart, within a tolerable limit.

### **Heap Utilization by the JVM**

Ensure garbage collection is not excessive, and adjust the heap allocation to the JVM if necessary.

### **Load Average on the Device**

Ensure that this remains equal to, or less than, the number of CPUs. Overall CPU utilization on the device should remain acceptable for the operation of the device and other applications running on it.

### **No Significant Swapping**

Ideally, the operating system should have sufficient RAM to ensure that no swapping to disk is required.

## *Test Configuration*

The configurations used for testing included configurations for Modbus Adapter, Hoover Spillway, WebSocket River, and Store and Forward.

Each configuration uses the following files, where  $\langle n \rangle$  is the ordinal number of the channel (0, 1, 2, 3, and so on).

#### **com.ge.dspmicro.machineadapter.modbus- $\langle n \rangle$ .xml**

Used to configure connection nodes and subscriptions.

#### **com.ge.dspmicro.machineadapter.modbus- $\langle n \rangle$ .config**

Used to configure the name, description, and location of the `com.ge.dspmicro.machineadapter.modbus- $\langle n \rangle$ .xml` file.

#### **com.ge.dspmicro.hoover.spillway- $\langle n \rangle$ .config**

Used to specify the name and description of the Spillway, list the data subscribers, and specify the name of the destination WebSocket river. Each `com.ge.dspmicro.hoover.spillway- $\langle n \rangle$ .config` subscribes to a unique `datasubscription`, for example:

```
com.ge.dspmicro.hoover.spillway.dataSubscriptions=[
  "Analytics_Subscription_5",
]
```

It also sends data to a unique WebSocket river destination, for example:

```
com.ge.dspmicro.hoover.spillway.destination="WS Sender Service
5"
```

If Store and Forward is enabled, it uses a unique one:

```
com.ge.dspmicro.hoover.spillway.storeforward="DefaultStoreForward_5"
```

### **com.ge.dspmicro.websocketriver.send-<n>.config**

Links the machine configured with the WebSocket River to the Time Series service.

This file references the corresponding Spillway destination from the `com.ge.dspmicro.hoover.spillway-<n>.config` file, for example:

```
com.ge.dspmicro.websocketriver.send.river.name="WS Sender  
Service 5"
```

This file specifies the Time Series service endpoint and Predix zone ID, for example:

```
com.ge.dspmicro.websocketriver.send.destination.url="wss://  
gateway-<host>.predix.io/v1/stream/messages"  
com.ge.dspmicro.websocketriver.send.header.zone.value="nnnnnnn-7479-4d5d-  
alaf-85720bf4c3ad"
```

### **com.ge.dspmicro.storeforward-<n>.config (only if Store and Forward is configured)**

Used to specify the intervals at which the callback reads and forwards data. Each Store and Forward config file uses a unique name, for example:

```
com.ge.dspmicro.storeforward.name="DefaultStoreForward_4"
```

## *Test Observation Data*

Test observations included data about JVM heap size, average percentage of CPU, and memory utilization, among others.

- Tests were performed on the Predix Machine data collection configuration with 4, 8, 12, 16, 24, 32, and 60 data sources (modbus adapters).
- Tests had a minimum duration of one hour.

### Analysis of Time Series Data Points After Using Modbus Adapter to Send Data

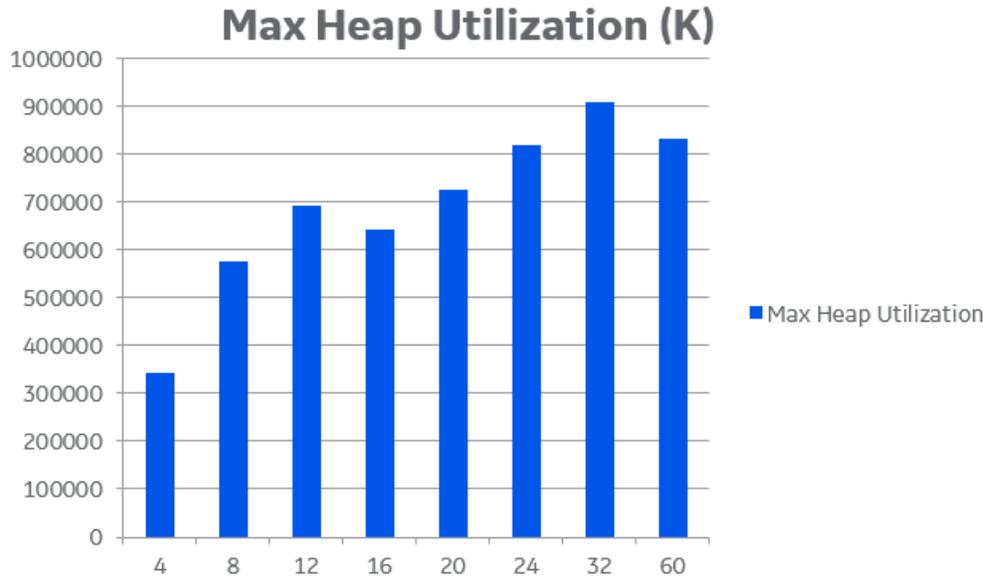
The data below was gathered from the Time Series database after using Modbus Adapter to send data to the Time Series service, and represents one out of 22 nodes for each adapter.

Sources	Min Gap (ms)	Max Gap (ms)	Average Gap (ms)	Total points	Interval Minutes	Average Count Per Sec	Average Count Per Source Sec	Total points per Sec
4	905	1309	999.996	51992	217	3.99	0.998	87.78
8	579	1536	999.996	60803	127	7.98	0.998	175.56
12	902	1537	999.999	177808	247	12	1	264
16	903	1450	999.992	173354	173	16.7	1.044	367.4
24	895	1537	999.998	217344	151	23.99	1	527.78
32	898	1537	999.998	360681	188	31.98	0.999	703.56
60	896	2148	1000.221	215494	60	59.86	0.998	1316.92

## JVM Heap Utilization

This shows the JVM Heap utilization that was observed during testing.

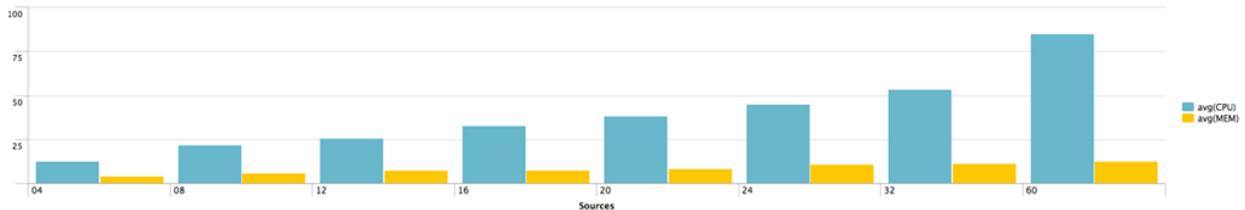
Sources	Max Heap
4	343040K
8	576512K
12	692224K
16	641536K
20	724480K
24	817664K
32	909312K
60	832512K



### Average Percentage of CPU and Memory Utilization by Predix Machine

This shows the average percentage of CPU and memory utilization by Predix Machine for each test performed.

The CPU utilization is per CPU, so the effective utilization is half of what is reported here.

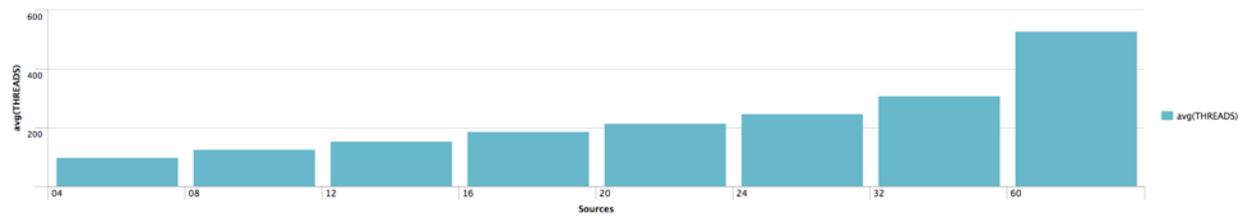


Sources	avg CPU	avg MEM	avg THREADS
04	12.840808	4.401253	98.133705
08	22.061792	5.898821	126.893868
12	25.923952	7.385928	155.318862
16	33.036590	7.488099	188.204263
20	38.231962	8.427262	214.706228
24	45.208085	11.030426	248.191489
32	53.688045	11.592246	309.702746

Sources	avg CPU	avg MEM	avg THREADS
60	85.402262	12.778733	529.194570

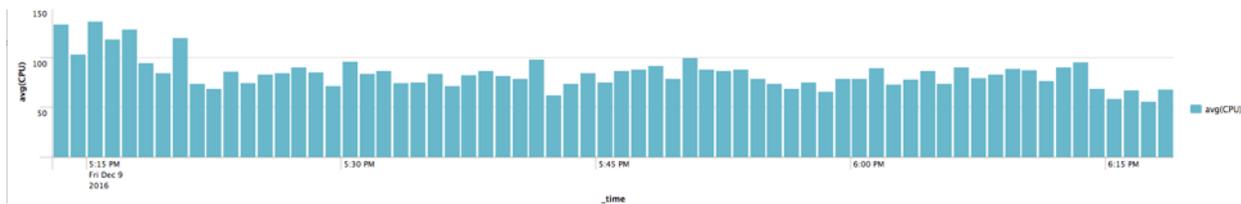
### Average Number of Threads

The following image shows the average number of threads used by Predix Machine for each test.



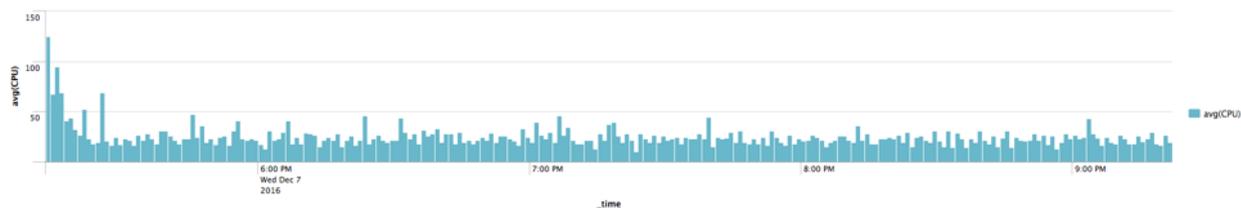
### Average CPU Usage for 60 Adapter Scenario

The following image shows the average percentage of CPU used by Predix Machine for the 60 adapter scenario, aggregated over every minute.



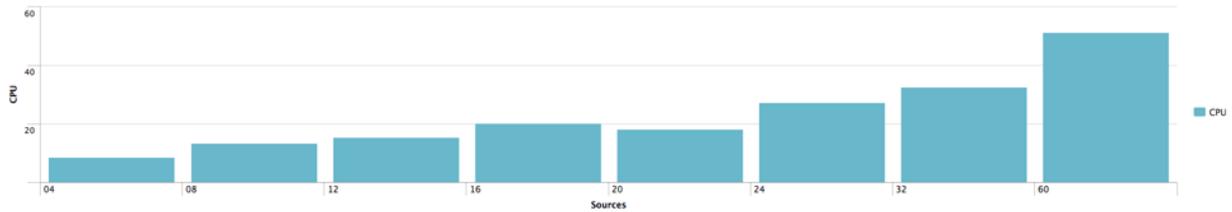
### Average CPU Usage for 12 Adapter Scenario

The following image shows the average percentage of CPU used by Predix Machine for the 12 adapter scenario, aggregated over every minute.



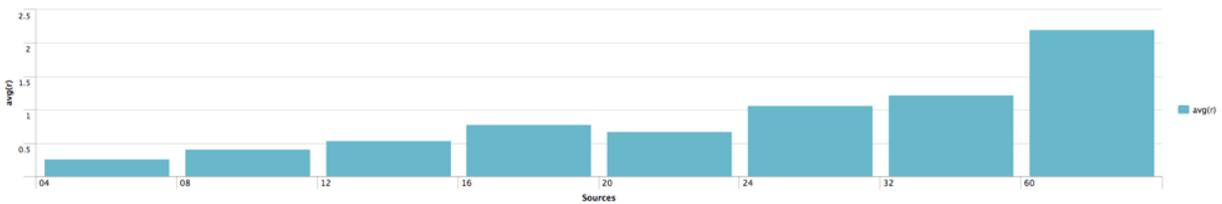
### Average CPU Usage on the Device

The following image shows the average percentage of CPU (sy+us) on the device for each scenario, as reported by vmstat.



## Average CPU Load by Predix Machine

The following image shows the average CPU load by Predix Machine for each scenario, as reported by vmstat.



## *Ingesting Data Using MQTT Adapter*

### *Configuration Guidelines for Predix Machine and MQTT Adapter*

These are the guidelines for configuring Predix Machine data ingestion using the MQTT Adapter to send data to the Time Series service using the WebSocket River.

A series of benchmark tests were run on a representative device with different numbers of channels and Spillway river combinations. Metrics for CPU, memory utilization, and heap utilization were collected from each test to determine the optimal configuration for using the MQTT Adapter to send data to the Predix Time Series service using Predix Machine WebSocket River.

### **Environment and Setup**

The following setup was used for testing:

#### **Device**

Dell 5500 Gateway using the following configuration:

- 8 GB RAM
- 2 CPU 500 Hz
- Ubuntu 15.04
- Java 1.8

 **Note:** Heap configuration was left up to the JVM. The JVM makes intelligent choices about memory requirements based on the class of the server Predix Machine is installed on, which in turn is determined by the total available memory, the number of CPUs, and platform architecture (32-bit or 64-bit).

By default, the initial heap size is set to 128699584 and the maximum heap size is set to 2059193344.

### **Predix Machine 17.1.1**

The following Predix Machine components were used:

- MQTT Adapter
- Spillway
- With and without Store and Forward.
- WebSocket River

### **MQTT**

- MQTT broker running on a separate VM on the same network as Predix Machine
- MQTT Java client running on the same VM as the broker

The client can be used to send datapoints (tags) in the Predix Machine PDataValue format at specific intervals.

### **Summary**

- Predix Machine was scaled using multiple channels **MQTT Adapter > Spillway > WebSocket River**.
- Each channel handled up to 2750 tags, resulting in a Time Series payload size of 500,744 bytes.
- Each test was run for two hours (7200 samples of each tag).
- Predix Machine was able to support the following configuration on the test device (Dell 5500 gateway):
  - Three channels, each receiving 2750 tags every second, for a total of 8250 tags per second with Store and Forward enabled.
  - Five channels, each receiving 2500 tags every second, for a total of 12500 tags with Store and Forward disabled.

### *What to Observe*

While testing the MQTT configuration, monitor the following conditions to ensure acceptable performance.

#### **Size of Store Forward Database**

This should remain stable at all times and should not exhibit growth. Predix Machine logs the size of the database and the number of records. The number of records should be roughly the same at each compression, and ideally small in size (<10).

### Heap Utilization by the JVM

Ensure garbage collection is not excessive, and adjust the heap allocation to the JVM if necessary.

### Load Average on the Device

Ensure that this remains equal to, or less than, the number of CPUs. Overall CPU utilization on the device should remain acceptable for the operation of the device and other applications running on it.

### No Significant Swapping

Ideally, the operating system should have sufficient RAM to ensure that no swapping to disk is required.

## Channel Configuration

The channel configurations for testing included configurations for MQTT Adapter, Hoover Spillway, WebSocket River, and Store and Forward (if used).

Each channel configuration uses the following files, where <n> is the ordinal number of the channel (0, 1, 2, 3, and so on).

### com.ge.dspmicro.machineadapter.mqtt-<n>.xml

Used to define connection nodes and subscriptions. Each file subscribes to a unique topic, for example:

```
<dataNodeConfigs>
  <dataNode name="Node-1" topic="TestTopic5" qos="2"
    serializedData="true" description="TestTopic 1 with Qos 0" />
</dataNodeConfigs>
```

The `datasubscription` value is also unique, for example:

```
<dataSubscriptionConfigs>
  <dataSubscriptionConfig name="Analytics_Subscription_5">
    <nodeName>Node-1</nodeName>
  </dataSubscriptionConfig>
</dataSubscriptionConfigs>
```

### com.ge.dspmicro.machineadapter.mqtt-<n>.config

Used to configure the name, description, and location of the `com.ge.dspmicro.machineadapter.mqtt-<n>.xml` file, for example:

```
com.ge.dspmicro.machineadapter.mqtt.configFile="configuration/
machine/com.ge.dspmicro.machineadapter.mqtt-<n>.xml"
```

This file also points to the MQTT Broker, for example:

```
com.ge.dspmicro.machineadapter.mqtt.broker="tcp://
<ip_address>:<port>"
```

### **com.ge.dspmicro.hoover.spillway-<n>.config**

Used to specify the name and description of the Spillway, list the data subscribers, and specify the name of the destination WebSocket river. Each `com.ge.dspmicro.hoover.spillway-<n>.config` subscribes to a unique `datasubscription`, for example:

```
com.ge.dspmicro.hoover.spillway.dataSubscriptions=[
  "Analytics_Subscription_5",
]
```

It also sends data to a unique WebSocket river destination, for example:

```
com.ge.dspmicro.hoover.spillway.destination="WS Sender Service
5"
```

If Store and Forward is enabled, it uses a unique one:

```
com.ge.dspmicro.hoover.spillway.storeforward="DefaultStoreForward_5"
```

### **com.ge.dspmicro.websocketriver.send-<n>.config**

Links the machine configured with the WebSocket River to the Time Series service.

This file references the corresponding Spillway destination from the `com.ge.dspmicro.hoover.spillway-<n>.config` file, for example:

```
com.ge.dspmicro.websocketriver.send.river.name="WS Sender
Service 5"
```

This file specifies the Time Series service endpoint and Predix zone ID, for example:

```
com.ge.dspmicro.websocketriver.send.destination.url="wss://
gateway-<host>.predix.io/v1/stream/messages"
com.ge.dspmicro.websocketriver.send.header.zone.value="nnnnnnn-7479-4d5d-
alaf-85720bf4c3ad"
```

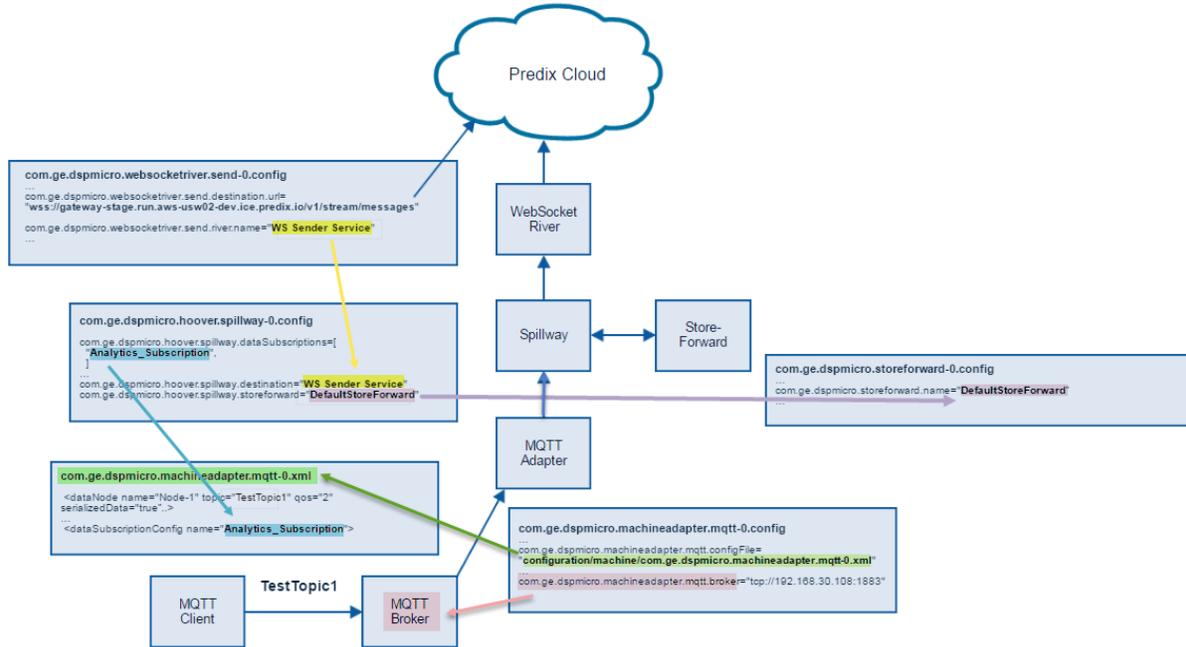
### **com.ge.dspmicro.storeforward-<n>.config (only if Store and Forward is configured)**

Used to specify the intervals at which the callback reads and forwards data. Each Store and Forward config file uses a unique name, for example:

```
com.ge.dspmicro.storeforward.name="DefaultStoreForward_4"
```

The following diagram illustrates the relationships between the configuration files, and the way they work together.

Figure: Channel Configuration



## MQTT Messages

The MQTT test client sent all the tags (2750) in a single message.

```
[ {
  "address": "com.intel.controller://jobId:502/1/3/rotor_name",
  "datatype": "INTEGER",
  "name": "INTNODE_07500",
  "category": "REAL",
  "value": "782107500",
  "timestamp": "1493944487821",
  "quality": "Good (0)"
}, {
  "address": "com.intel.controller://jobId:502/1/3/rotor_name",
  "datatype": "INTEGER",
  "name": "INTNODE_07501",
  "category": "REAL",
  "value": "782107501",
  "timestamp": "1493944487821",
  "quality": "Good (0)"
}, {
  "address": "com.intel.controller://jobId:502/1/3/rotor_name",
  "datatype": "INTEGER",
  "name": "INTNODE_07502",
```

```

    "category": "REAL",
    "value": "782107502",
    "timestamp": "1493944487821",
    "quality": "Good (0)"
  }...
}]

```

One client is used for each channel. Each client publishes on the topic subscribed to by the corresponding MQTT Adapter, for example, `TestTopic1`, `TestTopic2`, and so on. One message is published per second.

### *Analysis of Time Series Data Points After Using MQTT Adapter to Send Data*

Data was collected from the Time Series database and analyzed after sending data using the MQTT adapter.

In these tests, the timestamps were provided by the MQTT tool used to generate the data. As such, there were no packet delays in the data. It was verified at the end of each test that there were 7200 samples of each tag in the Time Series database. It was also verified that Predix Machine was not lagging while sending data.

### **JVM Heap Utilization**

The following table shows the heap utilization results for Predix Machine configured with  $x$  number of channels, and whether or not Store and Forward was used.

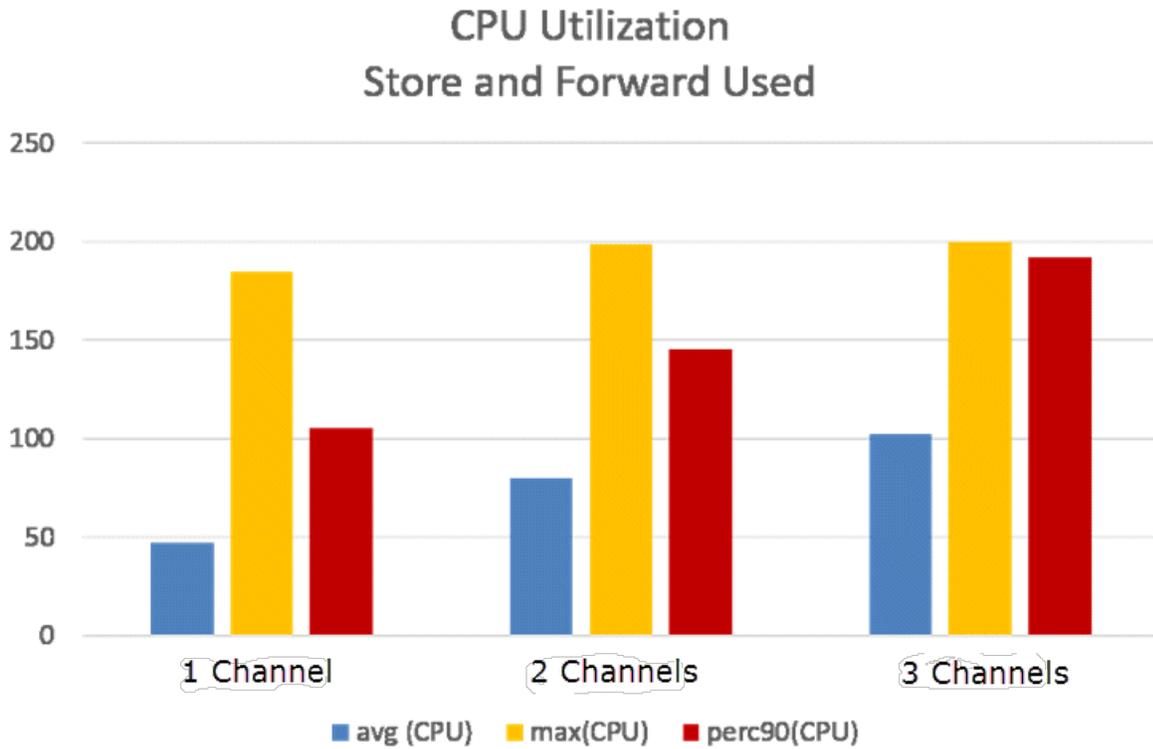
<b># of Channels Configured</b>	<b>Store &amp; Forward Used?</b>	<b>Max Heap</b>
1	Yes	746288K
2	Yes	783373K
3	Yes	1298104K
1	No	727819K
3	No	1177894K
5	No	2050740K

### **CPU Utilization**

Metrics were collected from each test to determine CPU utilization by Predix Machine when using MQTT Adapter to send data to the Time Series service.

### CPU Utilization with Store and Forward

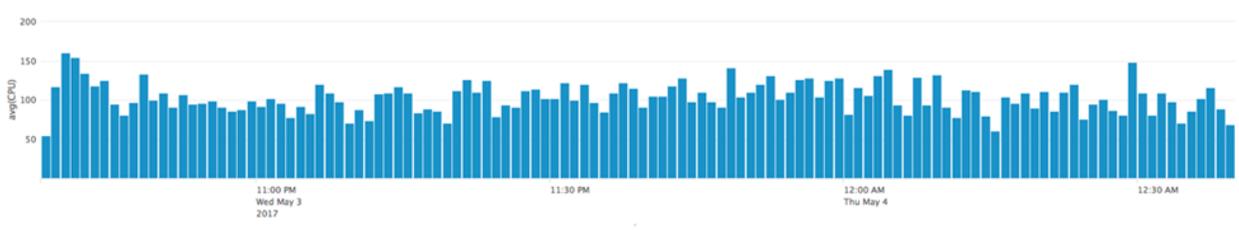
The following figure shows the average CPU utilization by Predix Machine configured with one, two, and three channels respectively, and using the Store and Forward service.



# of Channels	avg(CPU)	max(CPU)	perc90(CPU)
1	47.22	184.5	105.6
2	79.89	198.4	145.5
3	102.13	200.0	191.8

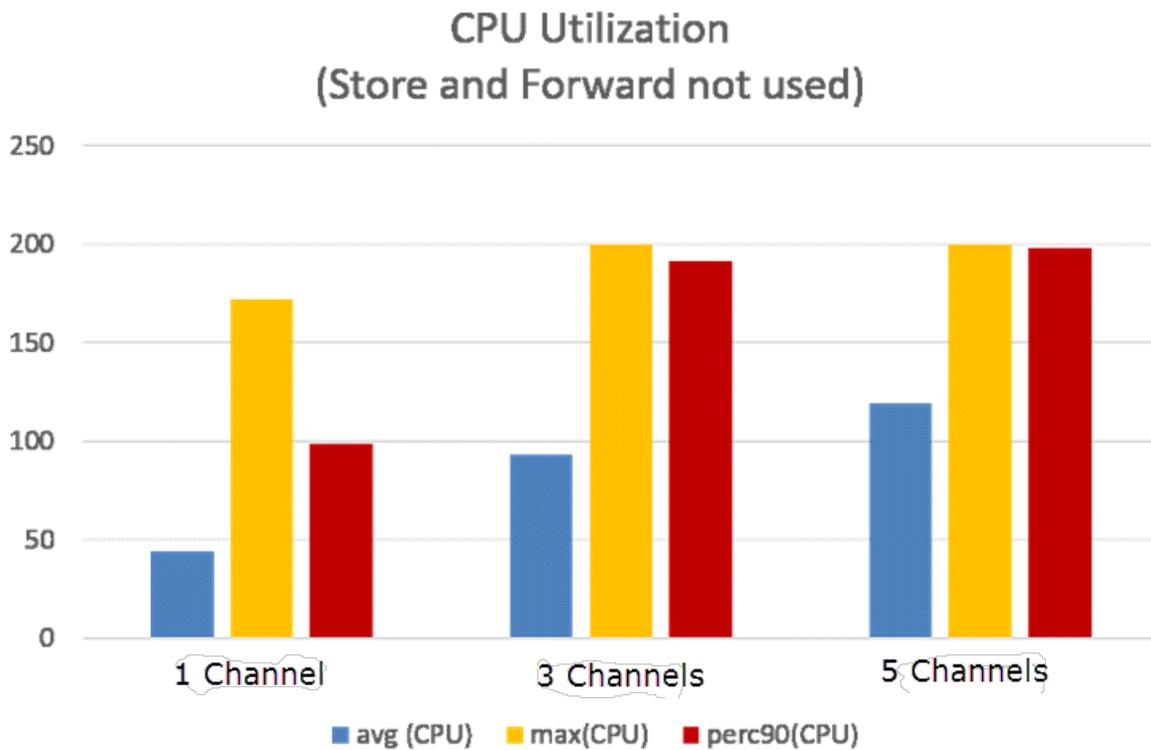
### Average CPU Profile with Store and Forward

The following graph shows the average CPU profile using Store and Forward, over a two-hour period, with three channels, and a total of 8,250 tags.



### CPU Utilization Without Store and Forward

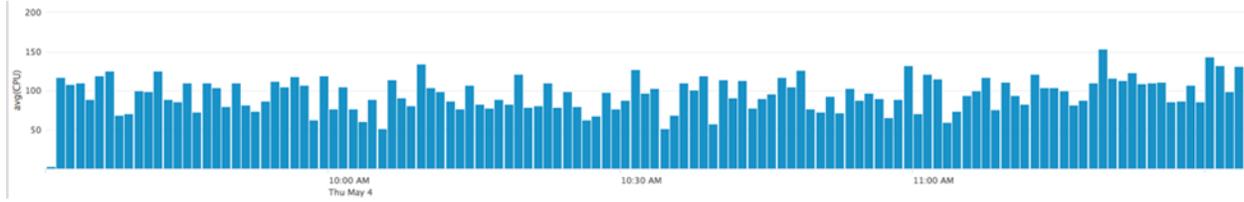
The following figure shows the average CPU utilization by Predix Machine configured with one, three, and five channels respectively and not using the Store and Forward service.



# of Channels	avg(CPU)	max(CPU)	perc90(CPU)
1	44.22	171.9	98.9
3	93.18	200.0	191.3
5	119.02	200.0	198.3

### Average CPU Profile Without Store and Forward

The following graph shows the average CPU profile when not using Store and Forward, over a two-hour period, with three channels, and a total of 7,500 tags.



The following graph shows the average CPU profile when not using Store and Forward, over a two-hour period, with five channels, and a total of 12,500 tags.



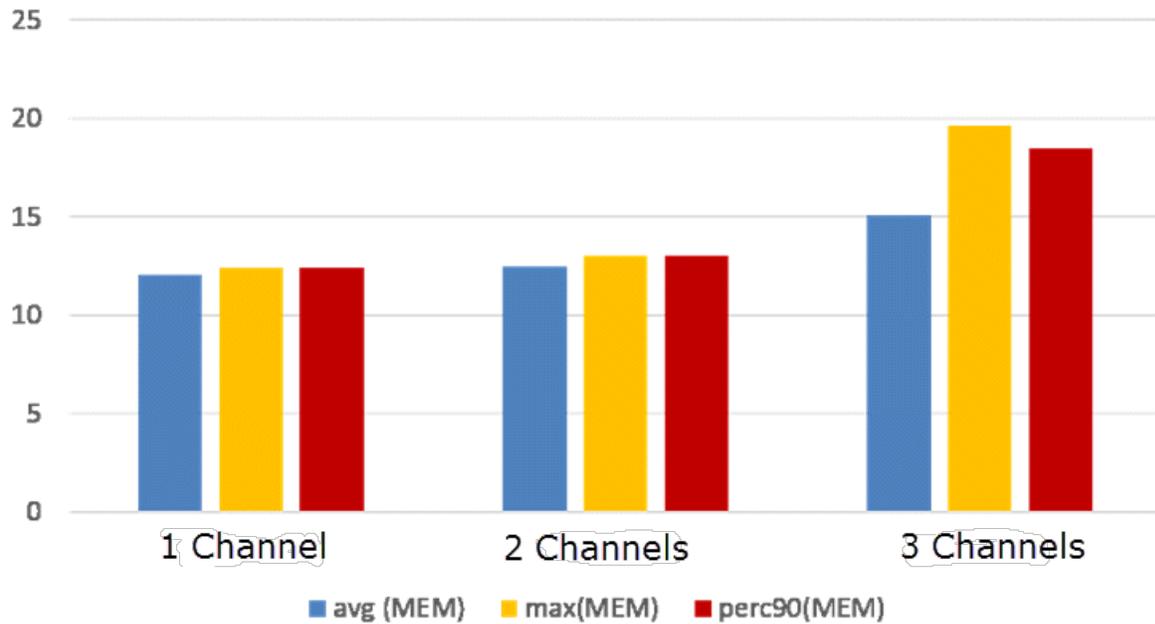
## Memory Utilization

Metrics were collected from each test to determine memory utilization by Predix Machine when using MQTT Adapter to send data to the Time Series service.

## Memory % Utilization with Store Forward

The following figure shows the average memory utilization by Predix Machine configured with one, two, and three channels respectively, and using the Store and Forward service.

## Memory Utilization (Store and Forward used)

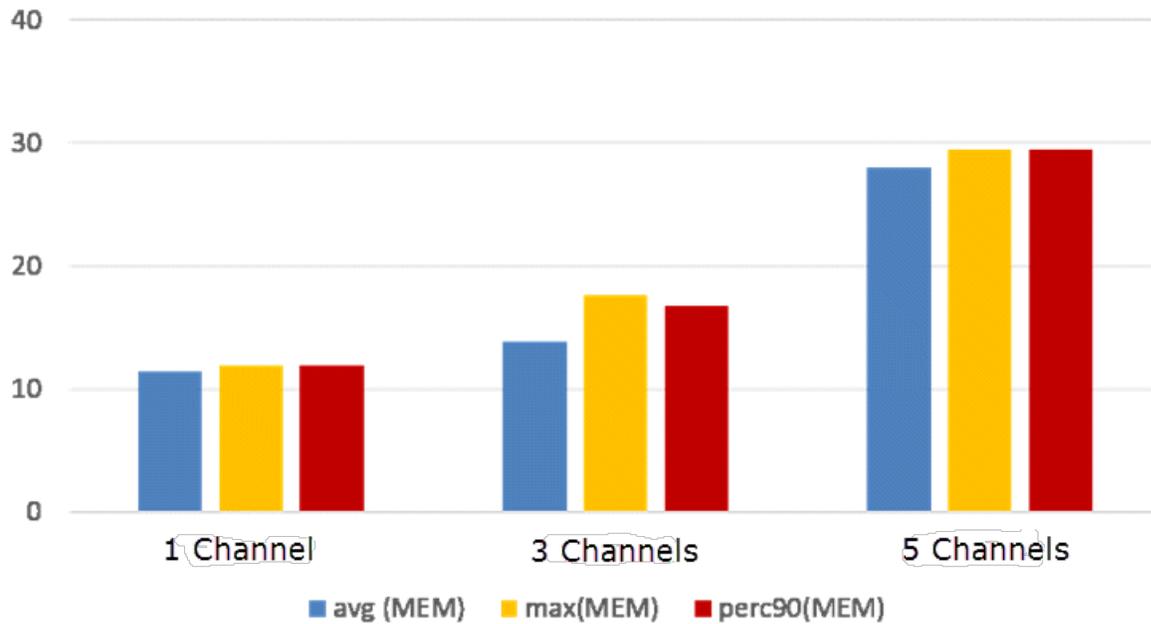


# of Channels	avg(MEM)	max(MEM)	perc90(MEM)
1	12.06	12.4	12.4
2	12.45	13.0	13.0
3	15.08	19.6	18.5

### Memory % Utilization Without Store Forward

The following figure shows the average memory utilization by Predix Machine configured with one, three, and five channels respectively, and not using the Store and Forward service.

### Memory Utilization (Store and Forward not used)



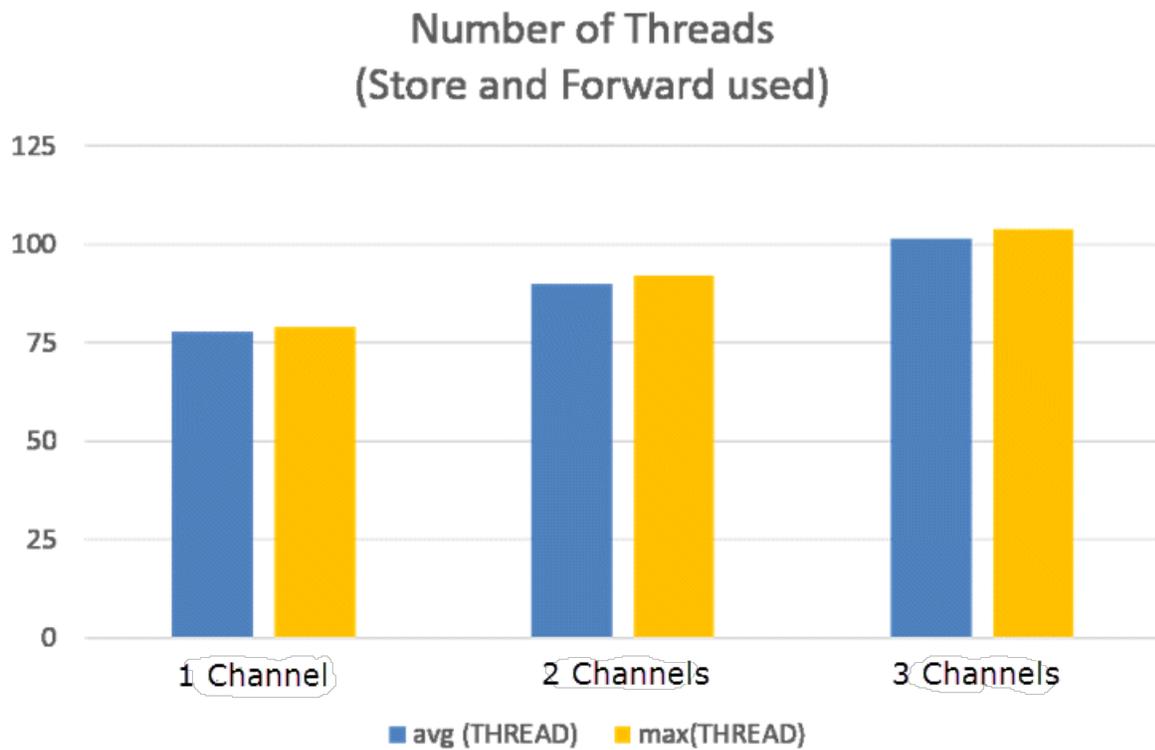
# of Channels	avg(MEM)	max(MEM)	perc90(MEM)
1	11.42	11.9	11.9
3	13.88	17.6	16.8
5	28.02	29.5	29.5

### Number of Threads

Metrics were collected from each test to determine the number of threads created by the Hoover spillway when using MQTT Adapter to send data to the Time Series service.

### Number of Threads with Store and Forward

The following figure shows the number of threads created when Predix Machine is configured with one, two, and three channels respectively, and the Store and Forward service is used.

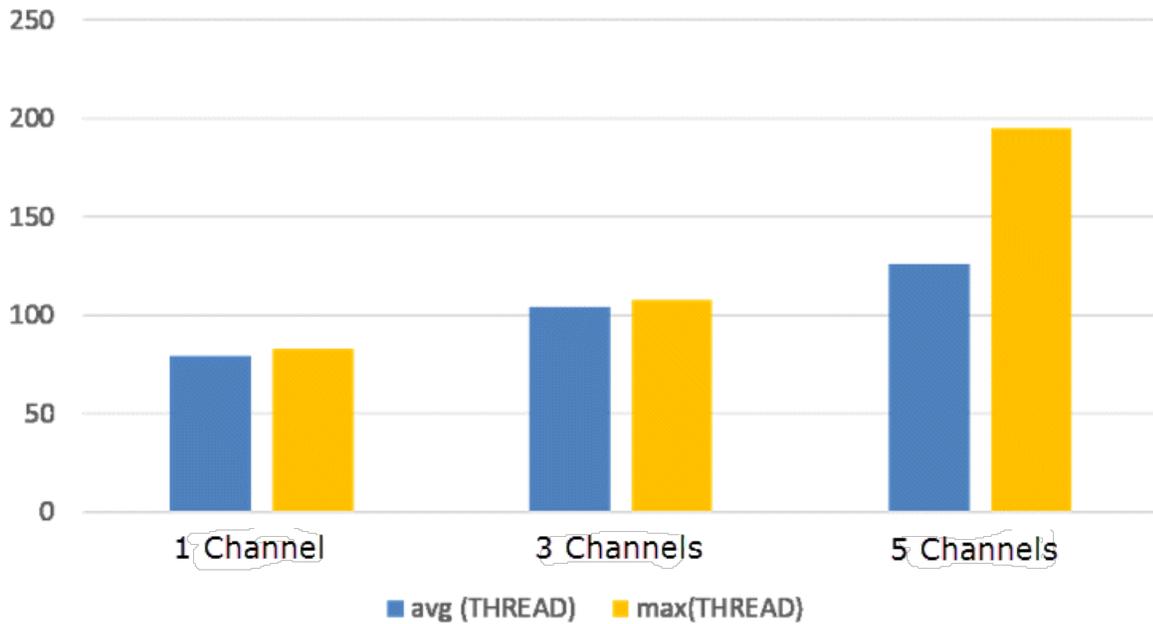


# of Channels	avg(THREADS)	max(THREADS)
1	77.95	79
2	89.95	92
3	101.45	104

### Number of Threads Without Store and Forward

The following figure shows the number of threads created when Predix Machine is configured with one, three, and five channels respectively, and not using the Store and Forward service.

### Number of Threads (Store and Forward not used)



# of Channels	avg(THREADS)	max(THREADS)
1	79.21	83
3	104.09	108
5	126.03	195

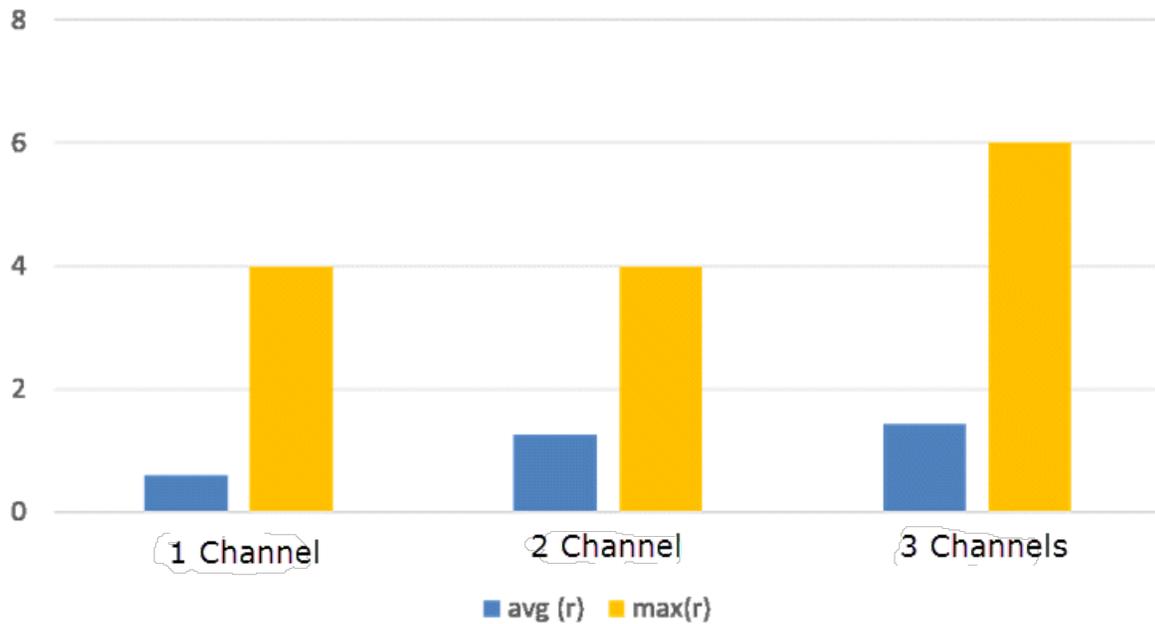
#### Average System Load

Metrics were collected from each test to determine the system load when using MQTT Adapter to send data to the Time Series service.

#### System Load with Store and Forward

The following figure shows the system load when Predix Machine is configured with one, two, and three channels, and is using the Store and Forward service.

### Average System Load (Store and Forward used)

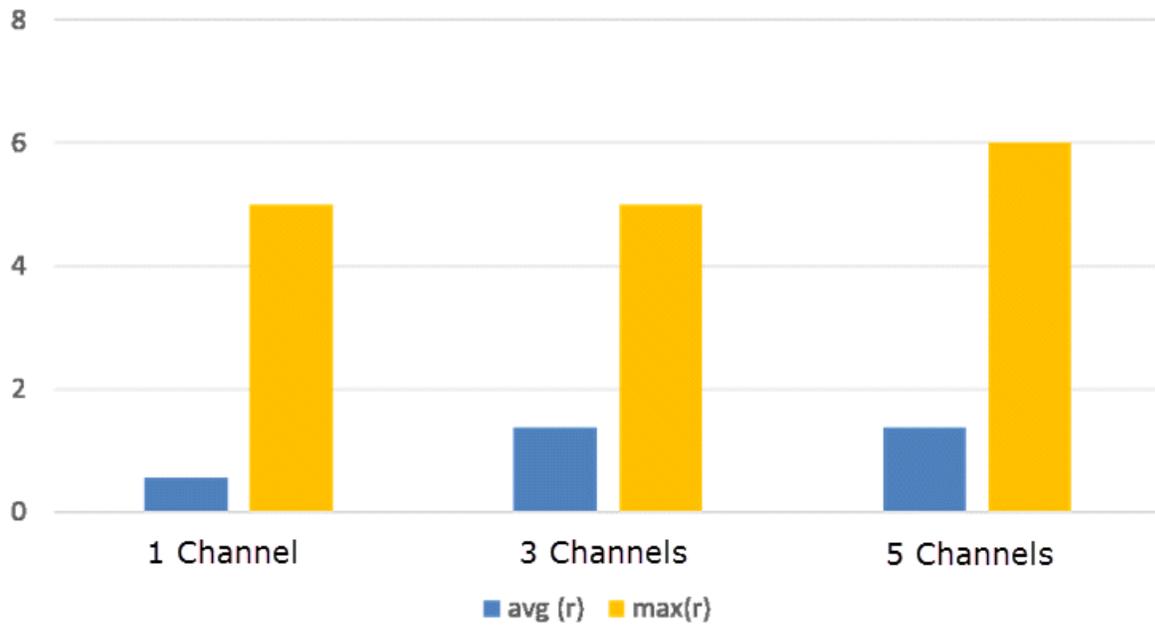


# of Channels	avg(r)	max(r)
1	0.59	4
2	1.25	4
3	1.43	6

### System Load Without Store and Forward

The following figure shows the system load when Predix Machine is configured with one, three, and five channels and is not using the Store and Forward service.

### Average System Load (Store and Forward not used)



Channels	avg(r)	max(r)
1	0.55	5
3	1.38	5
5	1.38	6

# Index

## A

- adapters  
142
- analytics  
10
- architecture  
7,10
- asset management  
10

## D

- DDS  
10

## J

- Java  
7

## M

- modbus  
142

## O

- OPC-UA  
10
- OSGi  
7

## P

- ping  
121
- Predix Ecosystem  
10
- Predix Gateway  
10