PowerMatcher Extension

Installation and Configuration

Authors:

A.H. Eisma (IBM)
C. Haverkamp (IBM)

Release: **0.9**

Revision: **2**

Date: 14-2-2013

TABLE OF CONTENTS

TABLE OF FIGURES

VERSION HISTORY

Version	Description	Author
0.7 - 1	Initial version for PowerMatcher release 0.7	C. Haverkamp (IBM)
0.9 - 1	Updated for PowerMatcher release 0.9	A.H. Eisma (IBM)
0.9 – 2	Updated for Bndtools 2.0	A.H. Eisma (IBM)

1. Introduction

PowerMatcher Extension provides components and component adapters that extend the PowerMatcher Core component set. This document describes these additional components, how to configure and run them, and what is required to setup the development environment.

2. PowerMatcher Extension overview

2.1 Components

The set of PowerMatcher Core components [1] is extended with the following PowerMatcher Extension components:

- **MicroBroker management agent**
  All communication between the components is done via a service bus using the MQTT-protocol. The MicroBroker management agent manages the creation of a MicroBroker MQTT bus. The agent supports different micro broker implementations.
- **Telemetry CSV Logging agent**
  This agent subscribes to status and telemetry log messages and stores them in a comma separated file (csv-file).

2.2 Adapters

PowerMatcher Extension provides the following additional adapters:

- **MQTT version 5 connection adapter**
  Messaging connections to the messaging bus of an MQTT broker are managed by MQTT connection adapter. PowerMatcher Core supports the open standard MQTTv3 protocol via the open source Eclipse Paho MQTTv3 client. PowerMatcher Extension also implements the MQTTv5 protocol, thereby supporting direct connections to IBM MicroBroker in the same runtime environment.
- **Telemetry adapters**
  The capability of sending or receiving telemetry data is added to an agent by this component, either via a direct connection or via a messaging connection. Device agents can report telemetry and other measurement data via these adapters. The CSV logging agent can receive telemetry data using this adapter.

2.3 Run-time environments

The run-time environments are the same as described in the Core documentation [1]. In addition, PowerMatcher Extension provides components and adapters that support the MQTT Java broker and client libraries that are shipped with IBM Lotus Expeditor (a licensed product) [2].

3. PowerMatcher Extension component configuration

The Power Matcher runtime configuration as described in the PowerMatcher Core Installation and Configuration documentation [1] is be used to explain how to extend a PowerMatcher Core-based system configuration with components from the Extension library.

Instead of direct connections, we will now use messaging connections to an MQTT MicroBroker running in the same Java runtime. Note that this is primarily for the purpose of illustrating the use of messaging connections. In practice, direct connections are preferred over messaging connections when running in the same Java runtime. Changes and additions from the PowerMatcher Core example are shown in *italics*.

Component	Main component	Adapter
Broker	*MicroBroker MQTT Broker Manager*	.
Auctioneer	PowerMatcher Auctioneer	Market Basis adapter
		MQTTv3 or v5 Matcher adapter
		MQTTv3 or v5 Logging adapter, to csv logging agent
Concentrator	PowerMatcher Concentrator	*MQTTv3 or v5 Protocol adapter*, to auctioneer and objective agent
		MQTTv3 or v5 Matcher adapter
		MQTTv3 or v5 Logging adapter, to csv logging agent
Objective Agent	PowerMatcher Objective Agent	*MQTTv3 or v5 Agent adapter*, to auctioneer
		MQTTv3 or v5 Logging adapter, to csv logging agent
CSV Logging Agent	PowerMatcher CSV Logging Agent	*MQTTv3 or v5 Log Listener adapter*
CSV Logging Agent	PowerMatcher CSV Logging Agent	*MQTTv3 or v5 Telemetry Listener adapter*
Example Agent 3	*PowerMatcher Extension Example Agent 3*	*MQTTv3 or v5 Agent adapter, to concentrator*
		MQTTv3 or v5 Logging adapter, to csv logging agent
		MQTTv3 or v5 Telemetry adapter, to telemetry logging agent
all		Time and scheduler adapter

Table 1 The components and component adapters for a PowerMatcher Extension application

In the configuration examples an MQTT version 5 connection adapter will be used. However, a version 3 connection adapter can be used as well.

In this chapter we will illustrate, just as for PowerMatcher Core, the creation of a basic PowerMatcher runtime using the following alternate approaches:

- Create a configuration manually using the OSGi web console.
- Configuration of an OSGi runtime using an XML configuration file.
- Configuration of a standard Java runtime using a properties file.

3.1 OSGi configuration example

3.1.1 Manual configuration using the web console

Using the Apache Felix Web Console the configuration can be created manually. Each configuration object for a component that is created with the web console is already prefilled with the default values. However, in this example we will not use the default direct connection adapters, so creating the configuration by hand is more involved.

Open the Apache Felix Web Console (http://localhost:8080/system/console). Then navigate to the Configuration tab and find the configuration name as listed in the table. Click the '+' to create a new configuration instance or edit a single configuration instance (e.g. for the Auctioneer) and fill in the properties as specified in the table. Select the Save button to create the configuration.

Component	Required OSGi configurations (name/class name)	Set property
Time Adapter	PowerMatcher Time Adapter Factory / net.powermatcher.core.scheduler.TimeAdapterFactory	*Accept all default values*

Scheduler Adapter	PowerMatcher Scheduler Adapter Factory / net.powermatcher.core.scheduler.SchedulerAdapterFactory	*Accept all default values*
Agent Adapter	PowerMatcher Agent Protocol Adapter / net.powermatcher.core.messaging.protocol.adapter. AgentProtocolAdapterFactory	Id: agentAdapterFactory Messaging adapter factory: mqttv5ConnectionFactory
Matcher Adapter	PowerMatcher Matcher Protocol Adapter / net.powermatcher.core.messaging.protocol.adapter. MatcherProtocolAdapterFactory	Id: matcherAdapterFactory Messaging adapter factory: mqttv5ConnectionFactory
Logging Adapter	PowerMatcher Logging Adapter / net.powermatcher.core.messaging.protocol.adapter. LoggingAdapterFactory	Id: loggingAdapterFactory Messaging adapter factory: mqttv5ConnectionFactory
Log Listener Adapter	PowerMatcher Log Listener Adapter / net.powermatcher.core.messaging.protocol.adapter. LogListenerAdapterFactory	Id: logListenerAdapterFactory Messaging adapter factory: mqttv5ConnectionFactory
Telemetry Adapter	Telemetry Adapter / net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryListenerAdapterFactory	Id: telemetryAdapterFactory Messaging adapter factory: mqttv5ConnectionFactory
Telemetry Listener Adapter	Telemetry Listener Adapter / net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryAdapterFactory	Id: telemetryListenerAdapterFactory Messaging adapter factory: mqttv5ConnectionFactory
Market Basis Adapter	PowerMatcher Market Basis Adapter / net.powermatcher.core.agent.marketbasis.adapter. MarketBasisAdapterFactory	*Accept all default values*
MQTTv5 Adapter	MQTTv5 Connection Adapter / net.powermatcher.expeditor.messaging.mqttv5. Mqttv5ConnectionFactory	*Accept all default values*
Auctioneer	PowerMatcher Auctioneer / net.powermatcher.core.agent.auctioneer.Auctioneer	Id: auctioneer Matcher factory id: matcherAdapterFactory Logging factory id: loggingAdapterFactory
Concentrator	PowerMatcher Concentrator / net.powermatcher.core.agent.concentrator.Concentrator	Id: concentrator Matcher id: auctioneer Agent factory id: agentAdapterFactory Matcher factory id: matcherAdapterFactory Logging factory id: loggingAdapterFactory
Objective Agent	PowerMatcher Objective Agent / net.powermatcher.core.agent.objective.ObjectiveAgent	Id: objectiveagent Matcher id: auctioneer Objective bid: (64,0);(64,-1000.0) Agent factory id: agentAdapterFactory Matcher factory id: matcherAdapterFactory Logging factory id: loggingAdapterFactory
CSV Logging Agent	PowerMatcher CSV logging Agent / net.powermatcher.core.agent.logging.CSVLoggingAgent	Id: csvlogging Log listener factory id: logListenerAdapterFactory
Telemetry CSV Logging Agent	Telemetry CSV logging Agent / net.powermatcher.agent.telemetry.logging.TelemetryCSVLoggingAgent	Id: telemetrylogging Telemetry listener factory id: telemetryListenerAdapterFactory
Test Agent	PowerMatcher Test Agent / net.powermatcher.core.agent.test. TestAgent	Id: testagent1 Matcher id: concentrator Agent factory id: agentAdapterFactory Logging factory id: loggingAdapterFactory
Example Agent 3	PowerMatcher Example Agent 3 / net.powermatcher.agent.template.ExampleAgent3	Id: exampleagent3 Matcher id: concentrator Agent factory id:

		agentAdapterFactory Logging factory id: loggingAdapterFactory Telemetry factory id: telemetryAdapterFactory

Table 2 Component properties to be defined in the web console for an Extension configuration.

3.1.2 Configuration using XML

This section describes how to extend the Core configuration with Extension components using XML configuration. The properties that must be defined for the example of Table 2 are shown in Table 3. In this table common adapter factory properties have been omitted; these are defined as global properties in the XML below.

Component	Required OSGi configurations (name/class name)	Set property
Time Adapter	PowerMatcher Time Adapter Factory / net.powermatcher.core.scheduler.TimeAdapterFactory	id: timeAdapterFactory
Scheduler Adapter	PowerMatcher Scheduler Adapter Factory / net.powermatcher.core.scheduler.SchedulerAdapterFactory	id: schedulerAdapterFactory
Agent Adapter	PowerMatcher Agent Protocol Adapter / net.powermatcher.core.messaging.protocol.adapter. AgentProtocolAdapterFactory	id: agentAdapterFactory
Matcher Adapter	PowerMatcher Matcher Protocol Adapter / net.powermatcher.core.messaging.protocol.adapter. MatcherProtocolAdapterFactory	id: matcherAdapterFactory
Logging Adapter	PowerMatcher Logging Adapter / net.powermatcher.core.messaging.protocol.adapter. LoggingAdapterFactory	id: loggingAdapterFactory
Log Listener Adapter	PowerMatcher Log Listener Adapter / net.powermatcher.core.messaging.protocol.adapter. LogListenerAdapterFactory	id: logListenerAdapterFactory
Telemetry Adapter	Telemetry Adapter / net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryListenerAdapterFactory	id: telemetryAdapterFactory
Telemetry Listener Adapter	Telemetry Listener Adapter / net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryAdapterFactory	id: telemetryListenerAdapterFactory
Market Basis Adapter	PowerMatcher Market Basis Adapter / net.powermatcher.core.agent.marketbasis.adapter. MarketBasisAdapterFactory	id: marketBasisAdapterFactory
MQTTv5 Adapter	MQTTv5 Connection Adapter / net.powermatcher.expeditor.messaging.mqttv5. Mqttv5ConnectionFactory	id: mqttv5ConnectionFactory
Auctioneer	PowerMatcher Auctioneer / net.powermatcher.core.agent.auctioneer.Auctioneer	id: auctioneer
Concentrator	PowerMatcher Concentrator / net.powermatcher.core.agent.concentrator.Concentrator	id: concentrator matcher.id : auctioneer
Objective Agent	PowerMatcher Objective Agent / net.powermatcher.core.agent.objective.ObjectiveAgent	id: objectiveagent matcher.id: auctioneer objective.bid: (64,0);(64,-1000.0)
CSV Logging Agent	PowerMatcher CSV logging Agent / net.powermatcher.core.agent.logging.CSVLoggingAgent	id: csvlogging log.listener.adapter.factory: logListenerAdapterFactory
Telemetry CSV Logging Agent	Telemetry CSV logging Agent / net.powermatcher.agent.telemetry.logging.TelemetryCSVLoggingAgent	id: telemetrylogging telemetry.listener.adapter.factory : telemetryListenerAdapterFactory
Test Agent	PowerMatcher Test Agent / net.powermatcher.core.agent.test. TestAgent	id: testagent1 matcher.id: concentrator
Example Agent 3	PowerMatcher Example Agent 3 / net.powermatcher.agent.template.ExampleAgent3	id: exampleagent3 matcher.id: concentrator

Table 3 Component properties to be defined in XML for a PowerMatcher runtime with Extension components.

In the following example all component and adapter factory configurations of Table 3 have been included in the XML. Note that factory ids must be specified explicitly and have been defined as global properties for the commonly used factories.

```
<?xml version="1.0" encoding="UTF-8"?>
<nodeconfig id="sample_node" name="Sample Extension node"
       description="Example PWM Extension configuration using XML." date="2012-06-01">
    <property name="time.adapter.factory" value="timeAdapterFactory" type="String"/>
    <property name="scheduler.adapter.factory" value="schedulerAdapterFactory" type="String"/>
    <property name="agent.adapter.factory" value="aqentAdapterFactory" type="String"/>
    <property name="matcher.adapter.factory" value="matcherAdapterFactory" type="String"/>
    <property name="logging.adapter.factory" value="loggingAdapterFactory" type="String"/>
    <property name="messaging.adapter.factory" value="mqttv5ConnectionFactory" type="String"/>
    <property name="log.listener.id" value="csvlogging" type="String"/>
    <configuration type="singleton" cluster="DemoPwmExt"
        pid="net.powermatcher.expeditor.broker.manager.BrokerManager" id="MicroBroker">
        <property name="broker.name" value="MicroBroker" type="String" />
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
                   pid="net.powermatcher.core.agent.auctioneer.Auctioneer" id="auctioneer">
        <property name="id" value="auctioneer" type="String"/>
        <property name="agent.adapter.factory" value="marketBasisAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
                   pid="net.powermatcher.core.agent.concentrator.Concentrator"
                   id="concentrator">
        <property name="id" value="concentrator" type="String"/>
        <property name="matcher.id" value="auctioneer" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
                   pid="net.powermatcher.core.agent.objective.ObjectiveAgent"
                   id="concentrator">
        <property name="id" value="objectiveagent" type="String"/>
        <property name="matcher.id" value="objectiveagent" type="String"/>
        <property name="objective.bid" value="(64,0);(64,-1000.0)" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
                   pid="net.powermatcher.core.agent.logging.CSVLoggingAgent"
                   id="csvlogging">
        <property name="id" value="csvlogging" type="String"/>
        <property name="log.listener.adapter.factory" value="logListenerAdapterFactory"
type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
                   pid="net.powermatcher.agent.telemetry.logging.TelemetryCSVLoggingAgent"
                   id="telemetrylogging">
        <property name="id" value="telemetrylogging" type="String"/>
        <property name="telemetry.listener.adapter.factory" value="telemetryListenerAdapterFactory"
type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
                   pid="net.powermatcher.core.agent.test.TestAgent"
                   id="testagent1">
        <property name="id" value="testagent1" type="String"/>
        <property name="matcher.id" value="concentrator" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
                   pid="net.powermatcher.agent.template.ExampleAgent3"
                   id="exampleagent1">
        <property name="id" value="exampleagent1" type="String"/>
        <property name="matcher.id" value="concentrator" type="String"/>
        <property name="telemetry.listener.id" value="telemetrylogging" type="String"/>
        <property name="telemetry.adapter.factory" value="telemetryAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.core.scheduler.TimeAdapterFactory"
                   id="timeAdapter">
        <property name="id" value="timeAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.core.scheduler.SchedulerAdapterFactory"
                   id="schedulerAdapter">
        <property name="id" value="schedulerAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.core.messaging.protocol.adapter.AgentProtocolAdapterFactory"
                   id="agentProtocolAdapter">
        <property name="id" value="agentProtocolAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.core.messaging.protocol.adapter.MatcherProtocolAdapterFactory"
                   id="matcherProtocolAdapter">
        <property name="id" value="matcherProtocolAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.core.messaging.protocol.adapter.LoggingProtocolAdapterFactory"
                   id="loggingProtocolAdapter">
        <property name="id" value="loggingProtocolAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.core.messaging.protocol.adapter.LogListenerProtocolAdapterFactory"
                   id="logListenerProtocolAdapter">
        <property name="id" value="logListenerProtocolAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryAdapterFactory"
                   id="telemetrydapter">
        <property name="id" value="telemetryAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryListenerAdapterFactory"
                   id="telemetryListenerAdapter">
        <property name="id" value="telemetryListenerAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.core.agent.marketbasis.adapter.MarketBasisAdapterFactory"
                   id="marketBasisAdapter">
        <property name="id" value="marketBasisAdapterFactory" type="String"/>
    </configuration>
    <configuration type="factory" cluster="DemoPwmExt"
         pid="net.powermatcher.expeditor.messaging.mqttv5.Mqttv5ConnectionFactory"
                   id=" mqttv5ConnectionFactory">
        <property name="id" value="mqttv5ConnectionFactory" type="String"/>
        <property name="broker.uri" value="tcp://localhost:1883" type="String" />
    </configuration>
</nodeconfig>
```

3.2 Stand-alone property file example

In order to create a stand-alone Java PowerMatcher environment similar to the environments created using the Apache Felix Web console and the XML configuration for the OSGi runtime, the same definitions as listed in Table 3 should be added to a properties file.

By default, PowerMatcher expects the properties file name ‘agent_config.properties’ in the same directory as the directory from where the application is started. It is possible to specify an alternative properties file path as described in section **Error! Reference source not found.**.The properties from Table 3 can be specified as follows in a properties file:

```
# Global properties.
cluster.id=DemoPwmExt
# By default logging will go to the configured CSVLoggingAgent agent.
log.listener.id=csvlogging
# By default telemetry will go to the configured TelemetryCSVLoggingAgent agent.
telemetry.listener.id=telemetrycsvlogging

# Define time & scheduler adapter factories
adapter.factory.timeAdapterFactory.class=net.powermatcher.core.scheduler.TimeAdapterFactory
adapter.factory.schedulerAdapterFactory.class=net.powermatcher.core.scheduler.SchedulerAdapterFactor
y

# Define adapter factories for message broker connections
adapter.factory.agentAdapterFactory.class=net.powermatcher.core.messaging.protocol.adapter.AgentProt
ocolAdapterFactory
adapter.factory.agentAdapterFactory.agent.messaging.protocol=INTERNAL_v1
adapter.factory.matcherAdapterFactory.class=net.powermatcher.core.messaging.protocol.adapter.Matcher
ProtocolAdapterFactory
adapter.factory.matcherAdapterFactory.matcher.messaging.protocol=INTERNAL_v1
adapter.factory.loggingAdapterFactory.class=net.powermatcher.core.messaging.protocol.adapter.Logging
AdapterFactory
adapter.factory.logListenerAdapterFactory.class=net.powermatcher.core.messaging.protocol.adapter.Log
ListenerAdapterFactory
adapter.factory.telemetryAdapterFactory.class=net.powermatcher.telemetry.messaging.protocol.adapter.
TelemetryAdapterFactory
```

```
adapter.factory.telemetryListenerAdapterFactory.class=net.powermatcher.telemetry.messaging.protocol.
adapter.TelemetryListenerAdapterFactory
adapter.factory.mqttv5ConnectionFactory.class=net.powermatcher.expeditor.messaging.mqttv5.Mqttv5Conn
ectionFactory


# Default scheduler adapter factories
time.adapter.factory=timeAdapterFactory
scheduler.adapter.factory=schedulerAdapterFactory

# Default adapter factories for message broker connections
agent.adapter.factory=agentAdapterFactory
matcher.adapter.factory=matcherAdapterFactory
logging.adapter.factory=loggingAdapterFactory
log.listener.adapter.factory=logListenerAdapterFactory
telemetry.adapter.factory=telemetryAdapterFactory
telemetry.listener.adapter.factory=telemetryListenerAdapterFactory
messaging.adapter.factory=mqttv5ConnectionFactory


# Auctioneer properties
agent.auctioneer.class=net.powermatcher.core.agent.auctioneer.Auctioneer
agent.auctioneer.id=auctioneer
agent.auctioneer1.agent.adapter.factory=marketBasisAdapterFactory

# Concentrator properties
agent.concentrator.class=net.powermatcher.core.agent.concentrator.Concentrator
agent.concentrator.id=concentrator
agent.concentrator.matcher.id=auctioneer

# ObjectiveAgent properties
agent.objectiveagent.class=net.powermatcher.core.agent.objective.ObjectiveAgent
agent.objectiveagent.id=objectiveagent
agent.objectiveagent.matcher.id=auctioneer
agent.objectiveagent.objective.bid=(64,0);(64,-1000.0)

# CSVLoggingAgent properties
agent.csvlogging.class=net.powermatcher.core.agent.logging.CSVLoggingAgent
agent.csvlogging.id=csvlogging

# TelemetryCSVLoggingAgent properties
agent.telemetrycsvlogging.class=net.powermatcher.agent.telemetry.logging.TelemetryCSVLoggingAgent
agent.telemetrycsvlogging.id=telemetrycsvlogging
agent.telemetrycsvlogging.enabled=true

# Test Agent 1
agent.testagent1.class=net.powermatcher.core.agent.test.TestAgent
agent.testagent1.id=testagent1
agent.testagent1.matcher.id=concentrator

# Example Agent 3
agent.testagent1.class=net.powermatcher.agent.template.ExampleAgent3
agent.testagent1.id=exampleagent3
agent.testagent1.matcher.id=concentrator
```

4. Additional Development environment setup for Extension

Set setup of an Eclipse IDE for PowerMacher Core has been described in the Core Installation and Configuration document [1]. This chapter describes the additional setup that is required for development with PowerMatcher Extension components. The pattern is the

4.1 *PowerMatcher maintenance versus application development*

There are two different configurations for the PowerMatcher Core and Extension development environment.

1. **The development environment for the Core and Extension agent and application developer**. This environment contains the Core and Extension Example template source code projects and uses the binary distribution of the PowerMatcher Core and Extension libraries and bundles.
2. **The development environment for the PowerMatcher Core and Extension developer**, for maintaining and extending the PowerMatcher Core and Extension code base.
   This environment uses all Core, Core Example, Extension and Extension Example source code projects as well as the binary distribution of the PowerMatcher Core and Extension libraries and bundles.

4.2 *Lotus Expeditor Client installation*

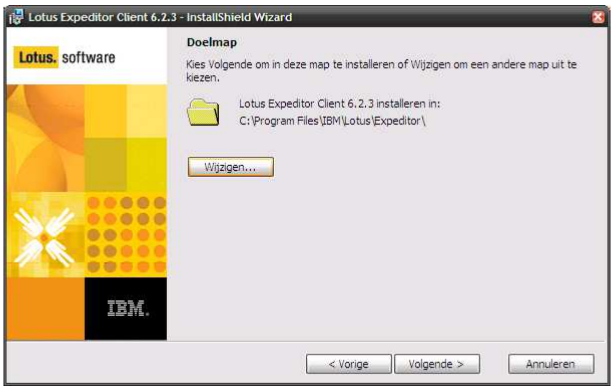
A subset of the PowerMatcher Extension source code, the `net.powermatcher.expeditor` projects, requires that you have the Java MicroBroker libraries installed on your system. MicroBroker is shipped with IBM Lotus Expeditor Client 6.2.3, a commercial product that requires a license [2]. Install the Lotus Expeditor Client on the Windows or Linux development workstation and accept the default installation paths. The default installation path is referenced in the Eclipse User Library definition that you will import in a later step.

After installation the following MicroBroker libraries are available as part of the IBM Lotus Expeditor Client installation:

- com.ibm.micro-3.0.2.jar (com.ibm.micro_3.0.3.0-20110708.jar)
- com.ibm.micro.admin-3.0.3 (com.ibm.micro.admin_3.0.3.0-20110708.jar)
- com.ibm.micro.client-3.0.2 (com.ibm.micro.client_3.0.3.0-20110708.jar)
- com.ibm.mqttclient-3.0.2 (com.ibm.mqttclient_3.0.3.0-20110708.jar)
- com.ibm.micro.utils-3.0.2 (com.ibm.micro.utils_3.0.3.0-20110708.jar)
- com.ibm.micro.utils.extended-3.0.2 (com.ibm.micro.utils.extended_3.0.3.0-20110708.jar).

4.2.1 Installation on Windows

For installation of Lotus Expeditor Client version 6.2.3 on Windows start the setupwin32.exe in the ‘desktop’ folder of the Client installation CD. Accept the default installation path and click next.



Lotus Expeditor Client 6.2.3 - InstallShield Wizard

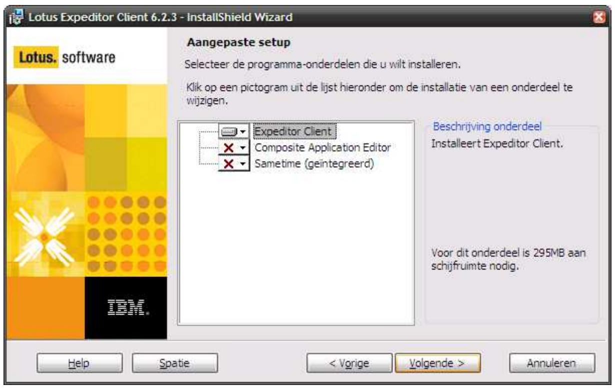
Doelmap

Kies Volgende om in deze map te installeren of Wijzigen om een andere map uit te kiezen.

Lotus Expeditor Client 6.2.3 installeren in:
C:\Program Files\IBM\Lotus\Expeditor\

Figure 1 Lotus Expeditor Client installation wizard start screen.

Accept in the next screen the defaults (only install Expeditor Client) and click next.


Lotus Expeditor Client 6.2.3 - InstallShield Wizard

Aangepaste setup

Selecteer de programma-onderdelen die u wilt installeren.

Klik op een pictogram uit de lijst hieronder om de installatie van een onderdeel te wijzigen.

- Expeditor Client
- ✗ Composite Application Editor
- ✗ Sametime (geïntegreerd)

Beschrijving onderdeel

Installeert Expeditor Client.

Voor dit onderdeel is 295MB aan schijfruimte nodig.

Figure 2 Lotus Expeditor Client installation wizard screen for changing setup defaults.

Finally click Install to start the installation.

The library files for MicroBroker can be found after installation of Lotus Expeditor Client in the following directory:

```
C:\Program Files (x86)\IBM\Lotus\Expeditor\rcp\eclipse\plugins

       or, for win32:

C:\Program Files\IBM\Lotus\Expeditor\rcp\eclipse\plugins
```

The Lotus Expeditor Client installation also includes an IBM JRE which can be found in the following directory:

```
C:\Program Files (x86)\IBM\Lotus\Expeditor\rcp\eclipse\\com.ibm.rcp.j2se.win32.x86_1.6.0.20110714a-
201108151128

       or, for win32:

C:\Program Files\IBM\Lotus\Expeditor\rcp\eclipse\plugins\com.ibm.rcp.j2se.win32.x86_1.6.0.20110714a-
201108151128
```

4.2.2 Installation on Linux

For installation of Lotus Expeditor Client version 6.2.3 on Linux, install the expeditor-6.2.3-201109020733.i586.rpm or expeditor-6.2.3-201109020733_i386.deb package in the ‘desktop’ folder of the Client installation CD.

The library files for MicroBroker can be found after installation of Lotus Expeditor Client in the following directory:

```
/opt/ibm/lotus/Expeditor/rcp/eclipse/plugins
```

The Lotus Expeditor Client installation also includes an IBM JRE which can be found in the following directory:

```
/opt/ibm/lotus/Expeditor/rcp/eclipse/plugins/com.ibm.rcp.j2se.win32.x86_1.6.0.20110714a-201108151128
```

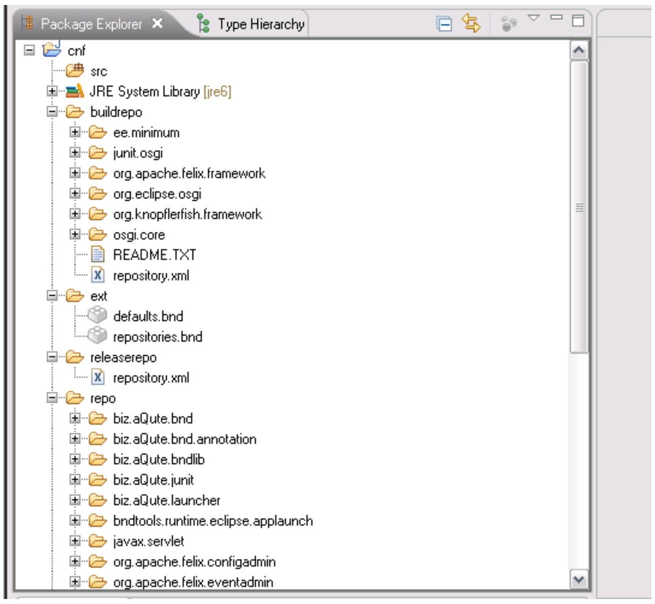
4.3 Retrieving the source code

The source code of PowerMatcher Core is delivered as an Eclipse project archive compressed in ZIP format. Download the PowerMatcher Extension zip file from the PowerMatcher download site.

Unzip the file to a directory and remember the location for the next steps.

4.4 IDE Setup

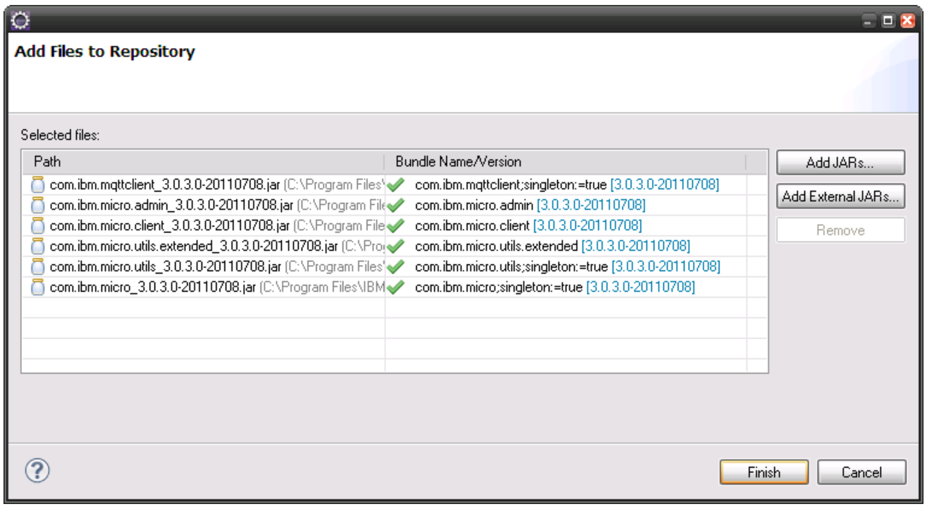
Start Eclipse and open the workspace that contains a workspace with the PowerMatcher Core sources. This workspace should also contain the 'cnf' project that was created by Bndtools. This project contains the bundle repository.



```
cnf
  src
  JRE System Library [jre6]
  buildrepo
    ee.minimum
    junit.osgi
    org.apache.felix.framework
    org.eclipse.osgi
    org.knopflerfish.framework
    osgi.core
    README.TXT
    repository.xml
  ext
    defaults.bnd
    repositories.bnd
  releaserepo
    repository.xml
  repo
    biz.aQute.bnd
    biz.aQute.bnd.annotation
    biz.aQute.bndlib
    biz.aQute.junit
    biz.aQute.launcher
    bndtools.runtime.eclipse.applaunch
    javax.servlet
    org.apache.felix.configadmin
    org.apache.felix.eventadmin
```

Figure 3 The 'cnf' project created by Bndtools prefilled with common bundles.

The MicroBroker bundles need to be added to the Bndtools Local repository. In the Bndtools perspective, navigate to the Repositories view and select Local repository. Click the icon of the 'Add bundles to repository' and add the jar files listed in paragraph 4.1, or drag and drop the files on the Local repository entry:



Add Files to Repository

Selected files:

Path	Bundle Name/Version
com.ibm.mqttclient_3.0.3.0-20110708.jar (C:\Program Files\	✓ com.ibm.mqttclient;singleton:=true [3.0.3.0-20110708]
com.ibm.micro.admin_3.0.3.0-20110708.jar (C:\Program File	✓ com.ibm.micro.admin [3.0.3.0-20110708]
com.ibm.micro.client_3.0.3.0-20110708.jar (C:\Program File	✓ com.ibm.micro.client [3.0.3.0-20110708]
com.ibm.micro.utils.extended_3.0.3.0-20110708.jar (C:\Pro	✓ com.ibm.micro.utils.extended [3.0.3.0-20110708]
com.ibm.micro.utils_3.0.3.0-20110708.jar (C:\Program Files\	✓ com.ibm.micro.utils;singleton:=true [3.0.3.0-20110708]
com.ibm.micro_3.0.3.0-20110708.jar (C:\Program Files\IBM	✓ com.ibm.micro;singleton:=true [3.0.3.0-20110708]

Figure 4 Add the MicroBroker jar files to the Bndtools repository.

4.5 Importing the PowerMatcher Extension binary distribution

This step is required for both the agent developer workspace, as well as for the workspace for PowerMatcher maintenance. In this step a new Bndtools local repository is added to the workspace that will contain the PowerMatcher Extension bundle library jars.

1. Create workspace folders `/cnf/extrepo` and `/cnf/agentrepo`.
2. Edit /cnf/ext/repositories.bnd and insert the following two lines to add two new repositories labeled 'Extension' and 'Agent':

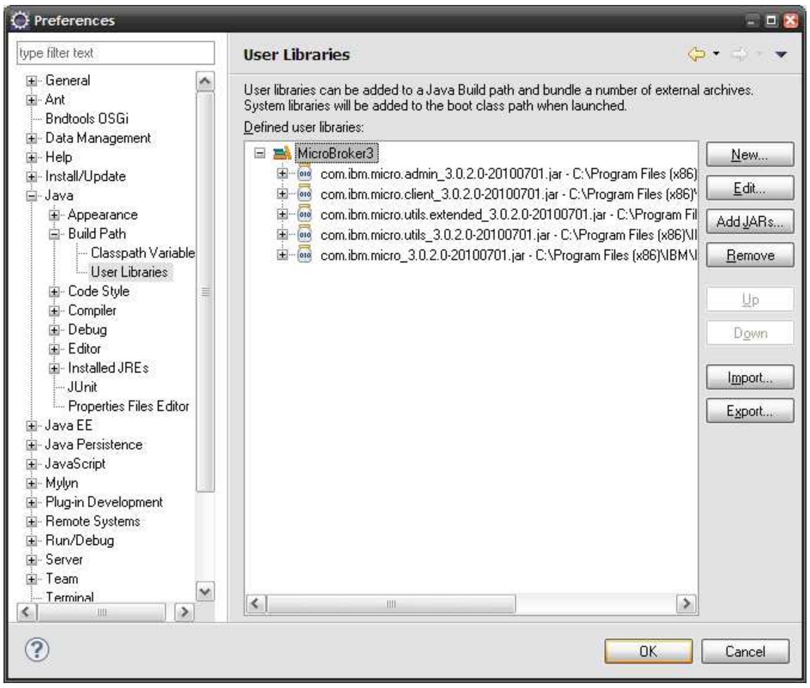
```
        aQute.bnd.deployer.repository.LocalIndexedRepo; name=Extension;
local=${workspace}/cnf/extrepo;pretty=true,\
       aQute.bnd.deployer.repository.LocalIndexedRepo; name=Agent;
local=${workspace}/cnf/agentrepo;pretty=true,\
```

3. Add all PowerMatcher Extension and Agent bundle jars to the Extension repository by dragging and dropping the jars on 'Extension' and 'Agent' in the Repositories View.

4.6 Importing the PowerMatcher Extension and Extension Example sources

The source projects can now be imported into the workspace. First switch off the automatic build option by de-selecting 'Build automatically…' in the Project menu. Then use the Import… wizard and select 'Import existing projects into workspace' and point the wizard to the folder where you exported the project archive file. Select all projects and check 'Copy projects into workspace' and import the projects by clicking the Finish button.

Before the sources can be successfully compiled a 'user library' for MicroBroker needs to be created. The user library settings are stored in a file and can be imported via the Eclipse Preferences window:



User libraries can be added to a Java Build path and bundle a number of external archives.
System libraries will be added to the boot class path when launched.

Defined user libraries:

MicroBroker3
- com.ibm.micro.admin_3.0.2.0-20100701.jar - C:\Program Files (x86)
- com.ibm.micro.client_3.0.2.0-20100701.jar - C:\Program Files (x86)
- com.ibm.micro.utils.extended_3.0.2.0-20100701.jar - C:\Program Fil
- com.ibm.micro.utils_3.0.2.0-20100701.jar - C:\Program Files (x86)\I
- com.ibm.micro_3.0.2.0-20100701.jar - C:\Program Files (x86)\IBM\

The user library definition file for the MicroBroker libraries is located in the net.powermatcher.expeditor.launcher.mbroker.template project and is named 'PowerMatcher MicroBroker Extension.userlibraries'. The import will create the ***MicroBroker3*** **user library** that references the MicroBroker library files in the Bndtools cnf directory. Click on the 'Import…' button in the User Libraries section in the Preferences window.

There is another user library pre-defined that is required for projects that use Telemetry features provided by PowerMatcher Extension. Import the file 'PowerMatcher Extension.userlibraries' that is located in project net.powermatcher.extension.launcher.main.template. This will create the ***PowerMatcher Extension*** **user library**.

After the import completed restart the workbench (File -> Restart). The cnf repository cache of BND tools will now be cleared. From the message bar at the bottom of the window you can see when this task is running. If the task is completed check the 'Build automatically…' option or hit Ctrl-B (i.e. Build All). All projects should now compile successfully.

4.7 Launching the example Extension applications from the workbench

This paragraph describes how to run the example PowerMatcher applications that contain Extension components, from the development workbench in Eclipse. There will be two options: running a stand-alone application and running it within OSGi.

4.7.1 Run the Extension example application as a stand-alone application

The PowerMatcher Extension stand-alone application is an example application that contains PowerMatcher Extension components. This example depends on an external broker. Because a broker is required, an external broker needs to be started first, using one of the following two options:

1. Start the open source Mosquitto broker as explained in the PowerMatcher Core Installation and Configuration manual.
2. Use the ‘Standalone MicroBroker Launch’ Eclipse launch configuration that is contributed by the `net.powermatcher.expeditor.launcher.mbroker` project. This launch configuration starts MicroBroker as a standard Java application, using the configuration properties defined in file broker_config.properties.

When the broker is running, the example application can be started with the launch configuration named ‘PowerMatcher Extension Launch Template’ from project `net.powermatcher.extension.launcher.main.template`.

4.7.2 Run the MicroBroker Extension example application as a stand-alone application

The MicroBroker Extension example application located in the project `net.powermatcher.expeditor.launcher.mbroker.template` contains an example of a launch configuration that starts both a PowerMatcher application and MicroBroker in the same Java runtime. This launch configuration also uses the configuration properties defined in file broker_config.properties to configure the broker.

The example application can be started with the launch configuration named ‘PowerMatcher Launch with MicroBroker Template.launch’ from project `net.powermatcher.expeditor.launcher.mbroker.template`.

4.7.3 Run the Extension example application as an OSGi application

The PowerMatcher Extension OSGi example application contains PowerMatcher Extension components and runs in an OSGi run-time environment. This example depends on an external broker. Because a broker is required, an external broker needs to be started first, using one of the following two options:

3. Start the open source Mosquitto broker as explained in the PowerMatcher Core Installation and Configuration manual.
4. Use the ‘Standalone MicroBroker Launch’ Eclipse launch configuration that is contributed by the `net.powermatcher.expeditor.launcher.mbroker` project. This launch configuration starts MicroBroker as a standard Java application, using the configuration properties defined in file broker_config.properties.

When the broker is running, the example application can be started with the Bndtools run configuration named ‘PowerMatcher OSGi Launch Example Template.bndrun’ from project `net.powermatcher.extension.launcher.osgi.template`.

4.7.4 Run the MicroBroker Extension example application as an OSGi application

The PowerMatcher MicroBroker Extension OSGi example application contains PowerMatcher Extension components, including the MicroBroker Management Agent, and runs in an OSGi run-time environment. The MicroBroker Management Agent starts a MicroBroker instance as defined in `example_extension_config.xml`.

The example application can be started with the Bndtools run configuration named ‘PowerMatcher OSGi Launch MBroker Example Template.bndrun’ from project `net.powermatcher.expeditor.launcher.osgi.template`.

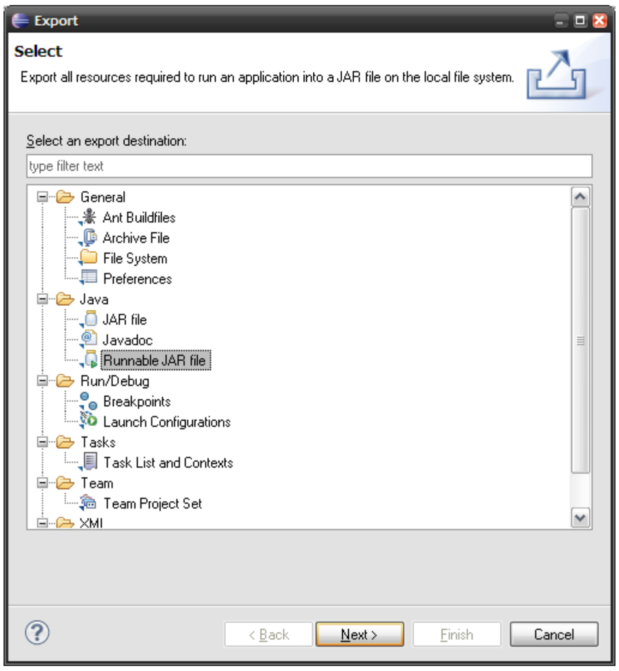
This is the preferred configuration for running a PowerMatcher component in a distributed environment because it is possible to define bridging of topics between distributed brokers from the configuration XML.

4.8 Building a PowerMatcher Extension application

4.8.1 Build the stand-alone Extension example application

Make sure the source code is compiled by verifying if the option ‘Build Automatically’ is checked in the Project menu. If not check the option and verify the compilation progress in the status bar at the bottom of the window.

From the File menu select Export… menu option. The Export dialog will appear.
Select the item Java>Runnable JAR and hit next:


Export

Select

Export all resources required to run an application into a JAR file on the local file system.

Select an export destination:

type filter text

- General
  - Ant Buildfiles
  - Archive File
  - File System
  - Preferences
- Java
  - JAR file
  - Javadoc
  - Runnable JAR file
- Run/Debug
  - Breakpoints
  - Launch Configurations
- Tasks
  - Task List and Contexts
- Team
  - Team Project Set
- XML

Figure 5 Selecting the ‘Runnable JAR file’ option from the Eclipse Export wizard.

The page ‘Runnable JAR file export’ will appear. Select the launch configuration ‘PowerMatcher Extension Launch Template’ and type the export destination (for example c:\export\exampleapp.jar). The option ‘Copy required libraries into a sub-folder next to the generated JAR’ will create an `exampleapp_lib` directory containing the required libraries.

The application needs also the `logback.xml` and the `agent_config.properties`. Copy these files from the `net.powermatcher.extension.launcher.main.template` project to the build directory.

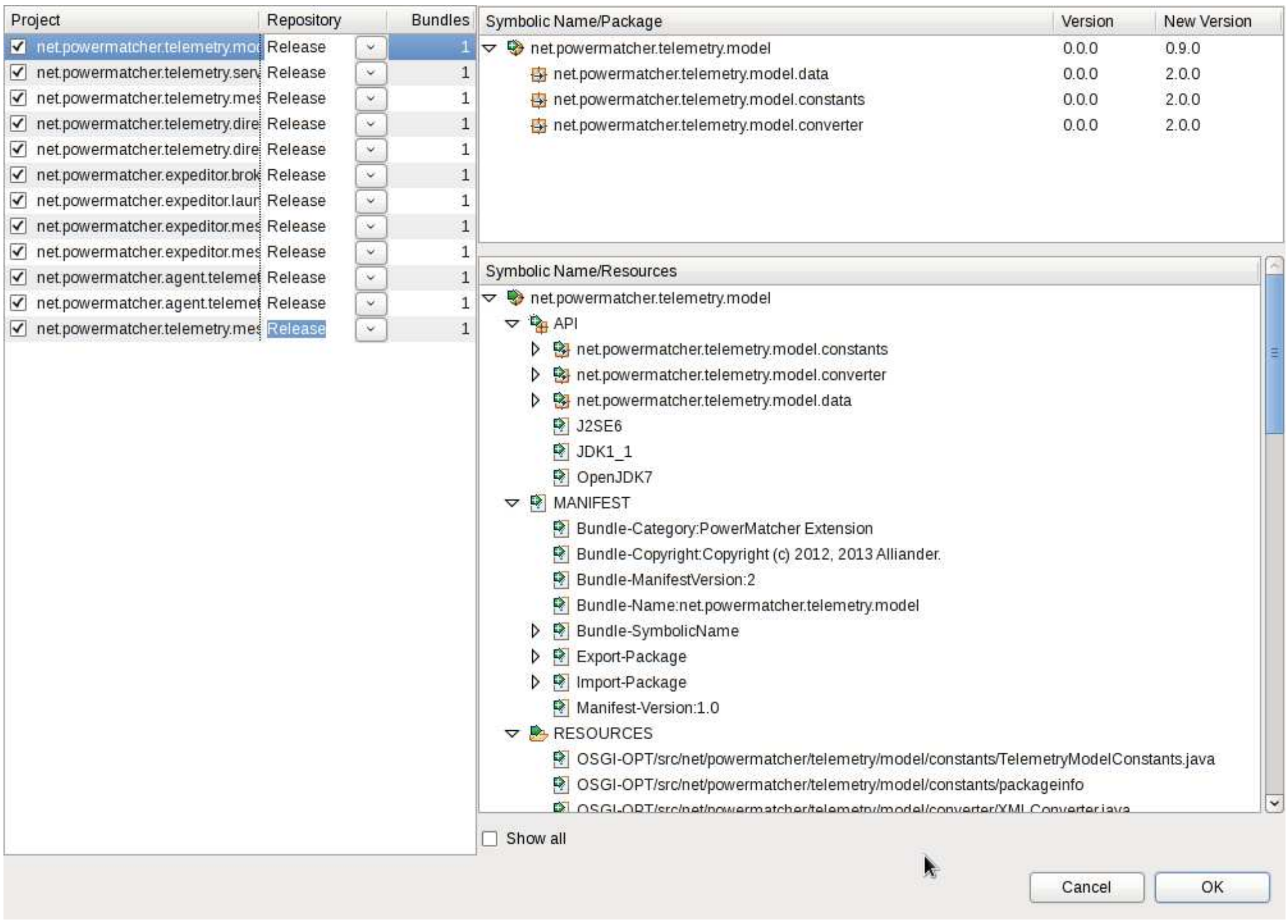
4.8.2 Build the PowerMatcher Extension OSGi bundle jars

If you run a PowerMatcher OSGi application from the workspace, the launch configurations have been set up to use the bundle jars located in the ‘generated’ directory of the projects. Besides building and releasing individual bundles using the Bndtools Release Bundle wizard on the bnd.bnd file in a bundle project, it is also possible to build all bundles using an Ant script for PowerMatcher Extension.

Note that when running a PowerMatcher application as a standard Java application, the library jars from the Bndtools Core and Extension repositories are referenced though the Eclipse User Library definitions. In this case the jars located in the ‘generated’ directory of the projects will not be used.

Bndtools provides two ways to build OSGi bundles for deployment and release. Since Bndtools 2.0 the built-in Release Bundles wizard can be used to build and release a single or multiple bundles in one step. Alternatively, the Bndtools Ant build scripts can be used as explained below. However, this is much more involved.

To build the Extension or Agent bundles using the Release Bundles wizard, select one or more projects, or select the working set containing all projects, and select Release Bundles from the context menu. By default, the wizard releases all bundles to the Release repository, as shown in the following figure.



Project	Repository	Bundles
☑ net.powermatcher.telemetry.mo	Release	1
☑ net.powermatcher.telemetry.serv	Release	1
☑ net.powermatcher.telemetry.mes	Release	1
☑ net.powermatcher.telemetry.dire	Release	1
☑ net.powermatcher.telemetry.dire	Release	1
☑ net.powermatcher.expeditor.brok	Release	1
☑ net.powermatcher.expeditor.laun	Release	1
☑ net.powermatcher.expeditor.mes	Release	1
☑ net.powermatcher.expeditor.mes	Release	1
☑ net.powermatcher.agent.telemet	Release	1
☑ net.powermatcher.agent.telemet	Release	1
☑ net.powermatcher.telemetry.mes	Release	1

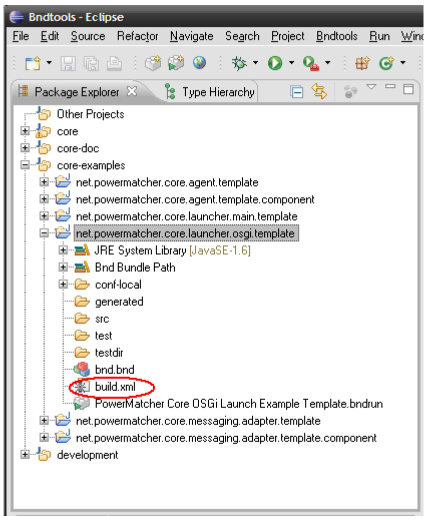
Symbolic Name/Package	Version	New Version
net.powermatcher.telemetry.model	0.0.0	0.9.0
net.powermatcher.telemetry.model.data	0.0.0	2.0.0
net.powermatcher.telemetry.model.constants	0.0.0	2.0.0
net.powermatcher.telemetry.model.converter	0.0.0	2.0.0

```
Symbolic Name/Resources
net.powermatcher.telemetry.model
  API
    net.powermatcher.telemetry.model.constants
    net.powermatcher.telemetry.model.converter
    net.powermatcher.telemetry.model.data
    J2SE6
    JDK1_1
    OpenJDK7
  MANIFEST
    Bundle-Category:PowerMatcher Extension
    Bundle-Copyright:Copyright (c) 2012, 2013 Alliander.
    Bundle-ManifestVersion:2
    Bundle-Name:net.powermatcher.telemetry.model
    Bundle-SymbolicName
    Export-Package
    Import-Package
    Manifest-Version:1.0
  RESOURCES
    OSGI-OPT/src/net/powermatcher/telemetry/model/constants/TelemetryModelConstants.java
    OSGI-OPT/src/net/powermatcher/telemetry/model/constants/packageinfo
```

Figure 6 Build Core bundles using the Bndtools Release Bundles wizard.

To build and release the bundles to a different repository, change the `–releaserepo` setting in /cnf/ext/repositories.bnd to Extension or Agent.

The build and release using Ant requires a workspace that does not contain the PowerMatcher Core and Core Example projects, but only the PowerMatcher Core and Core Example binary distribution. The reason is that the Bndtools Ant build script builds dependent projects recursively, which would result in a release of the rebuilt Core bundles into the Extension repository.



Bndtools - Eclipse
File Edit Source Refactor Navigate Search Project Bndtools Run Wind
Package Explorer
Type Hierarchy

```
Other Projects
core
core-doc
core-examples
  net.powermatcher.core.agent.template
  net.powermatcher.core.agent.template.component
  net.powermatcher.core.launcher.main.template
  net.powermatcher.core.launcher.osgi.template
    JRE System Library [JavaSE-1.6]
    Bnd Bundle Path
    conf-local
    generated
    src
    test
    testdir
    bnd.bnd
    build.xml
    PowerMatcher Core OSGi Launch Example Template.bndrun
  net.powermatcher.core.messaging.adapter.template
  net.powermatcher.core.messaging.adapter.template.component
development
```

Figure 7 Build all Extension bundles with the build.xml script in the OSGi example project.

The build and release using Ant is started by selecting the build.xml file en choosing “Run As > ANT Build…”. The default Ant target is ‘build’. To update the bundles in the Bndtools Extension or Agent repository in the workspace, the ‘release’ target must be selected and the `–releaserepo` setting in /cnf/ext/repositories.bnd must be set to Extension.

All PowerMatcher Extension, Extension Example and Agent bundles are built through the Ant build.xml file in the `net.powermatcher.extension.resource` project. Individual bundles can be built and released via Run As > Ant Build on the build.xml in a bundle project.

Note that the build and release may take a considerable amount of time. The reason is that for each bundle references, Bndtools rebuilds and releases all prerequisite bundles recursively.

After the build completes, the released Agent bundles can be moved from /cnf/extrepo to /cnf/agentrepo. The Eclipse workspace must be restarted after moving bundles in cnf for Bndtools to update its indices.

5. Appendix A: PowerMatcher Extension Components

5.1 Extension components

This section describes the extension components, their configuration properties and the dependencies.

5.1.1 Broker Manager

The Broker Manager component creates and starts a broker. This is a singleton component, which means that at most only once broker can be configured in the runtime.

Component: Broker Manager				**Type: singleton component**
Class: net.powermatcher.expeditor.broker.manager.BrokerManager				
Dependencies: **Microbroker libararies**				
Property	**Type**	**Req.**	**Default**	**Description**
broker.name	String	N	MicroBroker	MicroBroker instance name. A directory with the same name will be created to store MicroBroker data.
microbroker.dir	String	N	DefaultCluster	Directory where the MicroBroker data directory will be created.
microbroker.maxMessageSize	int	N	100	Maximum message size that is allowed on the created MQTT bus.
microbroker.maxNumberOfClients	int	N	-1	Maximum number of clients (-1 is no limit).
microbroker.numberOfLogFilesToKeep	int	N	10	Maximum number of log files to keep.
microbroker.persistence	int	N	0	Persist the messages on topics and queues.
microbroker.port	int	N	1883	MicroBroker port.
microbroker.queueSize	int	N	0	Message queue size.
microbroker.waitTimeout	long	N	10000	Timeout.

Security is enabled by default if JAAS is supported in the Java runtime environment. Anonymous connections are permitted by default. Local authentication requires customization of the default 'acl'-file that is generated when a new broker is started and requires a custom JAAS authentication module. This is out of scope for this document.

5.1.2 Micro broker bridge

The MicroBroker bridge or Pipe is used to connect two MQTT brokers by defining the topic patterns that are bridged from (outbound) and to (inbound) this broker.

Component: Broker Manager				**Type: factory component**
Class: net.powermatcher.expeditor.broker.manager.Pipe				
Dependencies: **Microbroker libararies**				
Property	**Type**	**Req.**	**Default**	**Description**
id	String	Y	-	Component instance id e.g. 'auctioneer1'
cluster.id	String	N	DefaultCluster	Cluster id or sometimes also referred to as namespace id.
pipe.name	String	N	-	Maximum message size that is allowed on the created MQTT bus.
in.topics	String	N	PowerMatcher/${cluster.id}/+/UpdatePriceInfo,Configuration/Update/+	Defines allowed incoming (from the remote host) messages.
in.target.topic	String	N	10	If specified incoming topics will be put on this topic on the (local) host.
out.topics	String	N	PowerMatcher/${cluster.id}/+/+/UpdateBid\\,PowerMatcher/${cluster.id}/+/+/Log\\,CPSS/#\\,Status/#	Defines allowed outgoing (to the remote host) messages.
out.target.topic	String	N	1883	If specified outgoing messages will be put on this topic on the remote host.
keep.alive.secs	short	N	60	Defines interval for the 'keep alive' signal.
remote.host	long	Y	-	Hostname of target MQTT bus.
remote.port	int	N	1883	Port number of target MQTT bus.
remote.secure	boolean	N	false	Enforce secure connection.
remote.username	String	N	-	Remote user for authentication secure connection.
remote.password	String	N	-	Remote password for authentication

				secure connection.
notification.enabled	boolean	N	true	Send notification messages.
host.name	String	N	-	Host name of Bridge start MQTT bus.
component.name	String	N	-	Component name.
notification.topic	String	N	Status/Bridge	Topic name of notification message.
notification.connected.message	String	N	CONNECTED\\,I\\,${cluster.id}\\,${id}\\,${pipe.name}\\,${component.name}\\,${host.name}	Format of notification message.
notification.cleandisconnected.message	String	N	CLEANDISCONNECTED\\,I\\,${cluster.id}\\,${id}\\,${pipe.name}\\,${component.name}\\,${host.name}	Format of clean disconnect message.
notification.disconnected.message	String	N	DISCONNECTED\\,W\\,${cluster.id}\\,${id}\\,${pipe.name}\\,${component.name}\\,${host.name}	Format of disconnect message caused by an exception.

5.1.3 Telemetry CSV Logging agent

The Telemetry CSV Logging Agent listens to status, telemetry, price update and bid log messages and stores them in a comma separated file (csv-file).

Component: Telemetry CSV Logging Agent			**Type: factory component**	
Class: net.powermatcher.agent.telemetry.logging.TelemetryCSVLoggingAgent				
Dependencies: **Telemetry Adapter (net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryAdapter),** **MQTT Messaging Connection (net.powermatcher.core.messaging.mqttv3.Mqttv3Connection or net.powermatcher.expeditor.messaging.mqttv5.Mqttv5Connection)**				
Property	**Type**	**Req.**	**Default**	**Description**
id	String	Y	-	Component instance id e.g. ‘auctioneer1’
cluster.id	String	N	DefaultCluster	Cluster id or sometimes also referred to as *namespace id*.
enabled	boolean	N	true	If true component will not be activated when all dependencies are resolved. Otherwise if false it will never be activated.
update.interval	int	N	30	Number of seconds of message interval.
measurement.logging.pattern	String	N	'measurement_log_'yyyyMMdd'.csv'	Define the measurement logging pattern.
status.logging.pattern	String	N	'status_log_'yyyyMMdd'.csv'	Define the status logging pattern.
date.format	String	N	yyyy-MM-dd HH:mm:ss	Logging date format
list.separator	String	N	;	Separator of logged data.

5.2 Library components

This section describes the components in the agent library, their configuration properties and the dependencies.

5.2.1 Clipping Concentrator

A special type of concentrator component, the Clipping Concentrator, can clip the power consumption of its group to a certain configurable level. The Clipping Concentrator has the following properties in addition to the properties of the Concentrator listed in section 6.1.2 of the PowerMatcher Core Installation and Configuration manual.

Component: Clipping Concentrator				**Type: factory component**
Class: net.powermatcher.core.agent.concentrator.ClippingConcentrator				
Dependencies: **Matcher Adapter** **Agent Adapter** **Peak Shaving Adapter (optional)**				
Property	**Type**	**Req.**	**Default**	**Description**
peak.shaving.enabled	boolean	N	false	Defines if peak shaving is enabled
power.lower.limit	double	N	-	Default lower limit in Watt, required if peak shaving enabled.
power.upper.limit	double	N	-	Default upper limit in Watt, required if peak shaving enabled.
immediate.update	boolean	N	true	Update bid and price immediately for a dynamic change in constraints via the peak shaving adapter interface.

5.2.2 Peak Shaving Concentrator

A special type of concentrator component, the Peak Shaving Concentrator, can reduce the power consumption of its group to a certain configurable level. The difference with the Clipping Concentrator is that the Peak Shaving Concentrator transposes the price and aggregated bid curve as a whole. The Peak Shaving Concentrator has the following properties in addition to the properties of the Concentrator listed in section 6.1.2 of the PowerMatcher Core Installation and Configuration manual.

Component: Peak Shaving Concentrator				**Type: factory component**
Class: net.powermatcher.agent.peakshavingconcentrator.PeakShavingConcentrator				
Dependencies: **Matcher Adapter** **Agent Adapter** **Peak Shaving Adapter (optional)**				
Property	**Type**	**Req.**	**Default**	**Description**
peak.shaving.enabled	boolean	N	false	Defines if peak shaving is enabled
power.lower.limit	double	N	-	Default lower limit in Watt, required if peak shaving enabled.
power.upper.limit	double	N	-	Default upper limit in Watt, required if peak shaving enabled.
immediate.update	boolean	N	true	Update bid and price immediately for a dynamic change in constraints via the peak shaving adapter interface.

5.3 *Extension Component Adapters*

This section describes the available adapters for PowerMatcher Extension components.

5.3.1 Direct Telemetry Adapter Factory

The Direct Telemetry Adapter Factory creates telemetry adapters that connect directly to a telemetry logging component. The adapter enables agents and matchers (and other components) to log telemetry events directly to a telemetry logging component running in the same runtime.

Component: Direct Logging Adapter Factory				**Type: factory component**
Class: net.powermatcher.core.direct.protocol.adapter.component.DirectTelemetryAdapterFactory				
Dependencies: -				
Property	**Type**	**Req.**	**Default**	**Description**
id	String	Y	-	Factory id e.g. ‘directTelemetryAdapterFactory’.
cluster.id	String	N	DefaultClu ster	PowerMatcher cluster id.

Note that it is possible to configure both a Direct Telemetry Adapter Factory as well as a Telemetry Adapter Factory that uses a messaging connection for the same component. In that case the telemetry event will be published both over a direct connection as well as over a messaging connection.

5.3.2 Telemetry Adapter Factory

The Telemetry Adapter is an adapter for sending telemetry data over a messaging connection.

Component: Telemetry Adapter Factory				**Type: factory**
Class: net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryAdapterFactory				
Dependencies: -				
Property	**Type**	**Req.**	**Default**	**Description**
id	String	Y	-	Factory id e.g. ‘telemetryAdapterFactory’.
cluster.id	String	N	DefaultClu ster	Cluster id or sometimes also referred to as *namespace id*.
connector.id	String	N		If a connector.id is specified, the messaging connection will use connector.id instead of id as the client id for the MQTT connection. This allows a single MQTT client connection to be shared, for the purpose of scalability, between multiple agents running in the same runtime environment.
messaging.adapter.factory	String	Y	mqttv3Con nectionFac tory	The adapter factory for creating the messaging connection.

5.3.3 Telemetry Listener Adapter Factory

The Telemetry Listener Adapter is an adapter for receiving telemetry data from a messaging connection.

Component: Telemetry Listener Adapter Factory				**Type: factory**
Class: net.powermatcher.telemetry.messaging.protocol.adapter.TelemetryListenerAdapterFactory				
Dependencies: -				
Property	**Type**	**Req.**	**Default**	**Description**
id	String	Y	-	Factory id e.g. ‘telemetryListenerAdapterFactory’.
cluster.id	String	N	DefaultCluster	Cluster id or sometimes also referred to as *namespace id.*
messaging.adapter.factory	String	Y	mqttv3ConnectionFactory	The adapter factory for creating the messaging connection.

5.3.4 MQTT v5 Connection Adapter Factory

The MQTT v5 Connection Adapter Factory creates messaging connections for adapters that use messaging. The adapter implements the MQTT version 5 protocol. The MQTTv5 client connection that is created by the adapter factory is shared between adapters that have the same connector.id (the default for the connector.id is the id of the component).

Component: MQTT v5 Connection Adapter Factory				**Type: factory component**
Class: net.powermatcher.expeditor.messaging.mqttv5.Mqttv5ConnectionFactory				
Dependencies: -				
Property	**Type**	**Req.**	**Default**	**Description**
id	String	Y	-	Factory id e.g. ‘mqttv5ConnectionFactory’.
cluster.id	String	N	DefaultCluster	Cluster id or sometimes also referred to as namespace id.
broker.uri	String	N	local://MicroBroker	Broker URI. The local URL denotes a direct connection to a broker in the same runtime. Use tcp://localhost:1883 for a network connection to the broker.
broker.username	String	N	-	Broker username. Only applicable when broker security is enabled.
broker.password	String	N	10	Broker password. Only applicable when broker security is enabled.
notification.enabled	boolean	N	false	Send notification messages.
hostname	String	N	-	Hostname of the MQTT client.
component.name	String	N	-	Component name.
notification.topic	String	N	Status/Agent	Topic name notification messages.
notification.connected.message	String	N	CONNECTED\\,I\\,${cluster.id}\\,${id}\\,${component.name}\\,${component.name}\\,${host.name}1883	Format of notification message.
notification.cleandisconnected.message	String	N	CLEANDISCONNECTED\\,W\\,${cluster.id}\\,${id}\\,${component.name}\\,${component.name}\\,${host.name}	Format of clean disconnect message.
notification.disconnected.message	String	N	DISCONNECTED\\,I\\,${cluster.id}\\,${id}\\,${component.name}\\,${component.name}\\,${host.name}	Format of disconnect message caused by an exception.
reconnect.interval	int	N	10	Retry interval to (re)connect.

6. Appendix B: OSGi runtime installation and setup

This chapter describes the additional steps required for a runtime installation that includes PowerMatcher Extension components.

6.1.1 Install the required bundles

A PowerMatcher application in OSGi requires a set of libraries. The `required-bundles` directory will contain all the bundle jars that are required. Some prerequisite libraries are not delivered with the release, but have to be acquired individually. For PowerMatcher Extension, copy the additional jar libraries listed in Table 4 to this directory for applications that are using MicroBroker as the MQTT broker:

Function	JAR file	Description	Download
MicroBroker	com.ibm.micro	IBM MicroBroker. Commercial MQTT broker implementation. Delivered with the IBM Lotus Expeditor product.	See [2] and paragraph 4.1.
	com.ibm.micro.admin		
	com.ibm.micro.client		
	com.ibm.micro.utils		
	com.ibm.micro.utils.extended		

Table 4 PowerMatcher required libraries for OSGi.

Note that when you set up a development environment when the configuration project ‘cnf’ is created by Bndtools the libraries listed in the table have either been downloaded by Bndtools in the repository, or have been added when setting up the workspace. You can copy the libraries from the workspace when you set up your runtime.

6.1.2 Install the application bundles

Next, copy all PowerMatcher Extensions bundle jars into the `required-bundles` directory.

JAR file	Description
net.powermatcher.agent.telemetry.logging.jar	Telemetry Logging bundle jar.
net.powermatcher.agent.telemetry.logging.component.jar	Telemetry Logging component bundle jar.
net.powermatcher.expeditor.broker.manager.jar	Broker manager implementation and bundle jar.
net.powermatcher.expeditor.messaging.mqttv5.component.jar	MQTT v5 bundle jar.
net.powermatcher.expeditor.messaging.mqttv5.jar	MQTT v5 wrapper implementation.
net.powermatcher.telemetry.messaging.protocol.adapter.component.jar	Messaging Telemetry adapter bundle jar.
net.powermatcher.telemetry.messaging.protocol.adapter.jar	Messaging Telemetry adapter implementation.
net.powermatcher.core.direct.protocol.adapter.component.jar	Direct Telemetry adapter bundle jar.
net.powermatcher.core.direct.protocol.adapter.jar	Direct Telemetry adapter implementation.

Table 5 Application bundles for PowerMatcher Extension applications.

7. REFERENCES

[1]	PowerMatcher Core – Installation and Configuration
[2]	Lotus Expeditor, product page: http://www-01.ibm.com/software/lotus/products/expeditor/