WebSphere MQ Telemetry Transport
Java Classes

Version 1.4.6

6 March, 2009

Property of IBM

Take Note!

Before using this report be sure to read the general information under "Notices".

March 2009

This edition applies to Version 1.4.6 of IA92 and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- MQSeries
- WebSphere

The following terms are trademarks of other companies:

- Java, Sun Microsystems

Summary of Amendments

Date	Changes
27 February 2003	Initial release
15 May 2003	Version 1.1 Contains improved samples for both the J2SE and J2ME MIDP environments, plus bug fixes to the protocol implementation.
30 November 2003	Version 1.2 Updates: • A persistence interface has been added to the client. • API return codes have been replaced by Exceptions • Combined OSGi and J2SE packages into one jar file for both environments. • Improved samples • Renamed all jar files to begin wmqtt instead of MQIsdp
16 April 2004	Version 1.3 Updates: • Added a linked exception to MQIsdpException to allow better reporting of errors. • Updated API so that all checked exceptions are MQIsdpException or its subclasses. Specific exceptions are linked to MQIsdpException. • Updated the MIDlet example so that it will work in both the MIDP-1.0 and MIDP-2.0 environments.
9 July 2004	Version 1.4 Updates: • API and package rename to match the protocol name (MQ Telemetry Transport). Main class has been renamed from ClientMQIsdp to MqttClient. • Subscribe and unsubscribe methods are now non-blocking with corresponding subscribed and unsubscribed callback methods in the MqttAdvancedCallback interface. • The client will throw an MqttNotConnectedException if the application tries to sending any data whilst it is not connected, including whilst connectionLost processing is occurring.
10 December 2004	Version 1.4.1 Updates: • Introduction of a batch window into the client. The API will block on publish, subscribe or unsubscribe if there are more than 50 unacknowledged messages in the process of being delivered to a broker. The API will unblock when the number of unacknowledged messages drops below 50. This increases overall performance because having large numbers of unacknowledged can cause a network to be

flooded with retries.

- Minor bug fixes

6 March 2009 Version 1.4.6 – minor bug fixes

Preface

This SupportPac provides a set of Java classes that implement the MQ Telemetry Transport protocol (MQTT, formerly known as MQIsdp). The classes provide a clean API that can be used to quickly MQTT-enable Java applications.

MQTT – WebSphere MQ Telemetry Transport

MQIsdp – MQ Integrator SCADA Device Protocol

SCADA – Supervisory, Control And Data Aquisition

Chapter 1. Introduction

This SupportPac provides a Java implementation of the client side of the WebSphere MQ Telemetry Transport publish/subscribe protocol. The API is encapsulated in one class, which contains verbs such as connect, publish, subscribe and unsubscribe for communicating with WebSphere Business Integration Message Broker.

The code with this SupportPac is packaged in to work in three different operating environments – J2SE, OSGi and J2ME_MIDP. The SupportPac contains two subdirectories called J2SE and J2ME_MIDP, which contain code as follows:

- **J2SE**
  wmqtt.jar contains the API implementation as documented below packaged for the J2ME CDC/Foundation and above environment.
  wmqttTraceFormat.jar contains trace formatting utilities. This jar file does not need to be present at runtime. It is only required on the machine on which the trace is to be formatted.
  wmqttSample.jar contains a sample WebSphere MQ Telemetry Transport application which has a Swing user interface. The source code for this application is supplied in package com.ibm.mqttsample.utility

- **OSGi**
  The J2SE wmqtt.jar is packaged with a manifest file that enables the code to be deployed onto the OSGi Service Platform. The OSGi bundle does not implement any services. It simply makes the J2SE classes available in the OSGi environment. See http://www.osgi.org for more information about OSGi.

- **J2ME_MIDP**
  wmqtt.jar contains the API implementation as documented below for the J2ME CLDC/MIDP 1.0 and above environment. The API is identical to that on the J2SE platform.
  wmqttSample.jar contains a sample MIDlet application that provides a J2ME user interface. The source code for this MIDlet is supplied in package com.ibm.mqttsample.midpapp

Chapter 2. MqttClient Java class and the programming model

The MQ Telemetry Transport protocol is accessible via a single class called com.ibm.mqtt.MqttClient, which is in the wmqtt jar file. This class provides methods for interfacing to WebSphere Business Integration Message Broker such as publish, subscribe and unsubscribe. There are also methods for setting attributes of the MQ Telemetry Transport connection, such as timeouts and retries. There is also a callback interface so that an application can be notified when events occur such as a publication arriving or a publication send completing.

An application should implement one of the callback interfaces to provide functionality as appropriate.

Any references to “message broker” include the following products:

WebSphere Message Broker V6.0
WebSphere Event Broker V6.0
WebSphere Message Broker V6.1

Programming model

After instantiating the MqttClient class the application can set any session parameters, such as the retry interval (setRetry()) and the call back interface(registerSimpleHandler() or registerAdvancedHandler()). The application should then call one of the connect methods to establish a connection with the message broker.

There is no limit on the number of times an application may connect and disconnect, but each instance of the class can only have one connection at a time.

When an application has finished using the MqttClient object it should call the terminate() method to shut down all threads started by the class. After the terminate method has been called no API methods may be used. A new instance of the class must be instantiated before a new connection can be established.

Callback methods may be invoked whilst the application is running. A complete list is defined in section Callback methods. Two important methods that should be implemented are publishArrived and connectionLost.

- publishArrived must be implemented if the application needs to receive publications.
- connectionLost must be implemented to handle the MQTT connection breaking. Typically the connectionLost method should invoke the connect method to reconnect. If the cleanstart flag is true then the connectionLost method should resubscribe for any topics the application is interested in because the broker will have automatically removed any previous subscriptions.

To implement the callback methods a class must implement one of the callback interfaces and register itself as the class that will handle the callback events.

Persistence

The MqttPersistence interface helps ensure that publications are delivered/received and are protected against machine failure. See the accompanying javadoc for the API to view the MqttPersistence interface and the MqttFilePersistence sample implementation.

Once a publish API call completes the application can rely on the persistence interface to protect the data and ensure that it gets delivered/received to/from the broker.

Application failure during publish
If the application fails during a publish API call then the outcome of the publish is unknown. Prior to reconnecting to the broker the application should query the persistence implementation to check if its state is consistent with the application by comparing message id's that each last knew about. If the persistence implementation is holding a message id 1 greater than the application then the message was persisted before the publish failed and the application need take no action. Otherwise the application should resend the message as the persistence has no knowledge of it.

Persistence and clean session
If the clean session flag is used on connect then both the broker and client ends of the connection will reset their state at disconnect time. To ensure that QoS 1 and 2 publications are delivered as expected use the MqttClient.outstanding() method or the MqttClient.published() callback method to confirm that all messages have been delivered prior to calling disconnect.

Chapter 3. Com.ibm.mqtt.MqttClient.java

Method documentation

See the accompanying Javadoc documentation.

Callback methods

Callback methods are invoked when particular events occur. Default callback methods are supplied in the MqttClient class, which do nothing.

If applications want to be more sophisticated and react to events occurring in the underlying protocol then there are a number of ways of doing this:

- The MqttClient class can be extended and a subset or all of the default callback methods can be overridden.
- A class can implement the MqttSimpleCallback interface and register itself using the registerSimpleHandler method
- A class can implement the MqttAdvancedCallback interface and register itself using the registerAdvancedHandler method

The methods in the MqttSimpleCallback interface are connectionLost and publishArrived as defined below.

- **publishArrived** must be implemented if the application needs to receive publications.
- **connectionLost** must be implemented to handle the MQTT connection breaking. Typically the connectionLost method should invoke the connect method at intervals until the connection is re-established, or the application decides to give up.
  If the cleanstart flag is true then the connectionLost method should resubscribe for any topics the application is interested in because the broker will have automatically removed any previous subscriptions.

The methods in the MqttAdvancedCallback interface are the methods in the MqttSimpleCallback interface plus published, subscribed and unsubscribed.

- **Published** is invoked when the broker acknowledges receipt of a Quality of Service 1 or 2 publication. It is not invoked for QoS 0 publications because the broker does not acknowledge receipt of these.
- **subscribed** is invoked when a subscribe is acknowledged by the broker.
- **unsubscribed** is invoked when an unsubscribe is acknowledged by the broker.

Registering a callback interface

To receive callback events another class can be registered to be notified of callback events. There is a simple and advanced callback interface. The class which implements one of these callback interfaces must include ‘implements MqttSimpleCallback’ or ‘implements MqttAdvancedCallback’ as part of the class declaration.

public void registerSimpleHandler(MqttSimpleCallback simpleCallback)

public void registerAdvancedHandler(MqttAdvancedCallback advCallback)

Diagnostics

Exceptions

Exceptions will be thrown for both user error and runtime exceptions. For user error exceptions the exception getMessage() method of the exception may yield some useful information.

MqttException and all its subclasses have a getLinkedException method which will return the specific exception that caused the error.

Explicit Mqtt exceptions are:

MqttException
MqttPersistenceException
MqttNotConnectedException
MqttBrokerUnavailableException

All these exceptions are documented in the doc\api directory of this SupportPac.

Trace

public void startTrace()
public void stopTrace()

The startTrace() and stopTrace() methods control the collection of trace if required. A binary trace file called mqe0.trc is generated in the current directory. This trace file may be moved to another system, or formatted in situ using the wmqttTraceFormat jar file.

To format the trace execute:

```
java – jar wmqttTraceFormat.jar mqe0.trc
```

Or place wmqttTraceFormat.jar in the classpath and execute:

```
java com.ibm.mqtt.trace.MQeTraceFromBinaryFile mqe0.trc
```

Formatted trace will be written to stdout.

NOTE: Tracing is only available in the J2SE implementation. The startTrace() and stopTrace() methods will have no effect in the J2ME MIDP environment.

Chapter 4. Using the sample applications

J2SE sample

The jar file J2SE\wmqttSample.jar contains a sample swing user interface for publish/subscribe, which uses the MQTT Java classes supplied in this SupportPac. The source code for this user interface is supplied for reference purposes in package com.ibm.mqttsample.utility.

To run the user interface: Make sure wmqttSample.jar and wmqtt.jar are in the same directory. Then execute: java –jar wmqttSample.jar

Or

Place wmqttSample.jar and wmqtt.jar in the classpath. Then execute Java com.ibm.mqttsample.utility.MQTTFrame

Compiling and packaging

The source code for the sample is provided in com\ibm\mqttsample\utility Use the following java utilities to compile and package the sample application:

- Compile the code using the following javac command line:

  javac -d <build output directory> com\ibm\mqttsample\utility*.java

- To package the code as a jar file execute the following jar command line. The manifest file specifies that the com.ibm.mqttsample.utility.MQTTFrame class contains the main method for the jar file.

  jar cvfm wmqttSample.jar com\ibm\mqttsample\utility\MANIFEST.MF
  \com\ibm\mqttsample\utility*.class com\ibm\mqttsample\persistence*.class

Navigating the user interface

Connection

Specify the TCP/IP address and port number of the SCADAInput node of your message broker. Prior to connecting the following options can be set by clicking the options tab:

Trace Start/Stop – Trace may be started and stopped at any time. A binary trace file will be produced in the current directory. See the section on Diagnostics.

Client Identifier – The application identifier that the MQTT protocol uses to connect

Clean Session – In the event of the MQTT connection unexpectedly terminating, should the message broker remove all subscriptions and publications for the previously connected client.

Keep Alive - If the message broker does not receive any data within this interval Seconds) it will assume the client application has stopped functioning. The MQTT Java classes automatically manage keeping the connection alive, providing the TCP/IP connection is alive, by sending a MQTT ping message.

Retry Interval - The time interval at which messages will be retried in seconds. This parameter also controls the length of time the client waits for an acknowledgement from the broker when connecting.

Use Persistence – Use the MqttFilePersistence implementation of MqttPersistence when running the protocol.

Persistence directory – The directory beneath which data should be persisted if persistence is being used.

Last Will and Testament - Specify a topic and data that should be published by the broker in the event of the MQTT connection being terminated unexpectedly.

Subscriptions

Subscribe Topic and Request QoS – Specify a topic to subscribe to or unsubscribe from and the Quality of Service at which the application wants publications delivered to it for this topic.

Received Topic – When the application receives a publication from the broker it displays the topic in this field.

Save… - Save the last message received to disk. If the message is binary then the message will still be saved correctly, even though it is not displayed sensibly in the user interface.

Hex/Text - Toggle the display between showing the data as text in the system code page or as hexadecimal values of the received bytes.

Publications

Topic, QoS and retained – Publish a message on this topic, at the request Quality of Service. Also specify whether the publication should be retained by the broker.

File… - Read data in from a file to be published. If the data is binary then the message will be published correctly, even though it is not displayed sensibly in the user interface.

Hex/Text - Toggle the display between showing the data as text in the system code page or as hexadecimal values of the received bytes.

J2ME MIDP

The jar file J2ME_MIDP\wmqttSample.jar contains a sample lcdui user interface for publish/subscribe, which uses the MQTT Java classes supplied in this SupportPac. The source code for this user interface is supplied for reference purposes in package com.ibm.mqttsample.midpapp.

The MIDP implementation needs to support TCP/IP sockets in order for the sample to run. Sockets are compulsory in MIDP-2.0 and optional in MIDP-1.0. See the javax.microedition.io.Connector documentation to determine if sockets are supported in your environment.

To run the user interface: Copy wmqttSample.jar and wmqttSample.jad onto the MIDP device.

For IBM's j9 embedded java on PocketPC the shortcut syntax to launch the MIDlet using the CLDC/MIDP-2.0 profile is below. This assumes the jar and jad file are installed in \wmqtt.

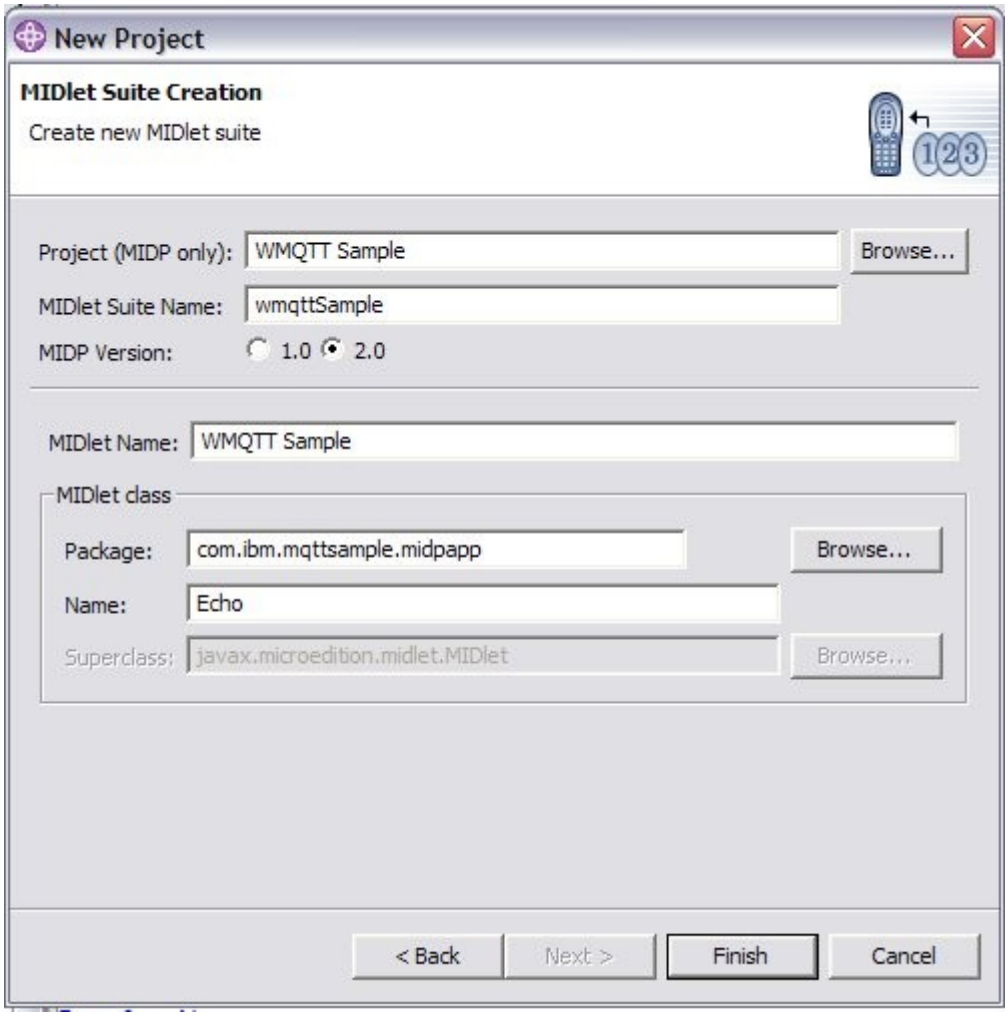
176#j9.exe -jcl:mpng:loadlibrary=ivempng20
-Xbootclasspath:\ive\lib\jclMidpNG\classes.zip;\wmqtt\wmqttSample.jar
javax.microedition.lcdui.AppManager \wmqtt\wmqttSample.jad

Compiling and packaging

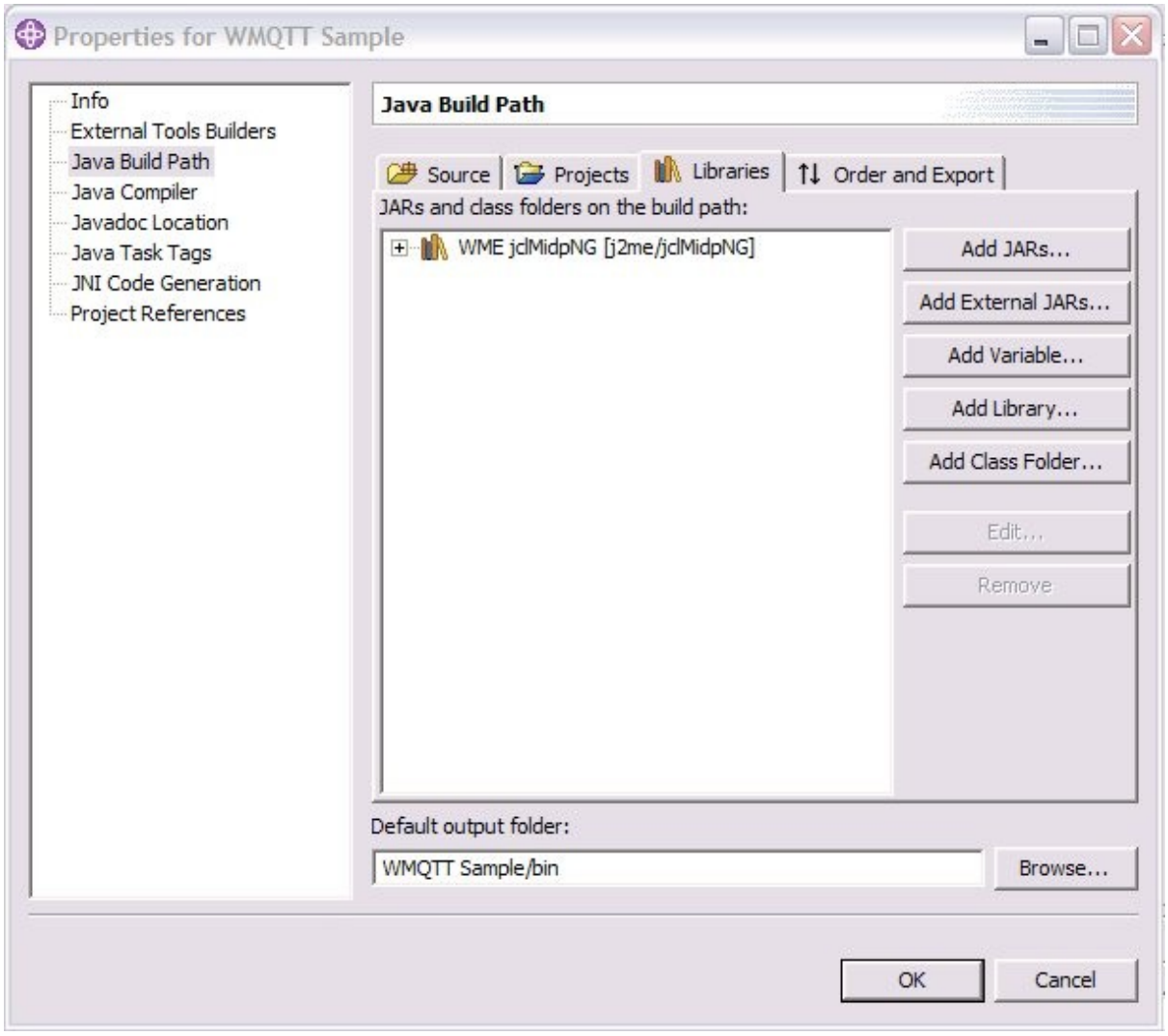
The code is supplied precompiled, but if you wish to recompile it here are the instructions. To compile the application you need Sun's J2ME Wireless Toolkit 2.0 which is available from http://java.sun.com or IBM's WebSphere Studio Device Developer.

WebSphere Studio Device Developer V5.6

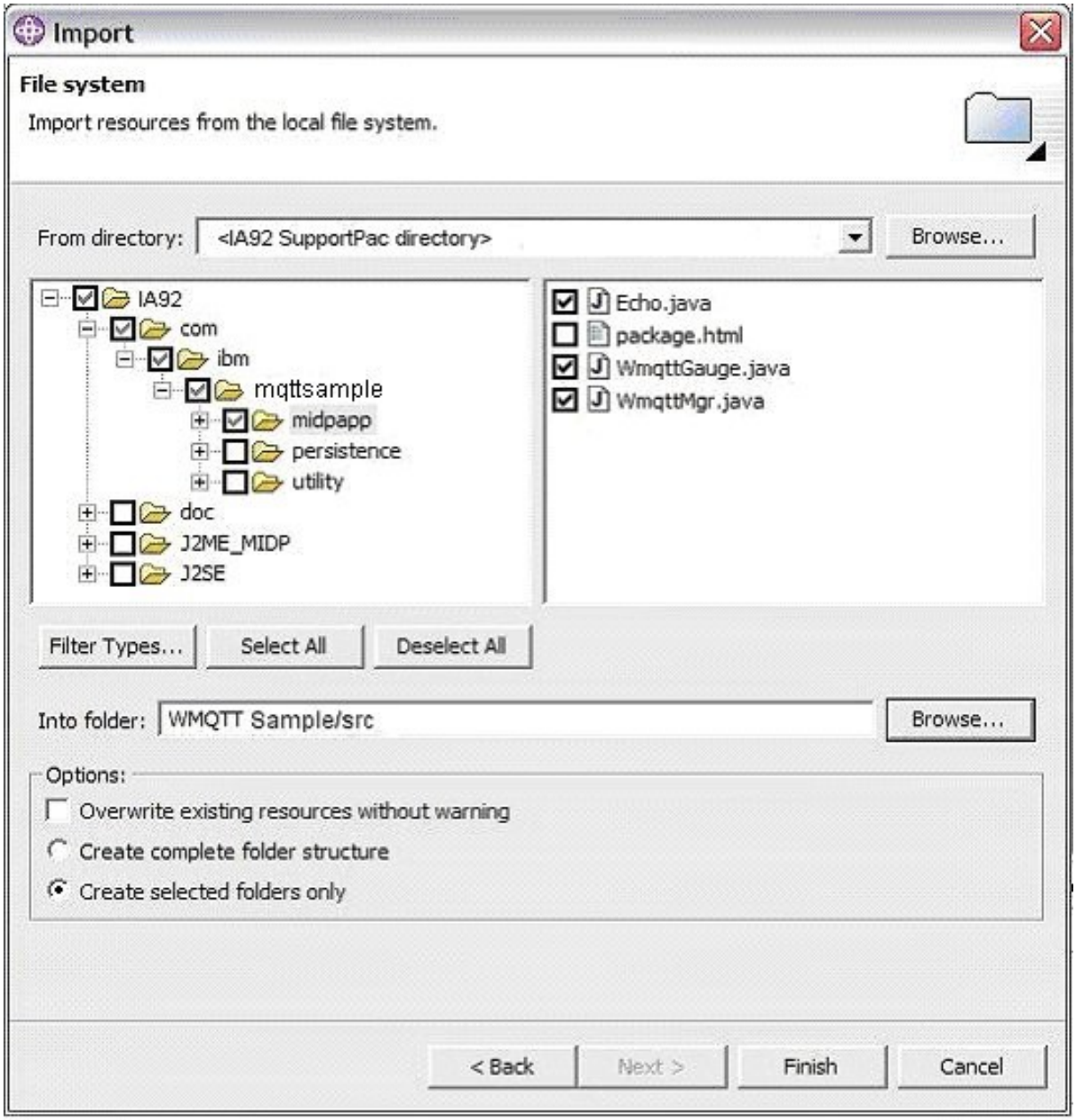
Create a J2ME MIDlet Suite project. In the package explorer panel right mouse New->Project->J2ME->MIDlet Suite. Fill in the dialog as shown. MQTT will run in a MIDP1.0 or higher environment provided sockets are supported.



Next, add the J2ME MIDP wmqtt.jar to the build path, as shown in the dialog below. Right mouse on the project and select properties. In the dialog shown below select the Libraries tab within the Java Build Path properties. Click on the Add External JARs button and select <IA92 SupportPac>\J2ME_MIDP\wmqtt.jar to add to the build path.



Now import the sample application, as shown in the dialog below. Right mouse on the 'WMQTT Sample' project and select import from the filesystem. Set the 'From directory' to be the root directory <IA92 SupportPac>. Select all the classes in the com.ibm.mqttsample.midpapp package.



Finally create builds for the target platform(s) that your MIDlet will run on. Right mouse on the project and select Device Developer Builds. A build target called 'Generic Build' may already exist. If not then create it.

Once the build is created a 'generic' subdirectory will appear in the project. Edit the file with extension .jxeLinkOptions.

1. On the 'Input' tab make sure that <IA92 SupportPac>/J2ME_MIDP/wmqtt.jar appears in the class search path. If it doesn't use the 'New…' button to add it.
2. On the 'In/exclusion' tab select 'Include whole classes' from the drop down and add com.ibm.mqtt.midp.MqttMidpSocket to the list. This class is dynamically loaded, so needs to be explicitly included when WSDD compiles the code.

Security
By default WSDD 5.6 does not allow MIDlets to create socket conections. The security policy for MIDlets is in a file called *security.policy* in the lib directory of the j9 runtime. When running MIDLets

within WSDD 5.6 on Windows the runtime security policy is in file <*Device Developer 5.6 install path*>*\wsdd5.0\ive-2.1\runtimes\win32\x86\ive\lib\jclMidpNG\uei\lib\security.policy*.

To allow MIDlets running in the untrusted domain to prompt for socket access add the following line to the security.policy file under the section for the untrusted domain:
session(session): javax.microedition.io.Connector.socket
This will cause the MIDlet to prompt for permission once each time the MIDlet is run.

J2ME Wireless Toolkit 2.0

Create a project called wmqttSample and copy the J2ME_MIDP\wmqtt.jar file into the lib subdirectory of the project. Copy the java source in com\ibm\mqttsample\midpapp into the src directory of the project. The resulting directory structure should be src\com\ibm\mqttsample\midpapp*.java.

You can then build the J2ME application using the KToolbar user interface. Selecting package from the project menu will create a jar and jad file that can be used on a MIDP compliant device.

Navigating the user interface

Before running the MIDlet make sure that the MIDP device is able to make a TCP/IP connection to a message broker.

The sample MIDlet simply echoes any publications it receives to the device screen and a response topic. It subscribes to topic midlet/echo/request and publishes the echo to topic midlet/echo/response.

Pressing cancel at any point will take the MIDlet back to the previous screen. Pressing cancel prior to connecting will simply close the MIDlet.

Connection

Specify the TCP/IP address and port number of the SCADAInput node of your message broker. Optionally you can change the Client Identifier which the MIDlet will use to identify itself to the broker.

The MIDlet will pop up an informational message after successfully connecting to the broker and subscribing for data. Pressing cancel after connecting will cause the MIDlet to unsubscribe and disconnect from the broker.

Echoing Publications

To send data to the MIDlet start up the J2SE Sample application as described above and subscribe to topic midlet/echo/response.

Publish a message to topic midlet/echo/request. The data published should appear on the screen of the device as well as being echoed to the response topic.

---------- End of Document ----------