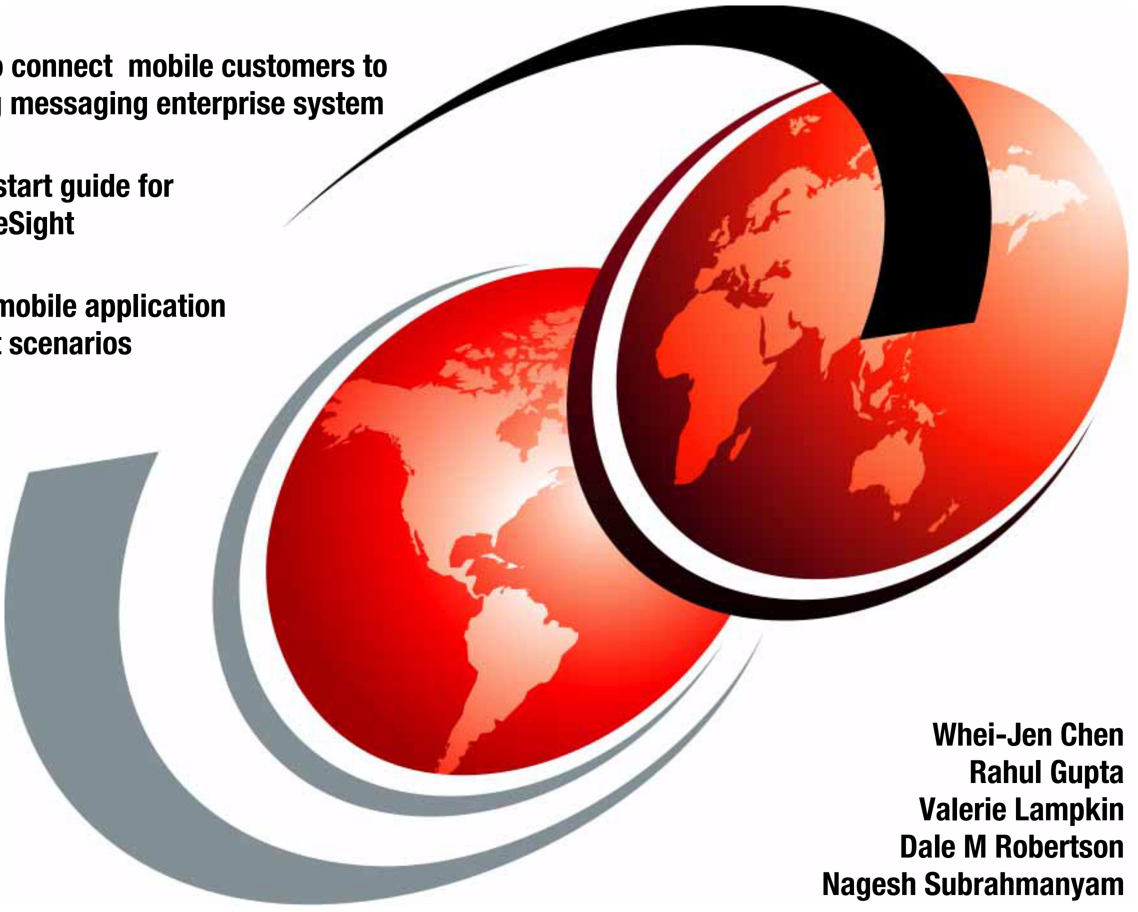
IBM

Responsive Mobile User Experience Using MQTT and IBM MessageSight



Learn how to connect mobile customers to your existing messaging enterprise system

See a quick start guide for IBM MessageSight

Understand mobile application development scenarios

Whei-Jen Chen
Rahul Gupta
Valerie Lampkin
Dale M Robertson
Nagesh Subrahmanyam

Redbooks

ibm.com/redbooks

International Technical Support Organization

Responsive Mobile User Experience Using MQTT and IBM MessageSight

March 2014

SG24-8183-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (March 2014)

This edition applies to IBM MessageSight Version 1.0.0.1, IBM WebSphere MQ Version 7.5.1, IBM Integration Bus Version 9

Contents

Notices

Trademarks

Preface

IBM® MessageSight is an appliance-based messaging server that is optimized to address the massive scale requirements of machine-to-machine (m2m) and mobile user scenarios. IBM MessageSight makes it easy to connect mobile customers to your existing messaging enterprise system, enabling a substantial number of remote clients to be concurrently connected.

The MQTT protocol is a lightweight messaging protocol that uses publish/subscribe architecture to deliver messages over low bandwidth or unreliable networks. A publish/subscribe architecture works well for HTML5, native, and hybrid mobile applications by removing the wait time of a request/response model. This creates a better, richer user experience.

The MQTT protocol is simple, which results in a client library with a low footprint. MQTT was proposed as an Organization for the Advancement of Structured Information Standards (OASIS) standard. This book provides information about version 3.1 of the MQTT specification.

This IBM Redbooks® publication provides information about how IBM MessageSight, in combination with MQTT, facilitates the expansion of enterprise systems to include mobile devices and m2m communications. This book also outlines how to connect IBM MessageSight to an existing infrastructure, either through the use of IBM WebSphere® MQ connectivity or the IBM Integration Bus (formerly known as WebSphere Message Broker).

This book describes IBM MessageSight product features and facilities that are relevant to technical personnel, such as system architects, to help them make informed design decisions regarding the integration of the messaging appliance into their enterprise architecture.

Using a scenario-based approach, you learn how to develop a mobile application, and how to integrate IBM MessageSight with other IBM products. This publication is intended to be of use to a wide-ranging audience.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), San Jose Center.

Whei-Jen Chen is a Project Leader at the ITSO, San Jose Center. She has extensive experience in application development, database design and modeling, and IBM DB2® system administration. Whei-Jen is an IBM Certified Solutions Expert in Database Administration and Application Development, and an IBM Certified IT Specialist.

Rahul Gupta is an Advisory IT Architect with IBM Global Technology Services® (GTS) in the US. He is a Certified service-oriented architecture (SOA) Architect with nine years of professional experience in IBM messaging technologies. At his current assignment, he works as a middleware consultant for various clients in North America. His core experiences are in lab testing, performance tuning, and Level 3 development for both IBM Integration Bus and WebSphere MQ products. Rahul has been a technical speaker for messaging-related topics at various WebSphere conferences, and is a recognized inventor by the IBM innovation community.

Valerie Lampkin is a Middleware Technical Resolution Specialist in the US. She has 14 years of experience supporting WebSphere MQ, previously as part of IBM Global Services and now with the Application and Integration Middleware Division of IBM Software Group. She holds a B.S. degree from Florida State University. She is a regular contributor to the WebSphere and IBM CICS® Support blog on IBM developerWorks®.

Dale M Robertson is a Client Technical Professional (CTP) with the US East IMT team. He specializes in IBM Connectivity and Integration products, such as WebSphere MQ, MQTT, IBM MessageSight, and the IBM Integration Bus. He has spent an embarrassingly long time in IT, and over fifteen years working with WebSphere MQ and the various versions of WebSphere Message Broker.

Nagesh Subrahmanyam is an IBM Certified Consulting IT Specialist in India. He has over 10 years of experience in the field spanning IBM z/OS®, WebSphere MQ, IBM Integration Bus, and XML. With his experiments on devices, such as Arduino and RaspberryPi, he has a deep interest in Internet of Things (IoT) and m2m technologies. He has contributed articles on the IBM developerWorks website, co-authored IBM Redbooks publications about XML and z/OS, and submitted WebSphere SupportPacs. He holds a bachelor's degree in mechanical engineering from the National Institute of Technology, Jamshedpur.

Acknowledgements

The authors would like to thank Joe Speed for the book topic, directions, and technical consultation for this book.

The authors would also like to recognize Peter Niblett, Mark Bluemel, and Graham Hopkins from IBM UK for their work done on the article *Using MQ Telemetry Transport Protocol in IBM Worklight® Mobile applications*. We refer to this article extensively when we describe developing the IBM Worklight hybrid application for MQTT.

Thanks to the following people for their contributions to this project:

Dave Locke
Peter Niblett
Nicholas O'Leary
Andrew Schofield
Andrew Banks
Richard J Coppen
Paul J Lacy
IBM United Kingdom

David Dhuyvetter
IBM US

IBM MessageSight development team in Austin, Texas

Sowmya H Dayananda
IBM India

Now you can become a published author, too!

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form:

  ibm.com/redbooks

- Send your comments in an email:

  redbooks@us.ibm.com

- Mail your comments:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction

This chapter provides information about the importance of mobile response time in today's business environment, and about the technologies available to achieve real-time user mobile experiences.

This chapter shows you how the MQTT protocol can be implemented with IBM MessageSight to achieve a business solution.

This chapter includes the following topics:

- Challenges
- IBM product solutions:
  - IBM MessageSight
  - IBM WebSphere MQ
  - IBM Worklight
  - IBM Integration Bus (formerly WebSphere Message Broker)
  - IBM InfoSphere® Streams
- MQTT advantages (versus HTTP)
- IBM MessageSight user examples

1.1 Challenges

As technology evolves, more devices are connected to each other across the Internet, evolving into what has often been referred to as the *Internet of Things* (IoT). It seems that we, as a society, are constantly connected through smartphones, notebooks, and tablets.

However, it is not only people who are sending data across the internet. Machines can also be producing, collecting, and sending data. The expansion of interconnected devices creates a demand for systems to receive and quickly process an ever-growing amount of data, and users expect a real-time response.

The enhancements in technology make our daily tasks easier by allowing people to perform tasks on the go, and not requiring access to the old desktop computer. Even activities that used to require face-to-face access, such as monitoring homes and health, can now be done remotely, thanks to technological advances.

This enables us to interact with information in more intimate ways than before, but it creates a demand on business systems to handle an increasing number of connections, and to provide analytical processing with quick response times.

One of the biggest challenges for today's marketplace is the evolution of mobile devices and the user experience. Previously, a phone only talked to other phones, and sent data when the user instructed it to send information.

Today, a phone or tablet can connect to almost anything, and share and receive information automatically. The user experience is constantly evolving, and requires a system that can facilitate the transports of data in near real time to optimize the user experience.

Most of today's edge connectivity follows a similar pattern, whereby devices collect data for central processing. Then decisions are made centrally and pushed out to devices. To enhance the mobile user experience, you need to add intelligence throughout the network.

The expanding mobile world has created a need for businesses to expand beyond their existing messaging enterprise. Massive amounts of data need to be processed in real time and kept secure.

The old request/response model does not work for many of today's current scenarios. To ensure that the user experience is satisfactory, messaging applications must be able to deliver secure, reliable behavior in an unreliable or intermittently connected wireless environment.

You can take the following approaches simultaneously:

- Enable systems to collect information, distill it, and then push it up to traditional central systems for processing.
- Use the information gathered from remote devices, and use MQTT connectivity to push information back down, applying central intelligence combined with contextual information to make things happen.

  For example, a promotion can be targeted at a particular consumer at exactly the correct time based on the customer's previous preference and current geolocation.

These are ways that you can use data today, which can make a huge difference to a business and its ability to meet customer expectations.

1.2 IBM product solutions

To help businesses meet the demands of today's interconnected world, IBM has a suite of messaging products. By using the IBM portfolio of messaging products, you can create a solution that extends from back-end enterprise applications to include millions of mobile users.

IBM MessageSight is one of the latest additions to the IBM messaging family. It is an appliance-based messaging server optimized to address the massive scale requirements of the machine-to-machine (m2m) and mobile use cases. IBM MessageSight is designed to sit at the edge of the enterprise, and can extend your existing messaging infrastructure or be used stand-alone.

1.2.1 IBM MessageSight

IBM MessageSight joins the IBM portfolio of middleware to help bridge back-end enterprise environments to remote *smarter* clients as the planet becomes more digitally instrumented, interconnected, and intelligent.

This enables organizations to provide an interactive experience with their users, and offer real-time analytics of large data volumes, as shown in Figure 1-1.

Figure 1-1 IBM MessageSight extends your enterprise to remote wireless clients

IBM MessageSight is built for high performance to offer persistent, transactional messaging. The hardware is 2U form factor, eight 1 GbE and four 40 GbE ports. IBM MessageSight includes built-in security, and makes integration with external Lightweight Directory Access Protocol (LDAP) security systems possible.

IBM MessageSight enables you to sense and respond to data coming from the edge of your enterprise. Messaging connectivity can be achieved using MQTT, Java Message Service (JMS), or WebSphere MQ. The ability to connect through MQTT makes the appliance ideal for use with mobile clients. Administration can be done with a web user interface (web UI) or using the command line.

The high-availability pair takes the form of a primary/standby configuration. The primary node continually replicates both the message store and the appliance configuration information to the standby node.

If the primary node fails, the standby node has the latest data that is needed for applications to continue messaging services. The standby node does not accept connections from application clients or provide messaging services until a failover operation occurs.

IBM MessageSight offers mobile application support, and can handle massive scaling of concurrent device connectivity and communication, offering high-performance messaging. The appliance is DMZ-ready, with no user-level operating system, and encrypted flash and storage media, allowing you to securely extend your existing messaging enterprise. The device can act as the gateway for business data flowing in and out of your enterprise's network.

IBM MessageSight is developer-friendly, designed for easy deployment and easy integration. This book explores how IBM MessageSight can be integrated with other IBM offerings to deliver a complete messaging solution that encompasses remote mobile clients and back-end enterprise systems.

With the versatile IBM MessageSight, you can use either the MQTT protocol for low latency publishing and subscribing (ideal for m2m), or JMS to transfer messages received from remote clients to back-end applications.

Figure 1-2 shows a few examples of how clients connected to IBM MessageSight can interface with WebSphere MQ and other back-end applications. In this diagram, IBM MessageSight connects many users and devices on the Internet to services that are deployed on an intranet. The users, devices, and services interact with each other by exchanging messages through IBM MessageSight.

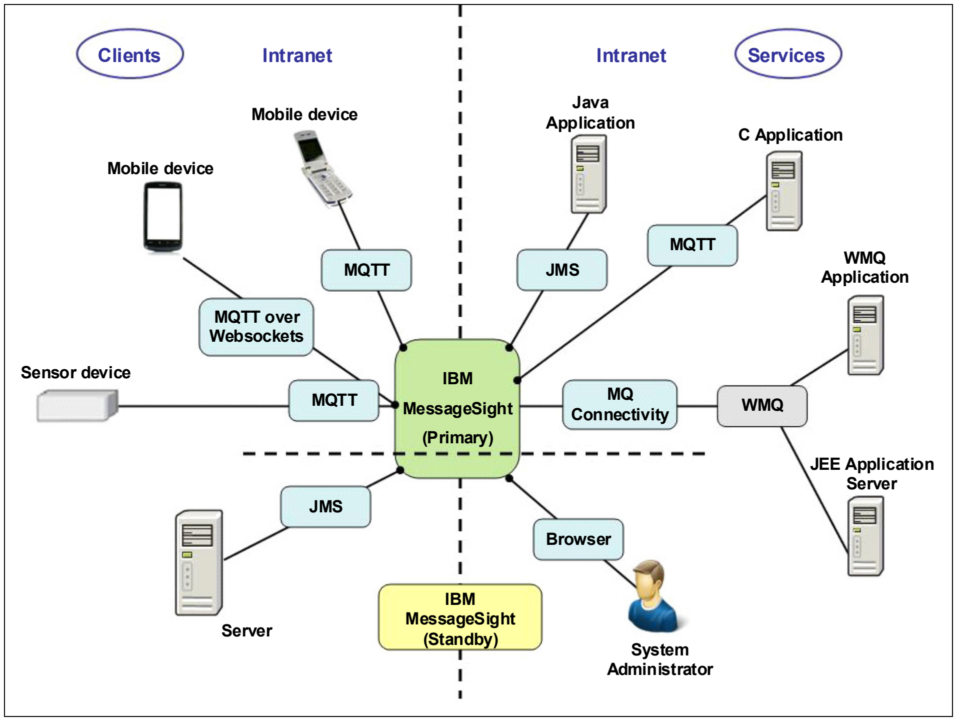


Figure 1-2 Typical IBM MessageSight connectivity designs

1.2.2 IBM WebSphere MQ

IBM WebSphere MQ has been the top message queuing software for two decades. It offers assured delivery of asynchronous messages, and can be an integral part of critical business architecture. Including WebSphere MQ in your topology enables a diversity of messaging styles. Applications can send data point-to-point, or use the publish/subscribe model.

A variety of application languages might be used, including C, COBOL, Java, and JMS, just to name a few. WebSphere MQ includes a telemetry feature that uses the reliability of WebSphere MQ messaging functions to be extended to include messages from a huge array of endpoints, ranging from client applications to applications running on mobile devices, Android phones, and even small sensors, radio frequency identifications (RFIDs), and telemetry devices.

The IBM MessageSight MQ Connectivity feature enables message data flow received from remote wireless devices to be funneled to a queue manager. The connection from IBM MessageSight to the queue manager is through a client channel, which can be secured using Secure Sockets Layer (SSL) or Transport Layer Security (TLS). A single user ID is defined on the queue manager for IBM MessageSight authentication.

1.2.3 IBM Worklight

IBM Worklight is a member of the MobileFirst family of products that provides a platform for the development of mobile applications. It includes a collection of application programming interfaces (APIs) that can be used to develop hybrid Worklight applications. Using the Worklight APIs, you can create web applications that are tailored to the mobile environment.

IBM Integration Bus, in conjunction with IBM Worklight, can also provide your mobile applications with secure access to your back-end systems, and integration that is robust and can scale to handle a growing number of connections.

1.2.4 IBM Integration Bus (formerly WebSphere Message Broker)

IBM Integration Bus V9.0 is the latest version of what was previously known as WebSphere Message Broker. The principal function of a message broker is to mediate communications between applications by providing capabilities for tasks, such as message validation, message transformation, message enhancement, and message routing.

IBM Integration Bus goes beyond the normal routing, transformation, or message aggregation, enabling you to solve most integration requirements between application systems, from simple point-to-point connectivity to more complex topologies. It has a wide range of built-in connectors that enable integration between queues, web and Representational State Transfer (REST) services, files, databases, and packaged applications.

IBM Integration Bus provides a range of flexible deployment options:

- Standard operating systems:
  - Linux
  - Microsoft Windows
  - IBM AIX®
  - IBM z/OS
- Virtualized systems:
  - AIX Hypervisor
  - VMware
- Public clouds:
  - IBM SmartCloud®

IBM Integration Bus is a significant evolution of the WebSphere Message Broker technology base, with continuous innovation to incorporate an increasing number of integration use cases.

The previous WebSphere Message Broker versions can be used in conjunction with IBM MessageSight by incorporating a WebSphere MQ queue manager between IBM MessageSight and the broker. IBM Integration Bus V9 has the added benefit of enabling you to configure a direct JMS publish/subscribe connection from IBM Integration Bus to IBM MessageSight, without the need for an intermediate queue manager.

IBM Integration Bus includes two comprehensive patterns that provide production-ready, bidirectional connectivity between back-end systems and connected IBM MessageSight clients:

- The *Event Filter pattern* uses IBM Integration Bus to receive events from mobile clients through IBM MessageSight, filter the events based on specified criteria, and forward events that match the criteria to a back-end system.
- The *Event Notification pattern* uses IBM Integration Bus to receive events from a back-end system, and dynamically publish those events to mobile clients through IBM MessageSight.

1.2.5 IBM InfoSphere Streams

Extracting insight from an immense volume and variety of data is now possible with IBM InfoSphere Streams. *Data in motion* is one of the fundamental aspects of big data. Being able to harness and use data as it flows through a system is essential for today's business world. InfoSphere Streams allow real-time analytic processing of big data. Integrating IBM InfoSphere Streams into your enterprise system enables massive volumes of data to be ingested and analyzed as it is received.

Rule-based decisions can be made as the data comes in, triggering actions based on the information received. For example, an influx of data can be received using IBM MessageSight, and funneled to InfoSphere Streams.

Through stream computing, InfoSphere Streams can determine what response or downstream processing is needed, and start it accordingly. InfoSphere Streams is able to ingest a variety of data, from simple text to images and geospatial information. It then uses its powerful analytics to link and evaluate the data as it is received then take appropriate actions.

1.3 MQTT advantages (versus HTTP)

The MQTT protocol is optimized for networks with limited processing capabilities, small memory capacities, or high latency. Compared to HTTP, MQTT imparts several advantages to mobile applications:

- Faster response times
- Higher throughput
- Higher messaging reliability
- Lower bandwidth usage
- Lower battery usage

MQTT's efficient use of network and battery power, along with its integration with enterprise messaging middleware, makes it ideal for scenarios where an enterprise application must push data or interact with one or more mobile applications.

HTTP is designed as a request/response protocol for client/server computing, not necessarily optimized for mobile and push capabilities, particularly in terms of battery usage. MQTT helps overcome problems experienced by HTTP, by offering reliable delivery over fragile networks. MQTT will deliver message data with the required quality of service (QoS) even across connection breaks.

MQTT includes several useful features:

Last Will and Testament	All applications know immediately if a client disconnects ungracefully.
Retained messages	Any user reconnecting immediately gets the latest business information.

The lightweight MQTT protocol enables publish/subscribe messaging to reliably send data to phone or tablet applications without having to code retry logic in the application. The MQTT protocol can be used to create an interactive user experience with any mobile platform.

IBM Worklight is an example of an application development tool that enables you to create mobile applications with MQTT quickly and easily.

An actual example where implementing the MQTT protocol rather than HTTP was beneficial is the case of a prominent social networking company. The company had experienced latency problems when sending messages. The message delivery was reliable but slow with their previous method.

A new mechanism was needed that can maintain a persistent connection with the messaging servers without using too much battery power. This capability is critical to users of the company's social networking site, because so many of them use the service on battery-powered mobile devices.

The company's developers solved the problem by using the MQTT protocol. By maintaining an MQTT connection and routing messages through its chat pipeline, the company was able to achieve message delivery at speeds of just a few hundred milliseconds, rather than multiple seconds.

1.4 Mobile user experience scenarios with MQTT and IBM MessageSight

In today's world, there are many practical instances where MQTT and IBM MessageSight implementations can offer a solution for an integrated system that enables users to interact using mobile devices. In this section, you examine a few possible scenarios.

1.4.1 Connected car

IBM MessageSight can enable customers to connect and interact with their cars. A *connected car* can use the MQTT protocol to send messages from the car to IBM MessageSight and then notify the customer, as shown in Figure 1-3.

For example, a connected car is a car that is able to send diagnostic information to the car company. Also, the car is able to receive messages, which might range from remotely locking the car to a request to send its current location.

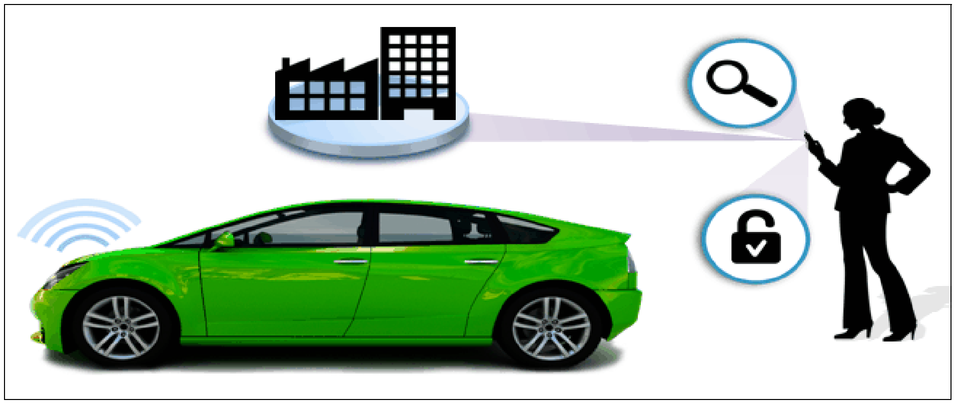


Figure 1-3 Connected car

MQTT and IBM MessageSight facilitate message routing, enabling the car to send diagnostic information to the car company. The car essentially acts as a rolling sensor platform publishing telematic events. The MQTT protocol is used to transport data from the car sensors to the automobile company.

The data can be analyzed using predictive maintenance analytics, acting as a virtual mechanic that sends a notification to the customer that service is needed before a component fails.

Also, the car is able to receive messages, which might range from remotely locking the car, setting the climate controls (heat or air), or requesting that the car send its current location. Most existing connected car solutions have previously taken a mobile response time measured in 10s of seconds. Tests with MessageSight using MQTT have shown the response time to be less than a second, equal to the time it takes to push a key fob.

1.4.2 Connected city

If you take the connected car scenario a step further, you can see how having the MQTT messaging features available within many cars can effectively translate into a *connected city*. If the car were involved in an accident that caused the airbag to deploy, it triggers an event to be published, as shown in Figure 1-4.

Publish/subscribe messaging enables different users to receive the alert that an accident has happened. It can be generic (to inform other drivers of the location of the accident), or it can be specific (to route only to the car owner's family), and so on. The alert might also be sent to emergency services, to alert police or medics of the accident.

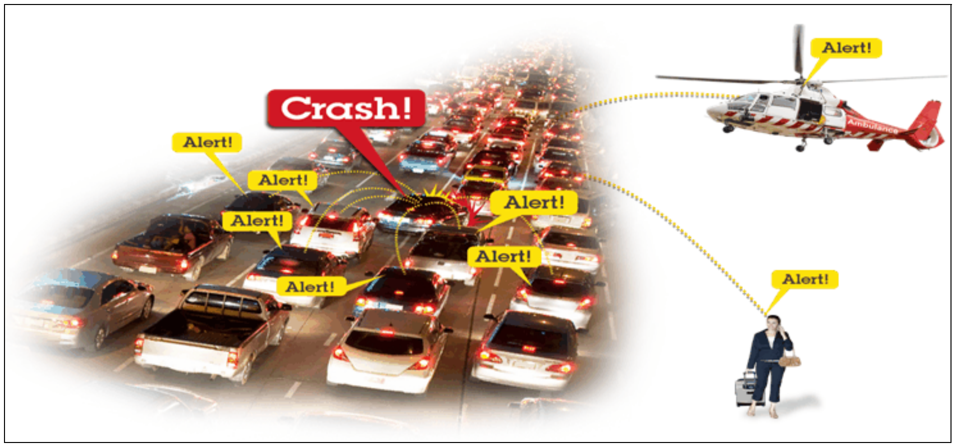


Figure 1-4 Connected city

For the connected city example, it is easy to see how millions of cars sending messages can create a massive amount of data to be processed. To date, analyzing this volume of data has presented a problem. Using IBM MessageSight in conjunction with IBM InfoSphere Streams stream computing helps alleviate this dilemma by allowing real-time analytics to be performed on big data.

1.4.3 Connected home

The interactive user experience can also apply to the home. Figure 1-5 shows a scenario where changing a channel on the TV creates a message that is sent back to the data center, which in turn determines how advertising might be catered specifically to the consumer currently watching TV.

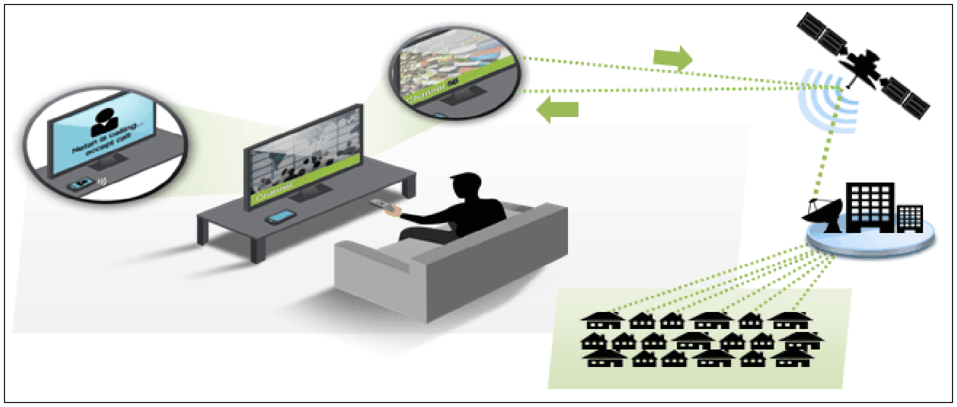


Figure 1-5 A connected home

Other convenient features of a *connected home* might be the ability for the homeowner to adjust the thermostat or unlock the door using his mobile device. These types of features not only offer convenience, but also help contribute to a smarter planet, making it possible for utilities to be adjusted to lower the usage when not needed. The ability to remotely manage door locks and utilities can apply to a rental or vacation home property as well.

1.4.4 Connected retail consumers

Retailers are able to provide a unique shopping experience tailored to a customer's location and shopping preferences. Bidirectional communication between a retailer's back-end systems and the customer's mobile devices enables retailers to provide notifications to consumers based on the customer's proximity to the store (Figure 1-6).

Figure 1-6 Connected retail consumers

For example, if a customer is browsing a product at home on his mobile device, and later enters the store, he can use the retailer's mobile application to find where the product is located within the store. The retailer might even want to push a notification for a sale or discount on that product or another product while the customer is browsing in the store.

The retailer can enable business rules to handle database calls, analytics, pricing, and so on, to cater the notifications based on the individual consumer, even the consumer's current geographic location. The retailer's central office is able to monitor the millions of messages using IBM MessageSight.

2

IBM MessageSight and MQTT

The integration of the IBM MessageSight appliance with the MQTT protocol provides reliable, secure messaging features that can be scaled to over one million concurrent remote client connections. The IBM MessageSight appliance can extend the reach of an existing enterprise messaging network.

This chapter describes how the MQTT protocol is used by the IBM MessageSight appliance, and includes a simple startup configuration. MQTT is ideal for mobile applications because of its small size, minimized data packets, and efficient distribution of information to one or many subscribers.

For the examples shown in this chapter and throughout this IBM Redbooks publication (unless otherwise noted), the WebSphere MQ software is installed and configured on a Linux 64-bit system.

This chapter provides information about the following topics:

- MQTT protocol
- Features of IBM MessageSight
- Getting started
- Configuration and administration

2.1 MQTT protocol

MQTT is an extremely simple and lightweight messaging protocol. The publish/subscribe architecture is designed to make it easy to build loosely-coupled HTML5, native, and hybrid mobile applications. MQTT is ideal for use in constrained environments where network bandwidth is low, or where there is high latency, and with remote devices that might have limited processing capabilities and memory.

MQTT minimizes network bandwidth and device resource requirements when attempting to ensure reliability and delivery. This approach makes the MQTT protocol particularly well-suited for your business applications to interact with remote devices, such as mobile phones.

The MQTT protocol enables you to extend connectivity beyond your enterprise boundaries to smart devices. The protocol is efficient on battery, processor, and network bandwidth in mobile scenarios.

The MQTT protocol includes the following highlights:

- Open and no-charge for easy adoption. MQTT is open to make it easy to adopt and adapt for the wide variety of devices, platforms, and operating systems that are used at the edge of a network.
- A publish/subscribe messaging model that facilitates one-to-many distribution.
- Ideal for constrained networks (low bandwidth, high latency, data limits, and fragile connections). MQTT message headers are kept as small as possible. The fixed header is just two bytes, and its on-demand, push-style message distribution keeps network use low.
- Multiple service levels allow flexibility in handling different types of messages. Developers can designate that messages will be delivered at most one time, at least once, or exactly once.
- Designed specifically for remote devices with little memory or processing power. Minimal headers, a small client footprint, and limited reliance on libraries make MQTT ideal for constrained devices.
- Easy to use and implement, with a simple set of command messages. Many applications of MQTT can be accomplished using just the **`CONNECT`**, **`PUBLISH`**, **`SUBSCRIBE`**, and **`DISCONNECT`** commands.
- Built-in support for loss of contact between client and server. The server is informed when a client connection breaks abnormally, enabling the message to be resent, or preserved for later delivery.

MQTT was co-invented by IBM and Eurotech. It was designed specifically for sending data over networks where connectivity is intermittent or bandwidth is at a premium. The protocol is simple, making it extremely efficient for reliable publish/subscribe messaging. MQTT enables devices to open a connection, keep it open using little power, and receive events or commands using a small two-byte header.

The MQTT protocol is built upon several fundamental concepts, all aimed at assuring message delivery and keeping the messages themselves as lightweight as possible. The following list includes a few features of the MQTT protocol:

- Publish/subscribe

  The MQTT protocol is based on the principle of publishing messages and subscribing to topics, which is typically referred to as a publish/subscribe model. Each message is published to a specific named topic. Clients who want to receive messages can make subscriptions against the topics that interest them.

- Topics and subscriptions

  Messages in MQTT are published to topics, which can be thought of as subject areas. Clients, in turn, sign up to receive particular messages by subscribing to a topic. Subscriptions can be explicit, which limits the messages that are received to the specific topic at hand, or you can use wildcard designators, such as a number sign (#) to receive messages for a variety of related topics.

  For example, topic strings might include `scores/football` and `scores/cricket`, where a subscriber uses the score number to receive scores of all sports.

- Quality of service (QoS) levels

  MQTT defines three QoS levels for message delivery:

  - QoS 0: At most once
  - QoS 1: At least once
  - QoS 2: exactly once

  Each level designates a higher level of effort by the server to ensure that the message gets delivered. Higher QoS levels ensure more reliable message delivery, but might use more network bandwidth.

- Retained messages

  Retained messages is an MQTT feature where the server keeps the messages and delivers them to future subscribing clients. A retained message enables a client to connect and receive the retained message as soon as it creates the subscription, without waiting for a new publication.

- Clean sessions and durable connections

  There are two types of MQTT clients:

  - Those which the server remembers when they disconnect
  - Those which the server does not remember

  When an MQTT client connects to the server, the client indicates which type of client to use by setting the *clean session* flag. If the clean session flag is set to `true`, all of the client's subscriptions are removed when the client disconnects from the server.

  If the flag is set to `false`, the connection is treated as durable, and the client's subscriptions remain in effect after any disconnection. In this event, subsequent messages that arrive carrying a high QoS designation are stored for delivery after the connection is reestablished. Using the clean session flag is optional.

- Will messages

  When a client connects to a server, it can inform the server that it has a *will message* that is published to a specific topic or topics in the event of an unexpected disconnection. A will message is particularly useful in settings where system managers must know immediately when a remote sensor has lost contact with the network

2.1.1 Eclipse Paho project

The Eclipse Paho project is aimed at developing open source, scalable, and standard messaging protocols. The scope of the project includes client software for use on remote devices, along with corresponding server support. It is focused on new, existing, and emerging applications for machine-to-machine (m2m) and Internet of Things (IoT).

The Eclipse Paho project is part of a broader m2m initiative at the Eclipse Foundation, to provide open source tools and protocols to simplify development using MQTT.

The Eclipse Paho project includes these major goals:

- Bidirectional messaging
- Determinable delivery of messages
- Loose coupling
- Constrained-platform usability

The Eclipse tools facilitate designing and developing connectivity solutions between devices and applications, thereby enabling and encouraging more innovative integration with remote devices through the MQTT protocol.

You can find more information about the Eclipse Paho project at the following website:

`http://www.eclipse.org/paho/`

2.1.2 OASIS

The Organization for the Advancement of Structured Information Standards (OASIS) Technical Committee for the standardization of MQTT was established for organizations from around the world to collaborate and develop a standardized version of the MQTT protocol. IBM and Eurotech, the original authors of the MQTT protocol, have brought MQTT into the OASIS open standards process.

OASIS plans to expand the range of enterprise solutions by facilitating integration with business applications and increasing connectivity to different types of networks and remote devices.

The OASIS MQTT technical committee is producing a standard compatible with MQTT V3.1, together with requirements for enhancements, documented usage examples, leading practices, and guidance for the use of MQTT topics with commonly available registry and discovery mechanisms.

The standard supports the following functions:

- Bidirectional messaging to uniformly handle both signals and commands,
- Deterministic message delivery
- Simple QoS levels
- Always/sometimes-connected scenarios
- Loose coupling
- Scalability to support large numbers of devices

Candidates for enhancements include the following functions:

- Message priority and expiry
- Message payload typing
- Request/reply
- Subscription expiry

For more information about the MQTT technical committee, see the following website:

`https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt`

2.2 Features of IBM MessageSight

IBM MessageSight delivers massive scale communication to extend your existing enterprise to include interactions from remote clients, such as mobile phones, sensors, machines, and other applications. The IBM MessageSight appliance delivers performance and scalability, enabling organizations to meet the demands of an always-connected world, and users who want an interactive experience. These are a few of the features that IBM MessageSight delivers:

- Ease of deployment

  Device can be up and running in 30 minutes. The User Interface helps guide administrators through the first steps.

- Developer friendly

  Simple yet powerful programming interfaces make application development easy. A simple paradigm of connect, subscribe, and publish promotes loosely coupled and scalable applications.

- Easy integration

  IBM MessageSight supports Java Message Service (JMS), WebSockets, and the MQTT protocol. It has built-in connectivity with WebSphere MQ (making it possible to connect to multiple back-end queue managers). It can be directly connected to IBM Integration Bus using JMS nodes.

- Secure

  Messaging policies allow you to filter for specific access. Options are available to add Secure Sockets Layer/Transport Layer Security (SSL/TLS), including Federal Information Processing Standard (FIPS) 140-2.

- Reliable

  Two devices can be connected to provide high availability without shared resources. Various options for QoS, including assured delivery.

- Optimized for wireless clients

  Efficient MQTT messaging protocol that is faster, and requires less bandwidth and less battery than traditional HTTP. Event-oriented paradigm provides for a better customer experience. The developer-friendly application programming interfaces (APIs) and libraries can be used for a variety of mobile platforms. IBM MessageSight and MQTT integrate easily with IBM Worklight.

- Massive scale

  One appliance can handle one million concurrent connections. This enables massive fan-out streaming of data, processing 13 million non-persistent messages per second. When assured delivery matters, the device can process 400 thousand persistent messages per second.

The IBM MessageSight appliance is designed to sit at the edge of the enterprise, and it can extend an existing messaging infrastructure or be used stand-alone. It handles the load and concentrates the data to the back-end applications (Figure 2-1).

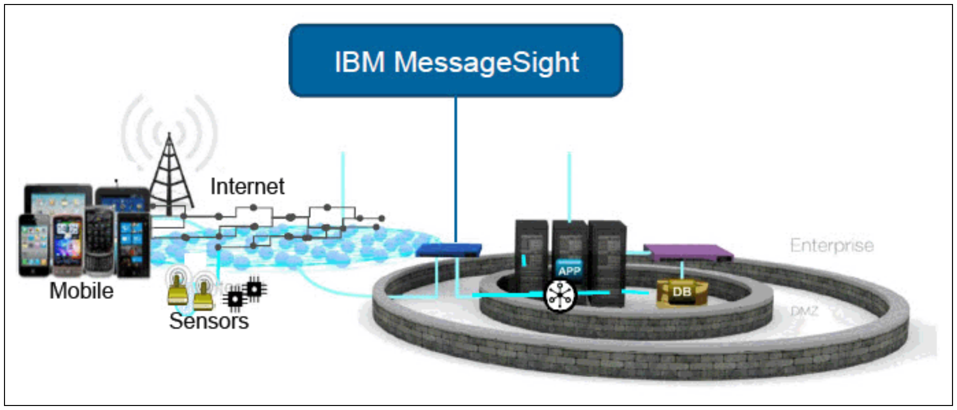


Figure 2-1   IBM MessageSight sits at the edge of the Enterprise to allow interactions with remote devices

The IBM MessageSight appliance supports client applications using protocols. At the time of publishing, IBM provides the following clients for those protocols:

- MQTT over TCP/IP:
  - MQTT C client
  - MQTT client for Android
  - MQTT client for Java
  - MQTT client for iOS
- MQTT over WebSockets:
  - MQTT client for JavaScript
- JMS:
  - IBM MessageSight JMS client

IBM MessageSight V1.1, released at the end of November, 2013, provides several enhancements to the appliance. These enhancements include:

- Updates to the appliance browser-based GUI, including a dashboard that provides an at-a-glance view of system status.
- A JMS resource adapter, which facilitates direct connection between WebSphere Application Server and the appliance using message-driven beans.

- Shared subscriptions in which the stream of messages for a single subscription can be shared among multiple consumers. This can be considered a type of message load sharing among consumers.
- Shared subscriptions are available for WebSphere Application Server consumers and conventional JMS clients.
- Appliance security policies can access information contained inside client certificates, and can be used for authentication and authorization.
- Appliance security can be configured to use National Institute of Standards and Technology (NIST) SP 800-131A security recommendations.
- A new feature for mobile applications using MQTT called *disconnected client notification* provides a mechanism whereby the IBM MessageSight appliance can use a gateway service to send a notification to platform-specific notification mechanisms, such as Apple Push Notification service (APNs) or Google Cloud Messaging (GCM).

  Through the use of this feature, a disconnected mobile application can be notified that there are messages for it at the IBM MessageSight server.

Due to the timing of the release, the author team was not able to take advantage of any of the new features for use in the sample scenarios in this book.

2.3 Getting started

To set up your IBM MessageSight appliance, follow these steps:

1. After the appliance is installed in the rack, you must connect an Ethernet cable between the management port (labeled `mgt0`) and the network that will be used by the administrator.
2. Connect a second Ethernet cable between the port labeled `eth0` and the network to be used by the messaging clients. Figure 2-2 on page 23 shows the locations of the `mgt0` and `eth0` ports, which are located on the back of the IBM MessageSight appliance.

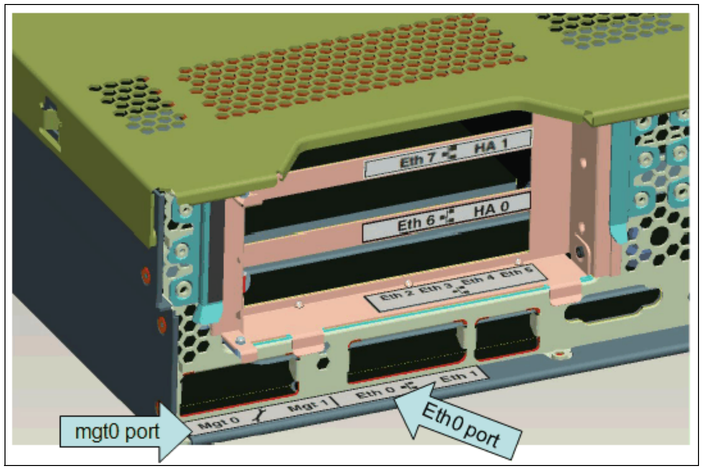


Figure 2-2 Port locations on IBM MessageSight appliance

3. Connect the VGA cable of the monitor to the VGA port, and connect the USB cable of the keyboard to one of the available USB ports on the appliance. You are then ready to turn on the appliance. It might take a few minutes for the startup to complete.
4. Log in with the default user ID (`admin`) and default password (`admin`).
5. In the initial setup wizard prompts, select which interface to configure. The default interface is `mgt0`.
   a. Choose whether to use Dynamic Host Configuration Protocol (DHCP). You must enter `yes` or `no` (abbreviations are not accepted).
   b. If you are not using DHCP, enter the IP address and gateway.
   c. After you press Enter to accept, the URL to connect through the web user interface (web UI) is displayed on the panel.

Tip: If you realize that you made a mistake, type the following text at a command line and make the necessary corrections:

```
edit ethernet-interface mgt0
```

6. After configuring the ports, you must accept the license. The IBM MessageSight appliance provides the web UI, which is intended for most administration and monitoring of the appliance. You can use the web UI to accept the license:

   a. To accept the license, access the web UI through a browser using the URL (`https://<IPAddress>:9087`) that was displayed when you configured the management and ethernet ports.

   b. If you do not know the web UI address and port, use the following commands to find the web UI address and port:

   ```
   imaserver get WebUIHost
   imaserver get WebUIPort
   ```

   c. The default login ID and password for the IBM MessageSight administrator are `admin` and `admin`.

   d. View the Software licensing and click **I Agree**.

7. You are then presented with the web UI **First Steps** tab. You can change the default password for the admin and configure the range for Ethernet client connections.

8. To save the configuration, click **Save** and **Close**.

2.4 Configuration and administration

The IBM MessageSight web UI is intended for most administration and monitoring of the appliance. Most of the functions available in the web UI are accessible with a command-line interface (CLI) as well.

2.4.1 Command-line interface

The IBM MessageSight appliance provides an extensive CLI, which includes a few utilities and diagnostic tools not available through the web UI.

You can access the IBM MessageSight CLI locally using a keyboard, video, and mouse (KVM) console, or over the network. If you are physically in the room with the IBM MessageSight appliance, you can access its CLI if you know an administrator name and password.

From the KVM console, log in with the administrator name and password. The default name and password are both `admin`. You can use the IBM MessageSight KVM console to display the machine type and serial number, as shown in Example 2-1 on page 25.

Example 2-1 Displaying the machine type and the serial number

```
Console>show version
Installation date: May 10, 2013 2:24:35 PM
Platform version: 5.0.0.10
Platform build ID: build99-20130510-1044
Platform build date: 2013-05-10 17:27:55+00:00
Machine type/model: 6188SM1
Serial number: KLM0199
Firmware type: Production
```

To determine the IP address of the Ethernet interface, you can use the **`status ethernet-interface`** command on the IBM MessageSight appliance CLI. For example, for the Ethernet interface `mgt0`, enter the following command:

```
status ethernet-interface mgt0
```

To access the IBM MessageSight CLI over the network, record some information when at the KVM console.

Use the **`status netif`** command (Example 2-2) to find the IP address of the mgt0 interface. This is the management IP address that you can use to access the IBM MessageSight CLI over the network.

Example 2-2 Finding the IP address of the mgt0 interface

```
Console> status netif mgt0
mgt0     OpState:[Up]
        generic MTU:1500 carrier:true flags:UP BROADCAST RUNNING
MULTICAST
            index:5
        inet addr:192.168.1.2 flags:PERMANENT mask:255.255.255.0
            scope:GLOBAL
```

Also record the Secure Shell (SSH) key fingerprint of the IBM MessageSight appliance so that you can verify it when you first access the system over the network:

```
Console> sshkey fingerprint
```

To access the CLI over the network, use SSH to connect to the appliance as the administrator. On Windows, there are several SSH clients that you can use, such as PuTTY. Linux, OS X, and UNIX users can use the **`ssh`** command to access the management IP address as the administrative user:

```
sh> ssh Admin_User@IP_Address
```

The first time you access a system, you might be prompted to accept the SSH key fingerprint for the server. If you recorded the SSH key fingerprint at the IBM MessageSight KVM console, you can validate it now, or you can choose to proceed.

If the IBM MessageSight appliance was given a node name, you can display it to make sure you are accessing the correct system:

```
Console> nodename get
nodename is Doncaster
```

The IBM MessageSight CLI enables you to run any of the commands listed in the Command Reference section of the IBM MessageSight Information Center. See Example 2-3 for an example.

Example 2-3 Command example

```
Console> status imaserver
Status = Stopped
Console> imaserver start
Start IBM MessageSight server.
The IBM MessageSight is running in production mode.
Console> exit
```

Use the **`exit`** command when you are done with the IBM MessageSight CLI.

To display the firmware version using the CLI, run the **`show imaserver`** command. The firmware version is included (in this case, `1.0`) along with some build date and time information:

```
Console> show imaserver
IBM MessageSight version is 1.0 20130510-1400 2013-05-10 19:12
```

The IBM MessageSight appliance includes a powerful command that gathers detailed information about your appliance configuration, log files, and other important diagnostic information.

The **`platform must-gather`** command is for troubleshooting, and gathers information needed by the IBM MessageSight support team if you need to open a problem management record (PMR). Sending output from the **`platform must-gather`** command when you open a PMR helps the IBM MessageSight support team understand your problem more quickly.

To collect this output, complete the following steps:

1. Access the IBM MessageSight CLI.
2. Run the **platform must-gather** command to create a .tgz file containing diagnostic information. If you have a PMR open with IBM, include that number in the file name:

   ```
   Console> platform must-gather PMR-12345,67R,890-May10.tgz
   ```

3. Run the **file list** command. You see the file that you created, and a separate collect-pd.txt file, as shown in Example 2-4.

   Example 2-4 Listing file

   ```
   Console> file list
   collect-pd.txt 676421 bytes created May 10, 2013 11:04:42 PM
   PMR-12345,67R,890-May10.tgz 33598707 bytes created May 10, 2013
   11:04:56 PM
   ```

You can then send those files to IBM as part of a PMR problem ticket if you need assistance with troubleshooting.

2.4.2 The web UI

You can access the web UI of an IBM MessageSight appliance with the IP address, port, and administrator name and password. The default web UI IP address is the one assigned to the mgt0 interface, and the default port is 9087. However, you can change both values.

If you do not know the web UI address and port used by your system, you can obtain it at the CLI, as shown in Example 2-5.

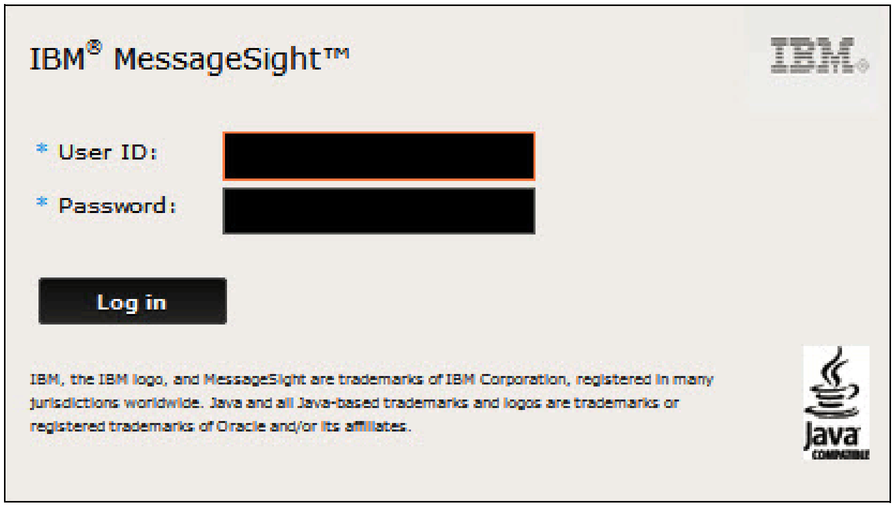
Example 2-5 Obtaining the web UI address

```
> imaserver get WebUIHost
WebUIHost = 192.168.1.2
> imaserver get WebUIPort
WebUIPort = 9087
```

Using one of the supported web browsers listed in the IBM MessageSight Information Center, connect to the web UI using the IP address and port in the following format:

```
https://<IPAddress>:9087
```

When you access the web UI through a browser, you are presented with a Login window to enter your User ID and password, as shown in Figure 2-3.

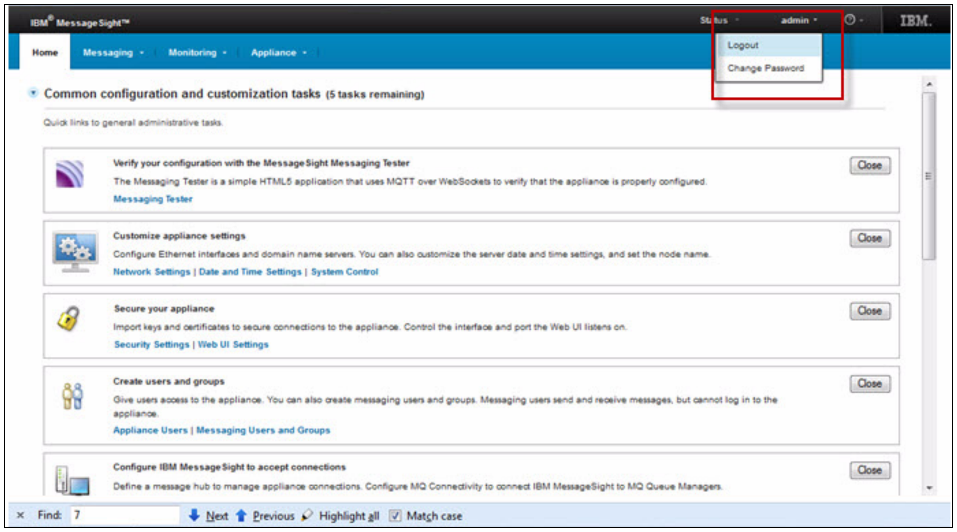


IBM, the IBM logo, and MessageSight are trademarks of IBM Corporation, registered in many jurisdictions worldwide. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Figure 2-3 Login window for the IBM MessageSight web UI

After you have successfully logged in to the IBM MessageSight appliance through the web UI, you can complete the Common configuration and customization tasks from the Home tab.

The web UI provides a user menu in the upper right corner (Figure 2-4). The icon for the menu is your user name. From the user menu, you can change your password or logout.



Common configuration and customization tasks (5 tasks remaining)

Quick links to general administrative tasks.

Verify your configuration with the MessageSight Messaging Tester

The Messaging Tester is a simple HTML5 application that uses MQTT over WebSockets to verify that the appliance is properly configured.

Messaging Tester

Customize appliance settings

Configure Ethernet interfaces and domain name servers. You can also customize the server date and time settings, and set the node name.

Network Settings | Date and Time Settings | System Control

Secure your appliance

Import keys and certificates to secure connections to the appliance. Control the interface and port the Web UI listens on.

Security Settings | Web UI Settings

Create users and groups

Give users access to the appliance. You can also create messaging users and groups. Messaging users send and receive messages, but cannot log in to the appliance.

Appliance Users | Messaging Users and Groups

Configure IBM MessageSight to accept connections

Define a message hub to manage appliance connections. Configure MQ Connectivity to connect IBM MessageSight to MQ Queue Managers.

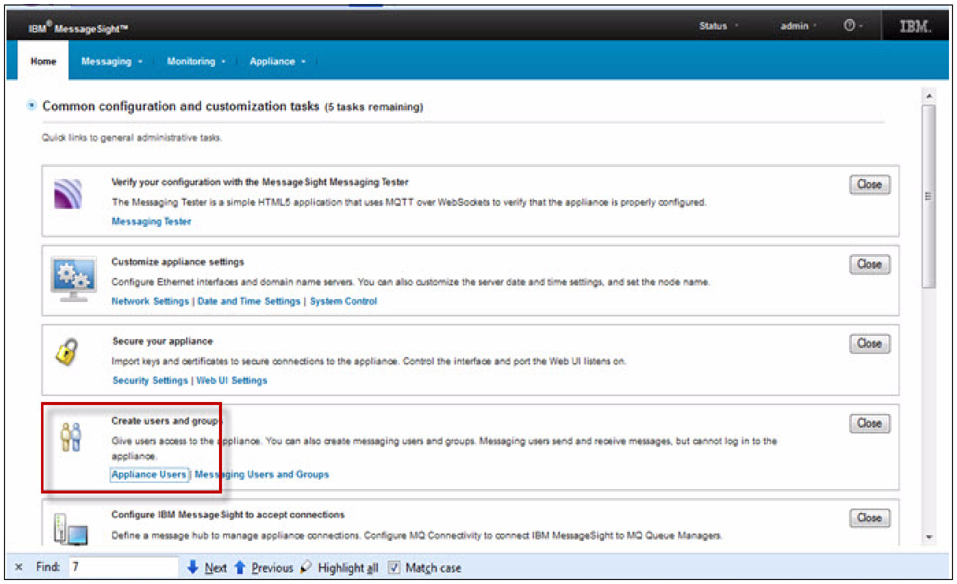
Figure 2-4 The web UI

There are three types of users that can be configured on the IBM MessageSight appliance:

- System administrator
- Messaging administrator
- Appliance users

System administrators are the only users who have access to the IBM MessageSight appliance. The system administrator can access IBM MessageSight using the web UI or the CLI.

The IBM MessageSight administration user interfaces (Figure 2-5) support role-based user actions. The system administrator role has access to every task on the appliance. Alternatively, the messaging administrator only has access to view or alter messaging-related tasks. For example, a messaging administrator can configure an endpoint, but is not able to configure the SSL/TLS certificates.



Common configuration and customization tasks (5 tasks remaining)

Quick links to general administrative tasks.

Verify your configuration with the MessageSight Messaging Tester
The Messaging Tester is a simple HTML5 application that uses MQTT over WebSockets to verify that the appliance is properly configured.
Messaging Tester

Customize appliance settings
Configure Ethernet interfaces and domain name servers. You can also customize the server date and time settings, and set the node name.
Network Settings | Date and Time Settings | System Control

Secure your appliance
Import keys and certificates to secure connections to the appliance. Control the interface and port the Web UI listens on.
Security Settings | Web UI Settings

Create users and groups
Give users access to the appliance. You can also create messaging users and groups. Messaging users send and receive messages, but cannot log in to the appliance.
Appliance Users | Messaging Users and Groups

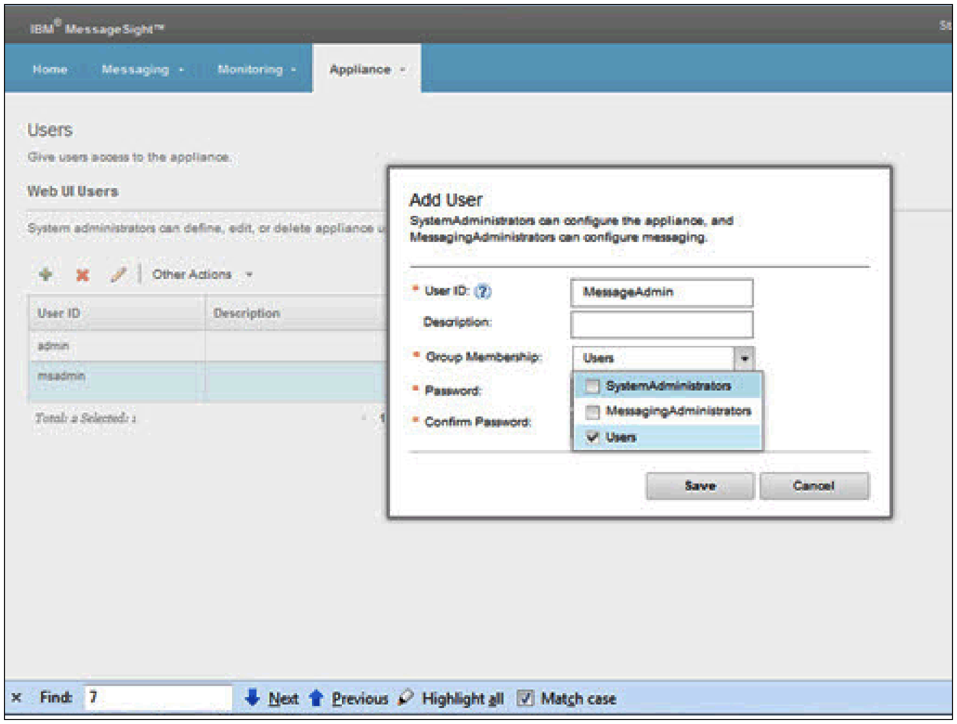
Configure IBM MessageSight to accept connections
Define a message hub to manage appliance connections. Configure MQ Connectivity to connect IBM MessageSight to MQ Queue Managers.

Figure 2-5 Use the web UI to configure users and groups for the IBM MessageSight appliance

Complete these steps to configure IBM MessageSight appliance users:

1. From the home page, under Create users and groups, select **Appliance Users**.
2. The default `admin` ID already exists, and you can edit that if you want by highlighting and clicking the pencil (edit) icon.

3. To create a new ID, select the plus sign (**+**). When the Add User pop-up box appears, you provide the User ID and Password and select the type of user, as shown in Figure 2-6.



Users

Give users access to the appliance.

Web UI Users

System administrators can define, edit, or delete appliance u

User ID	Description
admin	
msadmin	

Total: 2 Selected: 1

Add User

SystemAdministrators can configure the appliance, and MessagingAdministrators can configure messaging.

* User ID: MessageAdmin

Description:

* Group Membership: Users

- [ ] SystemAdministrators
- [ ] MessagingAdministrators
- [✓] Users

* Password:

* Confirm Password:

Find: 7

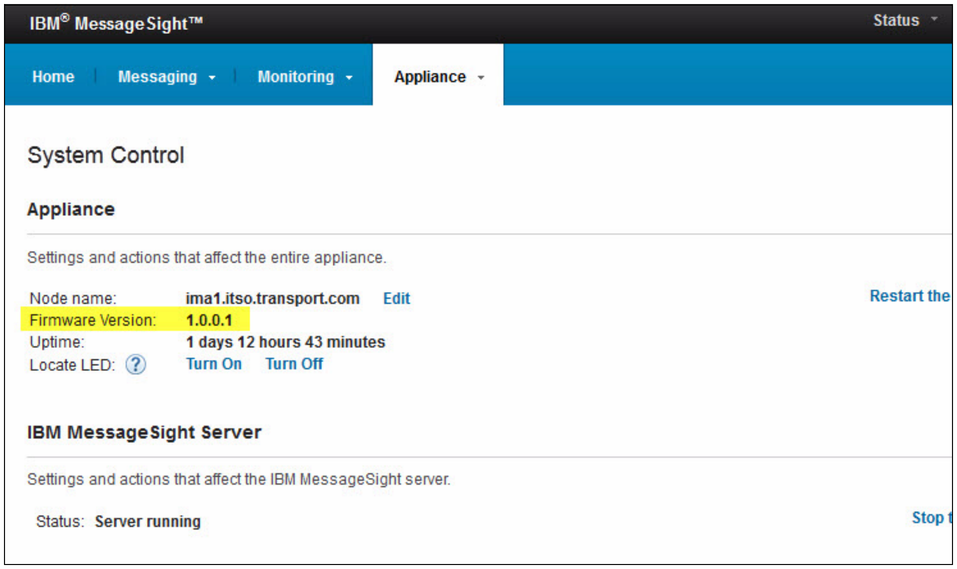
Figure 2-6 Adding a user with the web UI

2.4.3 Displaying the firmware version in the web UI

The IBM MessageSight firmware version is visible on the Home tab of the web UI. Log in to the IBM MessageSight web UI and scroll down to display the firmware version.

Restriction: If the firmware is not visible, you can log in to the web UI as a messaging administrator or appliance user, but not the administrator ID that has full system administrator permission.

Figure 2-7 shows the firmware version displayed in the web UI Home tab.



System Control

Appliance

Settings and actions that affect the entire appliance.

Node name: ima1.itso.transport.com
Firmware Version: 1.0.0.1
Uptime: 1 days 12 hours 43 minutes
Locate LED:

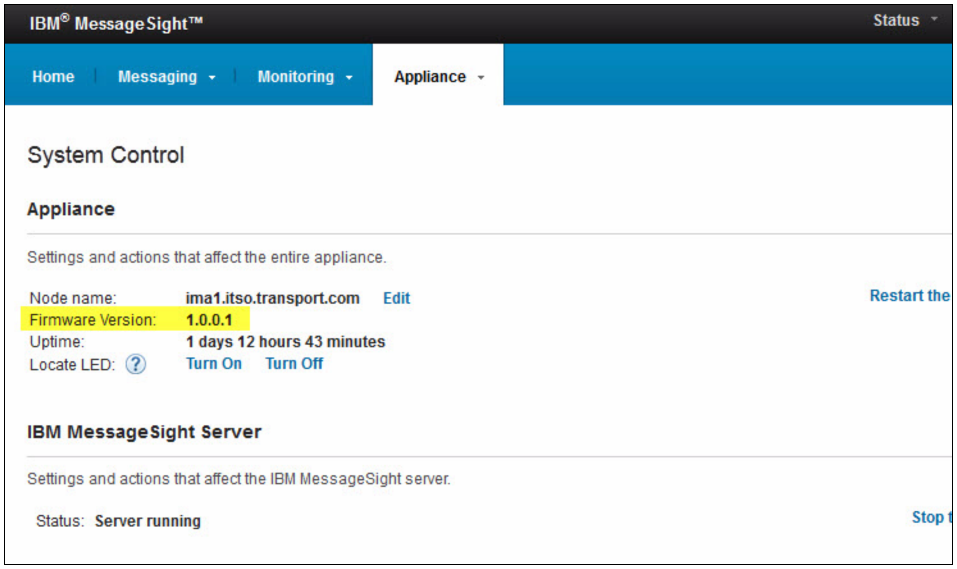
IBM MessageSight Server

Settings and actions that affect the IBM MessageSight server.

Status: Server running

Figure 2-7 Firmware Version displayed in the web UI Home Tab

The firmware version is also displayed when you select the **Appliance** tab and click the **System Control** option (Figure 2-8).



System Control

Appliance

Settings and actions that affect the entire appliance.

Node name: ima1.itso.transport.com
Firmware Version: 1.0.0.1
Uptime: 1 days 12 hours 43 minutes
Locate LED:

IBM MessageSight Server

Settings and actions that affect the IBM MessageSight server.

Status: Server running

Figure 2-8 Firmware version displayed in the web UI Appliance tab

2.4.4 Configuring message hubs

A message hub is an organizational object that groups endpoints, connection policies, and messaging policies that are associated with a specific goal. For example, you can create a message hub per application to organize the endpoints and policies that each application uses.

You can configure messaging hubs either by using the IBM MessageSight appliance web UI, or by using the commands shown in Example 2-6 from the CLI.

Example 2-6 Commands to configure messaging hubs

```
imaserver create
imaserver update
imaserver show
imaserver list
imaserver delete
```

You create message hubs, and the message hub components, in the following order:

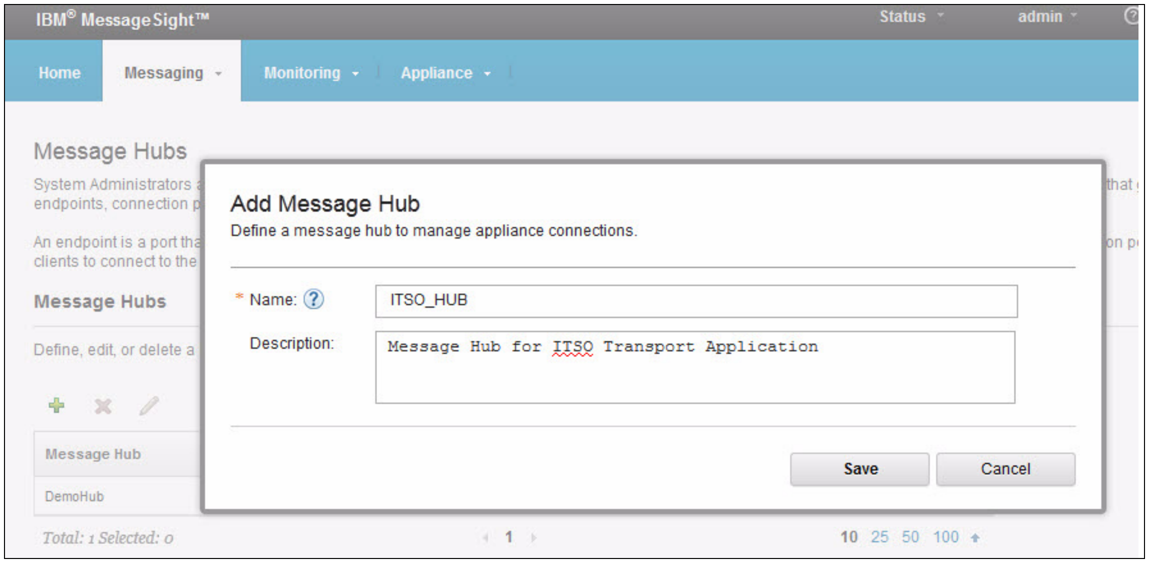
1. Message hubs
2. Connection policies
3. Messaging policies
4. Endpoints

When you create a message hub, the name you specify must not have leading or trailing spaces, and cannot contain control characters, commas, double quotation marks, back slashes, or equals signs. The first character must not be a number or any of the following special characters:

! # $ % & ' ( ) * + - . / : ; < > ? @

To create a message hub using the web UI, follow these steps:

1. Choose **Message hubs** from the drop-down box of the Messaging tab.
2. Click the green plus (**+**) sign to open a pop-up box where you enter the name and description, as shown in Figure 2-9.



Name: ITSO_HUB

Description: Message Hub for ITSO Transport Application

Figure 2-9 Example of Message Hub configuration

3. After the message hub is named and saved, you highlight the name of the message hub in the list and click the pencil icon to edit it. You must provide the Connection policy, messaging policy, and endpoint. Each message hub must have at least one endpoint.

Connection policies

A connection policy is used to authorize a client to connect to an endpoint. The connection policy can restrict which clients can connect to the endpoint. You must apply at least one connection policy to an endpoint so that a client can connect to that endpoint. When you create a connection policy, you can use the following filter attributes to restrict who is allowed to connect:

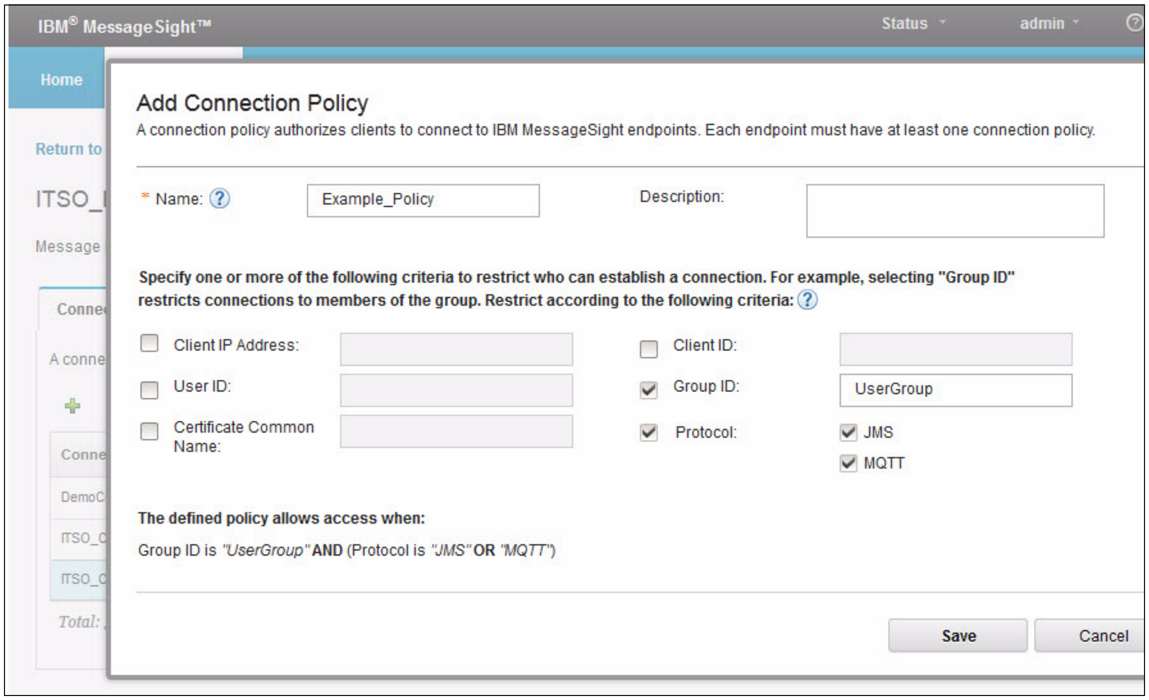
- Client IP address
- Client ID
- User ID
- Group Name
- Protocol
- Certificate common name

For example, an endpoint that is bound to an external-facing ethernet might be configured so that any IP address can connect. However, an endpoint that is bound to an internal-facing ethernet might be configured with a different connection policy, for which only certain IP addresses can connect.

A connection policy can be applied to more than one endpoint. For example, you can use a single connection policy to allow all clients from a particular IP address range to connect. You can then restrict the access of different clients to particular queues and topic strings by using a messaging policy.

To configure a connection policy using the web UI, you highlight the name of the message hub in the list and click the pencil icon to edit it. You then see tabs for Connection Policy, Messaging Policy, and Endpoint. On the Connection Policy tab, select the green plus (**+**) sign to create a new Connection Policy.

In the example shown in Figure 2-10, a sample connection policy is created that supports both JMS and MQTT connections. The IP address is not limited, but the Group ID parameter is populated to only allow IDs in the UserGroup to connect.



Add Connection Policy

A connection policy authorizes clients to connect to IBM MessageSight endpoints. Each endpoint must have at least one connection policy.

* Name: Example_Policy

Description:

Specify one or more of the following criteria to restrict who can establish a connection. For example, selecting "Group ID" restricts connections to members of the group. Restrict according to the following criteria:

☐ Client IP Address:		☐ Client ID:	
☐ User ID:		✓ Group ID:	UserGroup
☐ Certificate Common Name:		✓ Protocol:	✓ JMS ✓ MQTT

The defined policy allows access when:

Group ID is *"UserGroup"* **AND** (Protocol is *"JMS"* **OR** *"MQTT"*)

Figure 2-10 Example of a connection policy

Messaging policies

A messaging policy is used to control the topics or queue for which a client can send and receive messages. When you create a messaging policy, you must specify the following components:

- Name
- Destination Type (topics or queues)
- Destination
- Max Messages (only valid for topics)
- Authority

You must specify at least one of the following filters:

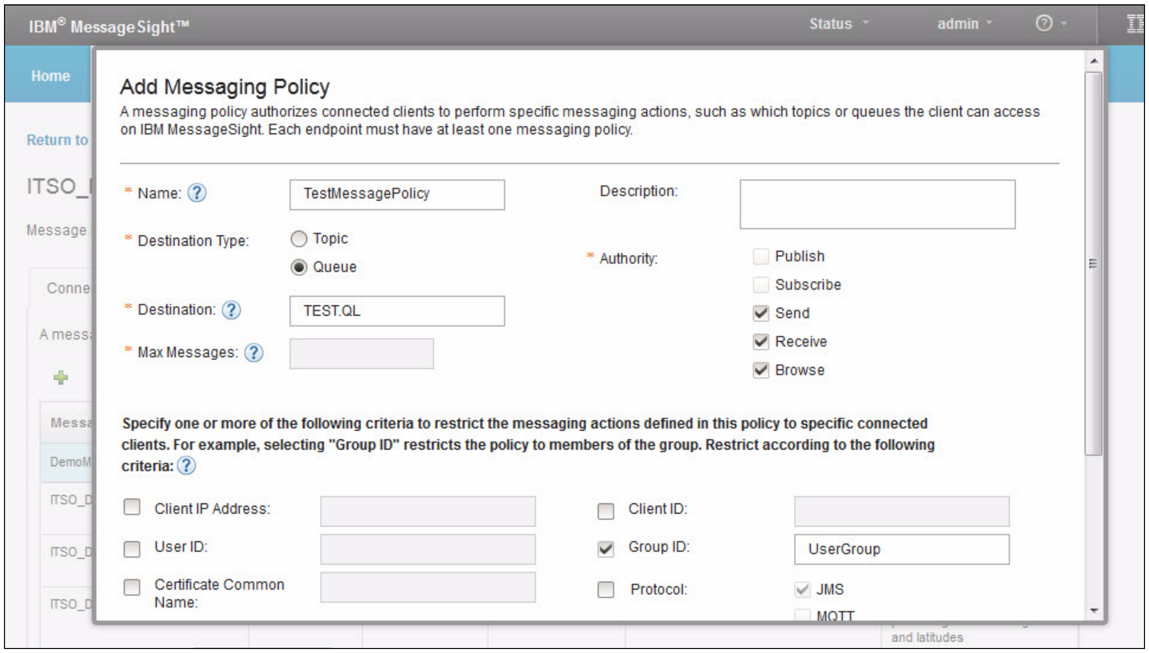
- Client IP address
- Client ID
- User ID
- Group Name
- Certificate Common Name
- Protocol

To configure a messaging policy using the web UI, you highlight the name of the message hub in the list and click the pencil icon to edit it. You then see tabs for Connection Policy, Messaging Policy, and Endpoint. On the Messaging Policy tab, select the green plus (**+**) sign to create a new messaging policy.

Tip: When specifying the Destination, you can use an asterisk (*) to specify *all* topic strings or queues. You can also use variable substitution in the topic string or queue to ensure that only specific user IDs or client IDs can access a topic. The variable for the user ID is *${UserID}*. The variable for the client ID is *${ClientID}*.

For example, if a topic string in a messaging policy is `ExampleTopic/TopicA/${ClientID}`, a client with an ID of client_a can access topic E`xampleTopic/TopicA/client_a`. A client with an ID of client_b cannot access topic `ExampleTopic/TopicA/client_a`, but can access `ExampleTopic/TopicA/client_b`.

In the example shown in Figure 2-11 on page 37, a sample Messaging Policy is created to allow applications with an ID in the group UserGroup to browse, put, or get messages from the `TEST.QL` queue.



Add Messaging Policy

A messaging policy authorizes connected clients to perform specific messaging actions, such as which topics or queues the client can access on IBM MessageSight. Each endpoint must have at least one messaging policy.

* Name: TestMessagePolicy

Description:

* Destination Type: Topic [✓ Queue]

* Authority: Publish, Subscribe, [✓ Send], [✓ Receive], [✓ Browse]

* Destination: TEST.QL

* Max Messages:

Specify one or more of the following criteria to restrict the messaging actions defined in this policy to specific connected clients. For example, selecting "Group ID" restricts the policy to members of the group. Restrict according to the following criteria:

Client IP Address:

Client ID:

User ID:

[✓ Group ID:] UserGroup

Certificate Common Name:

Protocol: [✓ JMS] MQTT

Figure 2-11 Example of a Messaging Policy

Endpoints

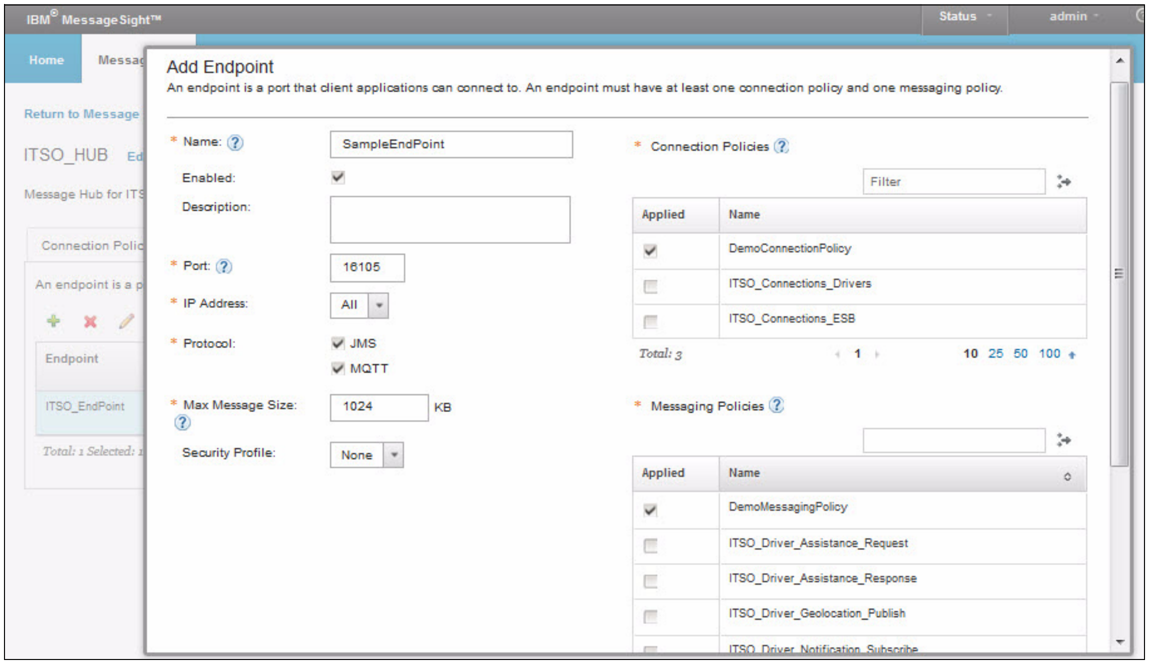
An endpoint enables a client to connect to the IBM MessageSight appliance. Each endpoint must have at least one connection policy, and at least one messaging policy.

When you create an endpoint, you can specify the following attributes:

- Name
- Enabled
- IP address
- Port
- Protocol
- Max Message Size
- Security Profile
- Connection Policies
- Messaging Policies

Important: When you configure an endpoint by using the CLI, you must also specify the Message Hub and Security components.

To configure an endpoint using the web UI, you highlight the name of the message hub in the list and click the pencil icon to edit it. You then see tabs for Connection Policy, Messaging Policy, and Endpoint. On the Endpoint tab, select the green (**+**) sign to create a new Endpoint, as shown in Figure 2-12.



Add Endpoint

An endpoint is a port that client applications can connect to. An endpoint must have at least one connection policy and one messaging policy.

* Name:	SampleEndPoint
Enabled:	✓
Description:	
* Port:	16105
* IP Address:	All
* Protocol:	✓ JMS ✓ MQTT
* Max Message Size:	1024 KB
Security Profile:	None

* Connection Policies

Filter

Applied	Name
✓	DemoConnectionPolicy
☐	ITSO_Connections_Drivers
☐	ITSO_Connections_ESB

Total: 3 1 10 25 50 100

* Messaging Policies

Applied	Name
✓	DemoMessagingPolicy
☐	ITSO_Driver_Assistance_Request
☐	ITSO_Driver_Assistance_Response
☐	ITSO_Driver_Geolocation_Publish
☐	ITSO_Driver_Notification_Subscribe

Figure 2-12 Example of Endpoint configuration by the web UI

Restriction: If an endpoint is configured with an IP address that does not match any of the IP addresses in the configured Ethernet-interfaces, the web UI adds that IP address to the list and selects it. That IP address will *not* show up in the list for any other endpoint, because it is not associated with an Ethernet-interface.

3

Integration with enterprise systems

This chapter contains information about how the IBM MessageSight appliance can be integrated with other IBM Messaging products, such as IBM WebSphere MQ and the IBM Integration Bus (formerly known as WebSphere Message Broker) to provide an important front edge feature for your existing messaging infrastructure.

For the purposes of this book, we assume a general working knowledge of WebSphere MQ, WebSphere Message Broker, and IBM Integration Bus. The book does not provide details about the installation and configuration of those products. It provides information about how to implement IBM MessageSight with those products to achieve the following connectivity options:

- WebSphere MQ connectivity and destination mapping
- Java Message Service (JMS) integration from WebSphere Message Broker to IBM MessageSight using WebSphere MQ as a JMS provider
- IBM MessageSight connectivity to IBM Integration Bus using JMS nodes that use the IBM MessageSight implementation of JMS
- Back-end enterprise applications direct connection to the IBM MessageSight appliance using the C or Java MQTT application programming interfaces (APIs), or the IBM MessageSight implementation of JMS

3.1 WebSphere MQ connectivity and destination mapping

You can configure the IBM MessageSight appliance to link with new or existing WebSphere queue managers with the MQ Connectivity feature. You can use the MQ Connectivity feature with WebSphere MQ Version 7.1 onwards.

The MQ Connectivity feature of the IBM MessageSight provides the bidirectional exchange of messages between the IBM MessageSight appliance and one or more WebSphere MQ queue managers. The MQ Connectivity feature works with messages that are both point-to-point (queue-based) and publish/subscribe.

The connection from the IBM MessageSight appliance to the queue manager uses a WebSphere MQ client channel that can be secured using Secure Sockets Layer (SSL) or Transport Layer Security (TLS). A single user ID is defined on the queue manager for IBM MessageSight authentication.

Important: When connecting to WebSphere MQ queue managers, the MQ Connectivity feature automatically creates queues on the queue manager. These queues have names that start with `SYSTEM.IMA`. Do not edit or delete these queues, because the MQ Connectivity feature requires them to function.

For the purposes of this book, we assume that you have already installed and created a WebSphere MQ queue manager. You learn about the steps necessary to connect the IBM MessageSight appliance to the queue manager, enable connectivity, and achieve integration that enables messages to flow between topics and queues from one product to the other.

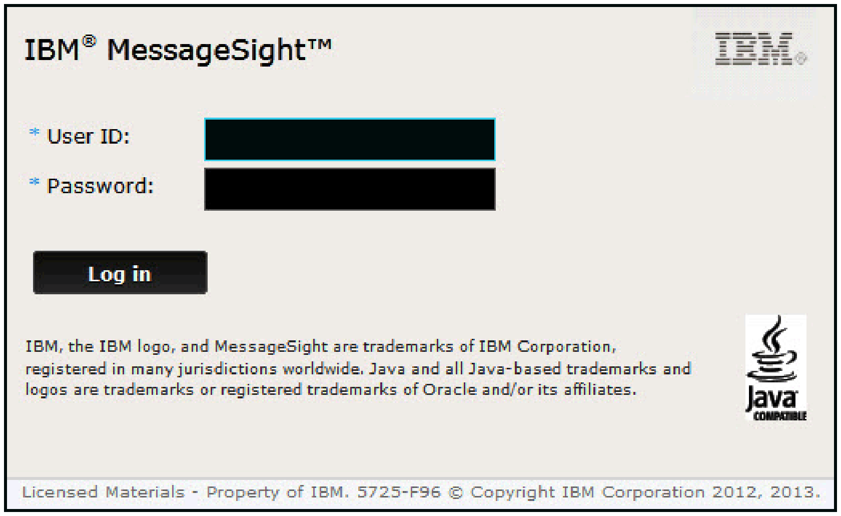
To connect IBM MessageSight to WebSphere MQ, you create a SVRCONN channel between the IBM MessageSight appliance and the queue manager, and then you configure destination mapping rules. To define a SVRCONN channel on the queue manager, run the **runmqsc** command on the server where the queue manager is running, as shown in Example 3-1.

Example 3-1 Creating a connection

```
$ runmqsc MQTT_QMGR
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager MQTT_QMGR.

define chl(IMA.SVRCONN) chltype(SVRCONN)
     1 : define chl(IMA.SVRCONN) chltype(SVRCONN)
AMQ8014: WebSphere MQ channel created.
```

After the channel is created on the queue manager, complete the following steps to configure the IBM MessageSight MQ Connectivity feature:

1. Log in to the IBM MessageSight appliance with a supported browser using an HTTPS IP address and port number. Sign in as an administrator (Figure 3-1).



IBM® MessageSight™

* User ID: [redacted]

* Password: [redacted]

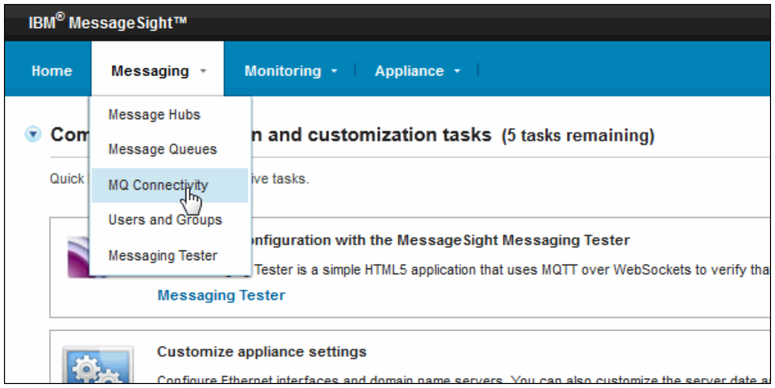
Log in

IBM, the IBM logo, and MessageSight are trademarks of IBM Corporation, registered in many jurisdictions worldwide. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Licensed Materials - Property of IBM. 5725-F96 © Copyright IBM Corporation 2012, 2013.

Figure 3-1 IBM MessageSight login window

2. After you are signed in, click **MQ Connectivity** at the bottom of the Home tab, or you can choose **Messaging** → **MQ Connectivity**, as shown in Figure 3-2.



IBM® MessageSight™

Home | Messaging | Monitoring | Appliance

Message Hubs
Message Queues
MQ Connectivity
Users and Groups
Messaging Tester

Com n and customization tasks (5 tasks remaining)

Quick ive tasks.

nfiguration with the MessageSight Messaging Tester

Tester is a simple HTML5 application that uses MQTT over WebSockets to verify tha

Messaging Tester

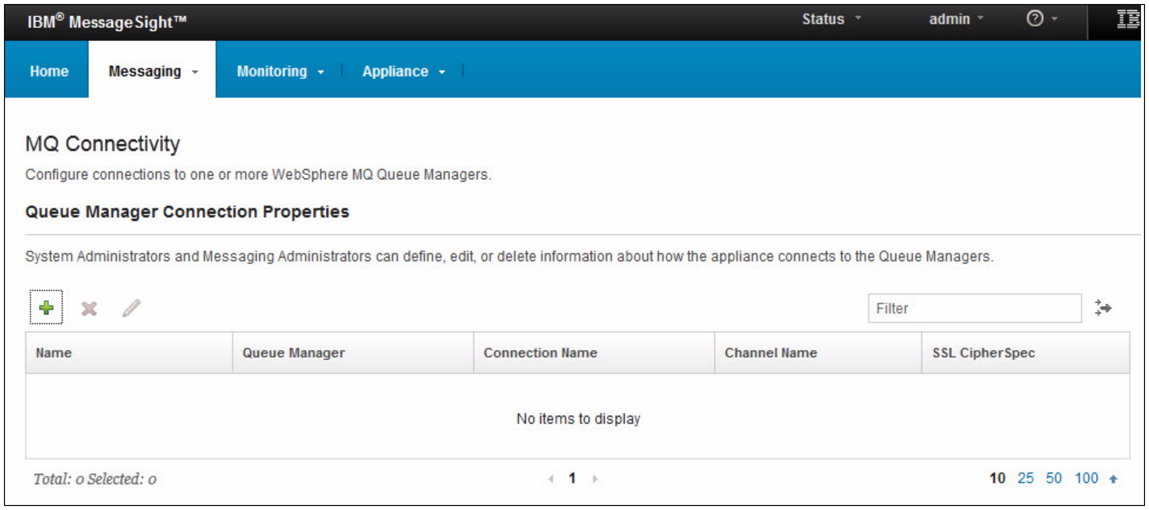
Customize appliance settings

Configure Ethernet interfaces and domain name servers. You can also customize the server date a

Figure 3-2 Select MQ Connectivity from the Messaging drop-down list

3. The Queue Manager Connection Properties (Figure 3-3) list is displayed. This is where you specify the connection details for a WebSphere MQ queue manager. Queue manager connections are used in destination mapping rules to specify the location of the WebSphere MQ topic or queue.

   Click the green plus (**+**) sign to add a queue manager connection.



MQ Connectivity

Configure connections to one or more WebSphere MQ Queue Managers.

Queue Manager Connection Properties

System Administrators and Messaging Administrators can define, edit, or delete information about how the appliance connects to the Queue Managers.

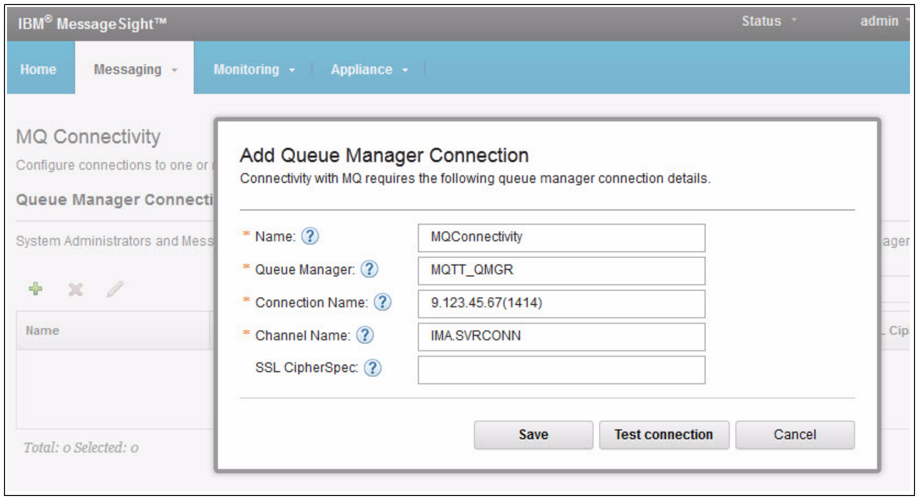
Name	Queue Manager	Connection Name	Channel Name	SSL CipherSpec
No items to display				

Total: 0 Selected: 0

Figure 3-3 WebSphere Queue Manager Connection Properties list

4. In the Add Queue Manager Connection window, complete the Name of the connection, the Queue Manager name, the Connection Name, and the SVRCONN Channel Name.

   You must supply the name of the SVRCONN channel that was defined on the queue manager to use for your IBM MessageSight connection. In this example, we use the `IMA.SVRCONN` channel, as shown in Figure 3-4.



Add Queue Manager Connection

Connectivity with MQ requires the following queue manager connection details.

* Name:	MQConnectivity
* Queue Manager:	MQTT_QMGR
* Connection Name:	9.123.45.67(1414)
* Channel Name:	IMA.SVRCONN
SSL CipherSpec:	

Figure 3-4 Adding queue manager connection information

5. Note that the ID you are using on IBM MessageSight must have proper authority to access the WebSphere MQ objects, or you see errors in the queue manager logs, as shown in Example 3-2.

Example 3-2 Invalid ID error message

```
AMQ5653: The user 'admin' is not defined.
EXPLANATION: The system call getpwnam("admin") failed with errno -1.
ACTION: Create the user 'admin' and retry the operation.
```

   To prevent these types of errors, be sure you have granted WebSphere MQ authorities, as shown in Example 3-3.

Example 3-3 WebSphere MQ authorities

```
setmqaut -m MQTT_QMGR -t qmgr  -g msadmin +connect +dsp +inq
setmqaut -m MQTT_QMGR -t queue -n SYSTEM.DEFAULT.MODEL.QUEUE -g
msadmin +dsp +get
setmqaut -m MQTT_QMGR -t queue -n SYSTEM.ADMIN.COMMAND.QUEUE -g
msadmin +dsp +put
```

```
setmqaut -m MQTT_QMGR -t queue -n SYSTEM.IMA.* -g msadmin +crt +put
+get +browse
setmqaut -m MQTT_QMGR -t queue -n SYSTEM.DEFAULT.LOCAL.QUEUE  -g
msadmin +dsp
```

For more information about how to manage WebSphere MQ security and authorities, see the WebSphere MQ Information Center:

`http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp`

6. Next, create destination mapping rules. While still on the MQ Connectivity tab of the web user interface (web UI), click the green plus (**+**) sign under The Destination Mapping Rules. The Add Destination Mapping Rule displays, and you can enter the rule information.

   The drop-down list for Rule Type (Figure 3-5) enables you to select how you want to map between topics and queues.

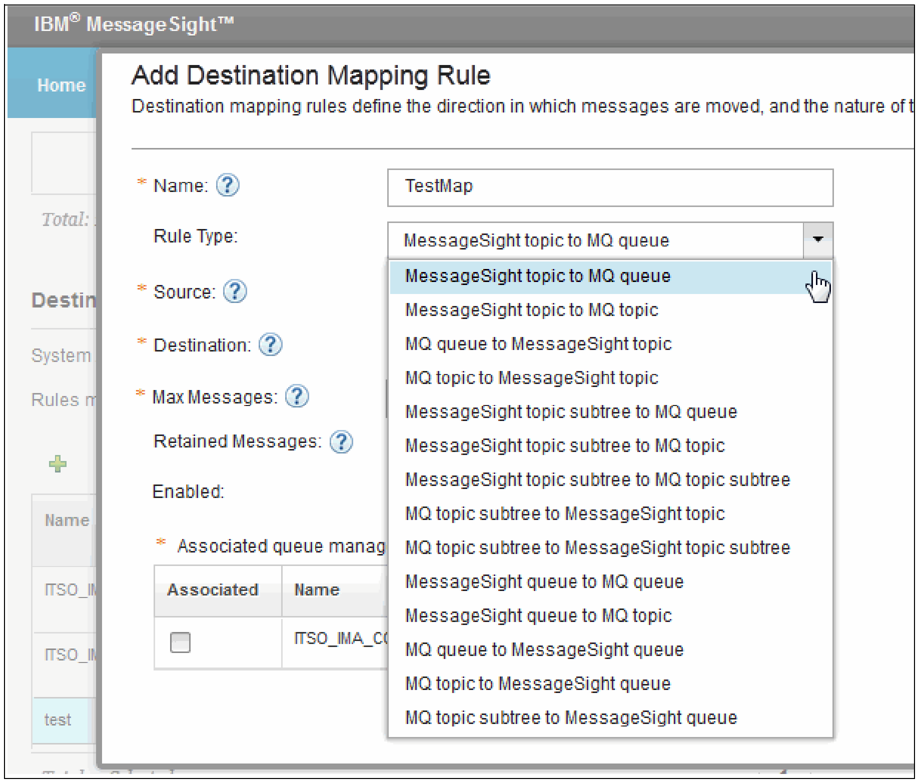
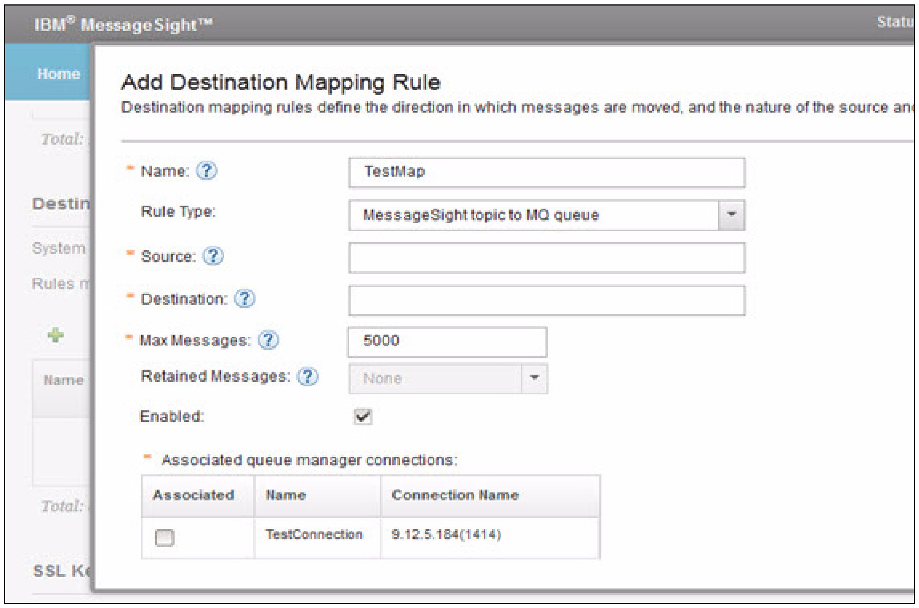

IBM® MessageSight™
Home
Total:
Destin
System
Rules m
Name
ITSO_I
ITSO_I
test
Add Destination Mapping Rule
Destination mapping rules define the direction in which messages are moved, and the nature of t
* Name:
TestMap
Rule Type:
MessageSight topic to MQ queue
* Source:
* Destination:
* Max Messages:
Retained Messages:
Enabled:
* Associated queue manag
Associated
Name
ITSO_IMA_C
MessageSight topic to MQ queue
MessageSight topic to MQ topic
MQ queue to MessageSight topic
MQ topic to MessageSight topic
MessageSight topic subtree to MQ queue
MessageSight topic subtree to MQ topic
MessageSight topic subtree to MQ topic subtree
MQ topic subtree to MessageSight topic
MQ topic subtree to MessageSight topic subtree
MessageSight queue to MQ queue
MessageSight queue to MQ topic
MQ queue to MessageSight queue
MQ topic to MessageSight queue
MQ topic subtree to MessageSight queue

Figure 3-5 Selecting a rule type for mapping objects

7. Figure 3-6 shows adding a rule to map an IBM MessageSight topic to a WebSphere MQ queue.



Add Destination Mapping Rule

Destination mapping rules define the direction in which messages are moved, and the nature of the source an

* Name:	TestMap
Rule Type:	MessageSight topic to MQ queue
* Source:	
* Destination:	
* Max Messages:	5000
Retained Messages:	None
Enabled:	✓

* Associated queue manager connections:

Associated	Name	Connection Name
☐	TestConnection	9.12.5.184(1414)

Figure 3-6 Adding destination mapping rules

You must also select the **Associated** check box under **Associated queue manager connections** to associate the Destination Mapping rule with the appropriate queue manager connection.

After a rule is created and enabled (select the **Enabled** check box), messages are routed as configured. Be aware that a destination mapping rule cannot be edited or deleted when it is enabled. If you want to edit or delete a rule, you must first disable the rule using the Other Actions menu located near the plus (+), x, and pencil icons under the Destination Mapping Rules heading.

The IBM MessageSight appliance can handle a higher volume of messages than WebSphere MQ. To deal with the higher volume of messages, you can maximize message throughput by using multiple WebSphere MQ queue managers to handle the load.

However, using multiple queue managers does not preserve the order of the messages. If your application design requires that you preserve the order of the messages, you must use only a single queue manager connection to send or receive messages.

If WebSphere MQ cannot keep up with the volume of messages that the IBM MessageSight appliance forwards, a backlog of messages forms on the IBM MessageSight appliance. For a destination mapping rule from an IBM MessageSight topic or topic subtree, the maximum messages parameter applies.

You define the number of messages that can be in this backlog in the **Max Message** field in the Add Destination Mapping Rule window.

The release of IBM MessageSight V1.1 includes new features, such as a resource adapter and shared subscriptions, which can help spread the workload and prevent message backlog from occurring.

Shared subscriptions allow multiple consumers to help process published messages, and is useful in situations where a single consumer cannot keep up with the rate of messages being published.

The combination of the resource adapter and shared subscriptions available in IBM MessageSight V1.1 is really powerful, because it enables a stream of messages to be distributed among a set of application servers, such as a WebSphere Application Server cluster.

3.2 JMS connectivity with IBM Integration Bus

Integration between remote clients and enterprise applications can be facilitated using IBM MessageSight and IBM Integration Bus (or the earlier WebSphere Message Broker). The JMS nodes in IBM Integration Bus V9 (and WebSphere Message Broker V8) can be configured to use the IBM MessageSight JMS classes. This enables IBM Integration Bus (and WebSphere Message Broker) message flows to interact directly with the appliance.

IBM Integration Bus also provides patterns that can be used to develop interactions between mobile devices and enterprise applications. Enterprise applications can connect directly to the IBM MessageSight appliance through the use of any of the following protocol APIs:

- JMS (using the IBM MessageSight JMS classes)
- MQTT (Java API)
- MQTT (C API)

An enterprise application can also use any of the WebSphere MQ APIs to communicate with the appliance through a queue manager. This requires appropriate configuration of the MQ Connectivity facility on the appliance.

3.2.1 JMS integration with WebSphere MQ as the JMS provider

This section describes a mechanism to integrate IBM MessageSight with WebSphere MQ as the JMS provider. This approach can also be used in an WebSphere Message Broker Version 8 environment. With IBM Integration Bus Version 9, IBM MessageSight is also available as a JMS provider.

This type of integration is most suitable in certain installations that are configured to communicate with IBM WebSphere MQ only. For example, existing systems might communicate with IBM WebSphere MQ over traditional WebSphere MQ API calls only. In such cases, the integration can only be through message queues between IBM MessageSight and IBM WebSphere Message Broker.

You can also implement an integration between IBM MessageSight and WebSphere MQ using message queues rather than JMS. To do this, a destination mapping rule exists on IBM MessageSight to map topics to queues, or topic subtrees to queues, and vice versa.

Figure 3-7 shows the integration of IBM MessageSight and WebSphere Message Broker V8.

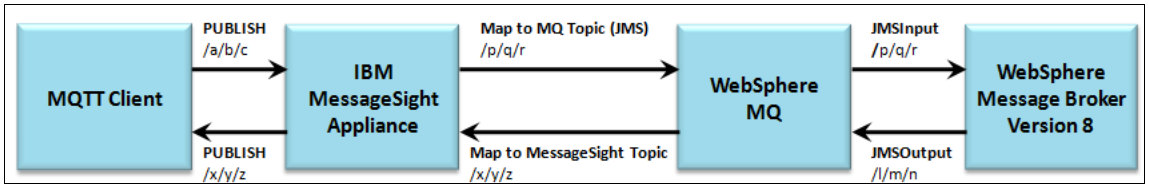

MQTT Client
PUBLISH /a/b/c
PUBLISH /x/y/z
IBM MessageSight Appliance
Map to MQ Topic (JMS) /p/q/r
Map to MessageSight Topic /x/y/z
WebSphere MQ
JMSInput /p/q/r
JMSOutput /l/m/n
WebSphere Message Broker Version 8

Figure 3-7 End-to-end integration of IBM MessageSight and WebSphere Message Broker V8

An MQTT client (for example, a digital temperature sensor) connects to the IBM MessageSight appliance to publish messages `/a/b/c`. You must use a destination mapping rule in WebSphere MQ Connectivity to map the topic string `/a/b/c` published on the IBM MessageSight appliance to WebSphere MQ topic string `/p/q/r`. You can develop a message flow to read this topic string using the `JMSInput` node.

After the message flow reads the message on the topic, it transforms the message and forwards it to a `JMSOutput` node for a different topic on WebSphere MQ. Another destination mapping rule on IBM MessageSight maps this topic to a topic on IBM MessageSight. An MQTT client (for example, a hand-held device monitoring the temperature) that subscribes to this topic receives the publication as a response.

For more information about setting destination mapping rules, see the *Configuring destination mapping rules* topic in the IBM MessageSight Information Center:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00110_.html

Integration procedure

The following section describes a high-level procedure to implement an integration between IBM MessageSight and WebSphere Message Broker Version 8 using JMS:

1. Create an IBM MessageSight topic to WebSphere MQ topic destination mapping rule in IBM MessageSight. This rule takes publications from the MQTT client to IBM MessageSight and forwards them to IBM WebSphere MQ.
2. Create a WebSphere MQ topic to IBM MessageSight topic destination mapping rule in IBM MessageSight. This rule takes publications from the message broker (on IBM MessageSight) and forwards them to IBM MessageSight.
3. Create a bindings file for the JMS nodes in WebSphere Message Broker message flow.

Destination mapping rule from IBM MessageSight to WebSphere MQ

You can use the IBM MessageSight web UI or command-line interface (CLI) to create a destination mapping rule between IBM MessageSight and WebSphere MQ. The following list includes elements of a destination mapping rule. As an example, consider a mapping between an IBM MessageSight topic (`/root/publication/temperature`) and a WebSphere MQ queue (`QLOCAL.TEMPERATURE`):

- Destination mapping rule name

  This is the name of the destination mapping rule that you want to create.

- Destination mapping rule type

  The destination mapping rule type includes topic strings, topic subtrees, and queues.

- Source

  Depending on the destination mapping rule type, the source can be set to the name of a queue, a topic string, or a topic subtree. In this example, the source is set to `/root/publication/temperature`.

- Destination

  Depending on the destination mapping rule type, the destination can be set to the name of a queue, a topic string, or a topic subtree. In this example, the destination is set to `QLOCAL.TEMPERATURE`.

- Maximum messages

  This is the maximum number of messages that can be buffered for a given destination mapping rule. This limit applies to destination mapping rules whose source is an IBM MessageSight topic or topic subtree. When this limit is exceeded, no more publications can happen to the topic on IBM MessageSight.

- Queue manager connection

  This is the name of the queue manager connection that was created when you set up connectivity with WebSphere MQ queue manager. At least one such connection name must be provided.

  If multiple connections are associated in the destination mapping rule, two consecutive messages can be sent to two different queue managers. However, in this case, the order of arrival of the messages is not guaranteed.

  Tip: If better throughput is a requirement *and* message order is not strict, a destination mapping rule can be associated with multiple queue manager connections.

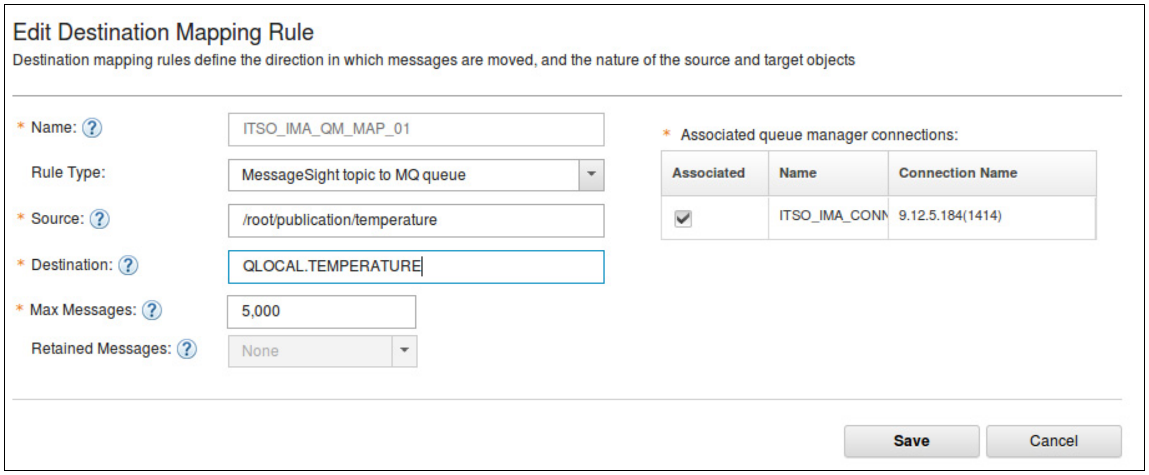
To set up this rule, you can either use the web UI or the CLI.

Creating a destination mapping rule using the web UI

To set up the destination mapping rule from the web UI, follow these steps:

1. Go to **Messaging** → **MQ Connectivity**.
2. Click the green plus (**+**) icon to add a new rule.

3. In the Edit Destination Mapping Rule window, enter details, as shown in Figure 3-8.


Edit Destination Mapping Rule

Destination mapping rules define the direction in which messages are moved, and the nature of the source and target objects

* Name:	ITSO_IMA_QM_MAP_01
Rule Type:	MessageSight topic to MQ queue
* Source:	/root/publication/temperature
* Destination:	QLOCAL.TEMPERATURE
* Max Messages:	5,000
Retained Messages:	None

* Associated queue manager connections:

Associated	Name	Connection Name
✓	ITSO_IMA_CONN	9.12.5.184(1414)

Figure 3-8 Destination mapping rule between IBM MessageSight and WebSphere MQ topic subtrees

4. The Rule Type field determines the type of mapping that you want to perform. In this example, we want the publications on IBM MessageSight to map to WebSphere MQ topic strings. Therefore, select **MessageSight topic to MQ queue** as the rule type.

Creating a destination mapping rule using CLI options

To set up a destination mapping rule using the command-line option, follow these steps:

1. Log in to the appliance using the **`ssh`** command.
2. Issue the **`imaserver create`** command, as shown in Example 3-4.

Example 3-4 Creating a destination mapping rule with the CLI

```
ssh admin@ima1.itso.transport.com
Welcome to IBM MessageSight

5725-F96

Copyright 2012, 2013 IBM Corp. Licensed Materials - Property of IBM.

IBM and MessageSight are trademarks or registered trademarks of IBM,
registered in many jurisdictions worldwide.  Other product and
service names might be trademarks of IBM or other companies.
```

```
Console> imaserver create DestinationMappingRule
"Name=ITSO_IMA_QM_MAP_01" "QueueManagerConnection=ITSO_IMA_CONN"
"RuleType=1" "Source=/root/publication/temperature"
"Destination=QLOCAL.TEMPERATURE"
The requested configuration change has completed successfully.
```

3. The value of `1` for RuleType indicates that this is a mapping between the topic string on IBM MessageSight and the message queue on WebSphere MQ. Note that the command must be entered in a single line.

   For a complete description with examples of command-line options for WebSphere MQ connectivity, see the *MQ Connectivity commands* topic on the IBM MessageSight Information Center:

   https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Reference/MQConnCmd/mqconnectivitycommands.html

   For a complete description with examples of mapping, see the *Configuring destination mapping rules* topic on the IBM MessageSight Information Center:

   https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00110_.html#ad00110___examples

Destination mapping rule from WebSphere MQ to IBM MessageSight

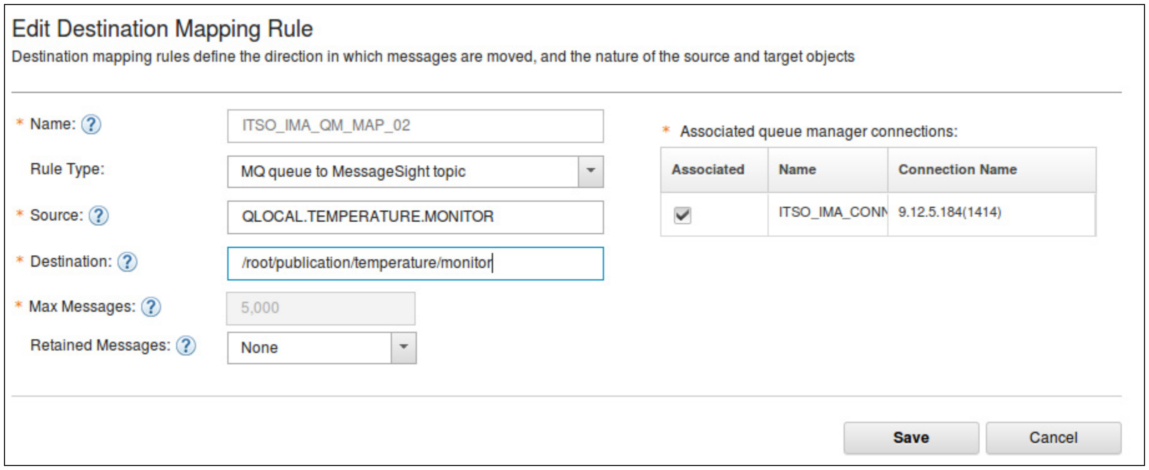
You can use the IBM MessageSight web UI or CLI to create a destination mapping rule between WebSphere MQ and IBM MessageSight.

Creating destination mapping rule with web UI

To set up the destination mapping rule between WebSphere MQ and IBM MessageSight, follow these steps:

1. From the web UI, go to **Messaging** → **MQ Connectivity**.
2. Click the plus (**+**) icon to add a new rule.

3. In the Edit Destination Mapping Rule window, enter details, as shown in Figure 3-9.


Edit Destination Mapping Rule

Destination mapping rules define the direction in which messages are moved, and the nature of the source and target objects

* Name:	ITSO_IMA_QM_MAP_02
Rule Type:	MQ queue to MessageSight topic
* Source:	QLOCAL.TEMPERATURE.MONITOR
* Destination:	/root/publication/temperature/monitor
* Max Messages:	5,000
Retained Messages:	None

* Associated queue manager connections:

Associated	Name	Connection Name
✓	ITSO_IMA_CONN	9.12.5.184(1414)

Figure 3-9   Destination mapping rule between WebSphere MQ and IBM MessageSight topic subtrees

Create a destination mapping rule using command-line options

The same rule can be set using the command-line option:

1. To start the CLI, log in to the appliance using the **ssh** command, as shown in Example 3-5.
2. Issue the **imaserver create** command.

Example 3-5   Create destination mapping rule, WebSphere MQ to IBM MessageSight

```
ssh admin@ima1.itso.transport.com
Welcome to IBM MessageSight

5725-F96

Copyright 2012, 2013 IBM Corp. Licensed Materials - Property of IBM.

IBM and MessageSight are trademarks or registered trademarks of IBM,
registered in many jurisdictions worldwide.  Other product and
service names might be trademarks of IBM or other companies.

Console> imaserver create DestinationMappingRule
"Name=ITSO_IMA_QM_MAP_02" "QueueManagerConnection=ITSO_IMA_CONN"
"RuleType=3" "Source=QLOCAL.TEMPERATURE.MONITOR"
"Destination=/root/publication/temperatureMonitor"

The requested configuration change has completed successfully.
```

3. The value of 3 for RuleType indicates that this is a mapping between a message queue on WebSphere MQ and an IBM MessageSight topic. Note that the command must be entered in a single line.

   For a complete description with examples of command-line options for MQ Connectivity, see the *MQ Connectivity command* topic on the IBM MessageSight Information Center:

   https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Reference/MQConnCmd/mqconnectivitycommands.html

   For a complete description with examples of mapping, see the *Configuring destination mapping rules* topic on the IBM MessageSight Information Center:

   https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00110_.html#ad00110___examples

Listing and showing destination mapping rules

After creating the destination mapping rules, you can list them and show details about individual mappings, using command-line options, as shown in Example 3-6. The type of the rules is highlighted to denote the type of mapping.

Example 3-6 Command-line option to list and show the destination mapping rules

```
Console> imaserver list DestinationMappingRule
ITSO_IMA_QM_MAP_01
ITSO_IMA_QM_MAP_02
Console> imaserver show DestinationMappingRule "Name=ITSO_IMA_QM_MAP_01"
Name = ITSO_IMA_QM_MAP_01
QueueManagerConnection = ITSO_IMA_CONN
RuleType = 1
Source = /root/publication/temperature
Destination = QLOCAL.TEMPERATURE
MaxMessages = 5000
Enabled = True
RetainedMessages = None
Console> imaserver show DestinationMappingRule "Name=ITSO_IMA_QM_MAP_02"
Name = ITSO_IMA_QM_MAP_02
QueueManagerConnection = ITSO_IMA_CONN
RuleType = 3
Source = QLOCAL.TEMPERATURE.MONITOR
Destination = /root/publication/temperature/monitor
MaxMessages = 5000
Enabled = True
RetainedMessages = None
```

Bindings file for JMS nodes

To create the bindings file for JMS nodes, use WebSphere MQ Explorer. When creating the bindings file, it is important to have the values in the `JMSInput` and `JMSOutput` nodes correspond to values in the bindings file.

Testing the integration

To test the integration, complete the following steps:

1. Connect an MQTT device to the IBM MessageSight appliance as the publishing device.
2. Connect another MQTT device as the subscribing device.
3. Set up a subscription to your topic string (for example, `/root/publication/temperature/monitor`) from the subscriber MQTT device.
4. Publish a message from your publishing MQTT device to the topic string (for example, `/root/publication/temperature`).

The destination mapping rule forwards the publication from the publishing MQTT device to WebSphere MQ. This publication is used by the WebSphere Message Broker Version 8 message flow as a JMS message. The message flow publishes its response back to WebSphere MQ, where another destination mapping rule forwards it to IBM MessageSight. This message is finally used by the MQTT subscribing device.

3.2.2 JMS integration with IBM MessageSight as the JMS provider

This section explains how back-end applications can interface to the “mobile world” using a combination of IBM Integration Bus V9 and the IBM MessageSight appliance. The IBM Integration Bus JMS nodes are configured to use the IBM MessageSight JMS classes.

The key differences between this approach and the approach described in 3.2.1, “JMS integration with WebSphere MQ as the JMS provider” on page 47, are shown in the following list:

- The IBM Integration Bus component connects directly to the IBM MessageSight appliance. Transiting a WebSphere MQ queue manager is not required. This is relevant where higher message loads must be supported, or a lower latency per transaction is required.
- The JMS provider is different (one uses the WebSphere MQ implementation of JMS, the other uses the IBM MessageSight JMS implementation).

Overview of the integration

IBM Integration Bus uses the `JMSInput` and `JMSOutput` nodes to receive JMS messages, and to produce messages. The two nodes support both the queuing and the publish/subscribe paradigms.

For example, an IBM Integration Bus message flow can receive a JMS message intended for a queue destination, process the message (such as store in a database and transform), and then publish the transformed message to a specified topic destination. Other JMS clients can subscribe to this published topic.

Figure 3-10 shows a high-level overview of the mobile applications communicating with the IBM Integration Bus. The mobile applications use MQTT to publish requests and subscribe to responses. The IBM Integration Bus message flows are using JMS to subscribe to requests and to publish responses. IBM MessageSight does the required protocol bridging between MQTT and JMS.

In this scenario, the mobile applications publish on the request topics and subscribe to the related response topics. The JMS nodes in the IBM Integration Bus message flows subscribe to the request topics and publish on the response topics.

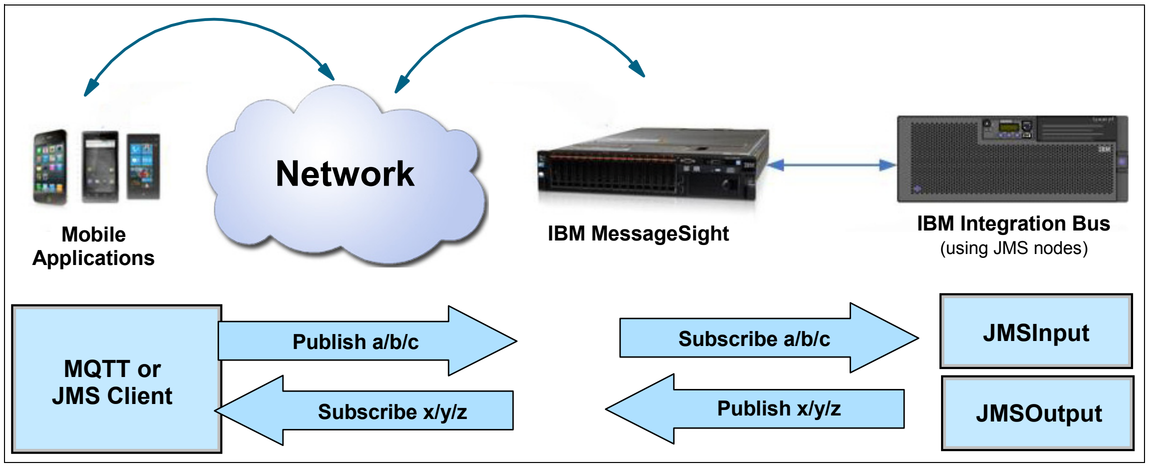


Figure 3-10 High-level overview of mobile applications, IBM MessageSight, and IBM Integration Bus

For more information about message delivery and the mapping between MQTT and JMS, see the IBM MessageSight Information Center:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Planning/pl00010.html

Integration details

The following steps are necessary to implement the integration between IBM MessageSight and the IBM Integration Bus:

1. A `.bindings` file is required by the IBM Integration Bus `JMSInput` and `JMSOutput` nodes. This bindings file must be created and saved in a directory to which the IBM Integration Bus message flows have access.
2. Configure the two IBM Integration Bus JMS nodes to use the bindings file (created in step 1), and point the JMS nodes at the IBM MessageSight JMS classes.

Creating a bindings file

No utilities are currently available to create this bindings file. There is no IBM MessageSight equivalent to the WebSphere MQ Explorer with the JMS Administered Objects feature. A sample of a bindings file is provided with the JMS SDK. You can edit this file with a suitable text editor.

For more information about using JMS applications with IBM MessageSight, see *Creating administered objects* in the IBM MessageSight Information Center:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Developing/devjms_creatingadministeredobjects.html

Important: The name of the bindings file must be `.bindings`. Note the leading period in the name.

Example 3-7 depicts an example of a `.bindings` file. In this scenario, the `.bindings` file is saved in the `C:\JNDI\JIMA-Sample` Windows directory. The contents of this example `.bindings` file configure the following JMS-administered objects:

- A JMS Connection Factory named `connFactory1`. The attributes for this connection factory include the IP address of the JMS Server (the IBM MessageSight appliance) and the associated port (IBM MessageSight endpoint).
- A JMS topic for the request topic named `testRequestTopic`.
- A JMS topic for the response topic named `testResponseTopic`.

Example 3-7 JNDI .bindings example

```
connFactory1/ClassName=com.ibm.ima.jms.impl.ImaConnectionFactory
connFactory1/FactoryName=com.ibm.ima.jms.impl.ImaConnectionFactory
connFactory1/RefAddr/0/Encoding=String
connFactory1/RefAddr/0/Type=Port
connFactory1/RefAddr/0/Content=16105
```

```
connFactory1/RefAddr/1/Encoding=String
connFactory1/RefAddr/1/Type=Server
connFactory1/RefAddr/1/Content=9.12.5.184
connFactory1/RefAddr/2/Encoding=String
connFactory1/RefAddr/1/Type=ObjectType
connFactory1/RefAddr/2/Content=common

RequestTopic/ClassName=com.ibm.ima.jms.impl.ImaTopic
RequestTopic/FactoryName=com.ibm.ima.jms.impl.ImaTopic
RequestTopic/RefAddr/0/Encoding=String
RequestTopic/RefAddr/0/Type=Name
RequestTopic/RefAddr/0/Content=testRequestTopic

ResponseTopic/ClassName=com.ibm.ima.jms.impl.ImaTopic
ResponseTopic/FactoryName=com.ibm.ima.jms.impl.ImaTopic
ResponseTopic/RefAddr/0/Encoding=String
ResponseTopic/RefAddr/0/Type=Name
ResponseTopic/RefAddr/0/Content=testResponseTopic
```

Configuration for the use of IBM MessageSight JMS classes

Ensure that the JMS nodes in the IBM Integration Bus are using the JMS classes implemented for IBM MessageSight. This is done through the use of the **mqsichangeproperties** administration command.

Example 3-8 shows an example of the **mqsichangeproperties** command to configure the JMS nodes to use IBM MessageSight JMS classes. The shell variable `$MQSI_INSTALL_PATH` (used in the `-v` option in the example) points at the base installation directory for IBM Integration Bus. For example, this can point at `/opt/ibm/mqsi` on a Linux system.

You can place these Java archive (JAR) files in any suitable directory. They do not have to be in the IBM Integration Bus installation directory hierarchy.

Example 3-8 The mqsichangeproperties command

```
mqsichangeproperties ITSOBK2
   -c JMSProviders
   -o IBM_MessageSight
   -n jarsURL
   -v $MQSI_INSTALL_PATH/ImaClient/jms/lib

Verify that the response from the mqsichangeproperties command is:
BIP8071I: Successful command completion.
```

Verify that the JMS configuration of the integration node is changed appropriately.

Example 3-9 shows an example of the **mqsireportproperties** command to report on the configuration of JMS providers.

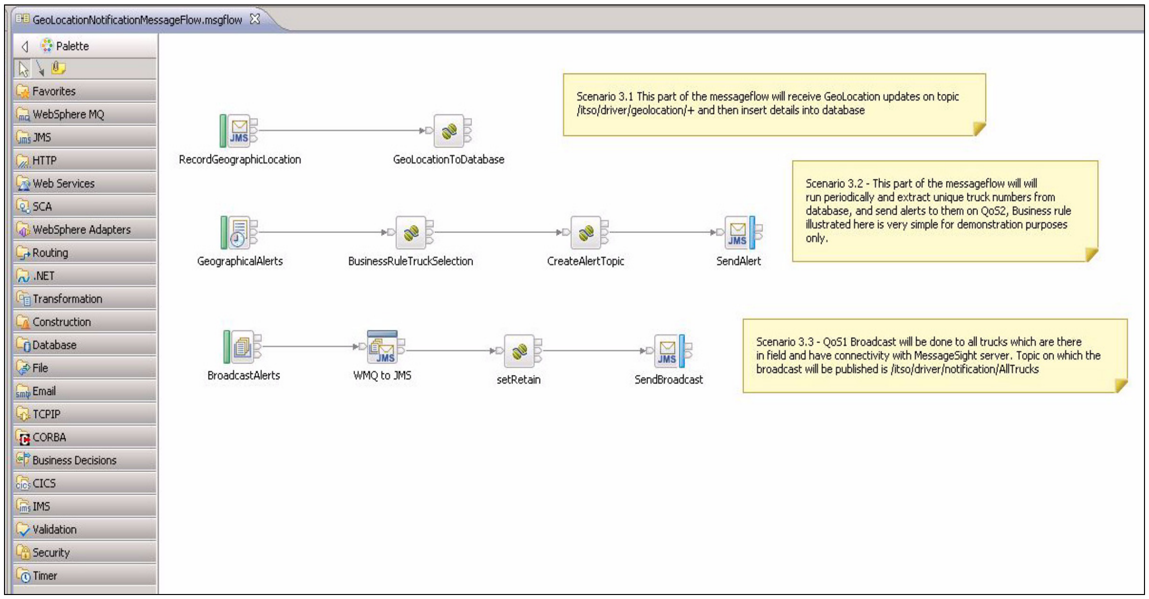
Example 3-9 mqsireportproperties command

```
mqsireportproperties ITSOBK2
   -c JMSProviders
   -o IBM_MessageSight
   -r

The response returned should look similar to the following response:
JMSProviders
   IBM_MessageSight
      clientAckBatchSize='0'
      clientAckBatchTime='0'
      connectionFactoryName=''
      initialContextFactory=''
      jarsURL='/opt/ibm/mqsi/ImaClient/jms/lib'
      jmsAsyncExceptionHandling='false'
      jmsProviderXASupport='true'
      jndiBindingsLocation=''
      jndiEnvironmentParms='default_none'
      nativeLibs='default_Path'
      proprietaryAPIAttr1='default_none'
      proprietaryAPIAttr2='default_none'
      proprietaryAPIAttr3='default_none'
      proprietaryAPIAttr4='default_none'
      proprietaryAPIhandler='default_none'

BIP8071I: Successful command completion
```

Configuring the IBM Integration Bus JMS nodes

This section explains the configuration of the `JMSInput` node and `JMSOutput` node used in the message flows.

Figure 3-11 on page 59 shows the overall IBM Integration Bus message flow, and where the JMS nodes are used in the flow.



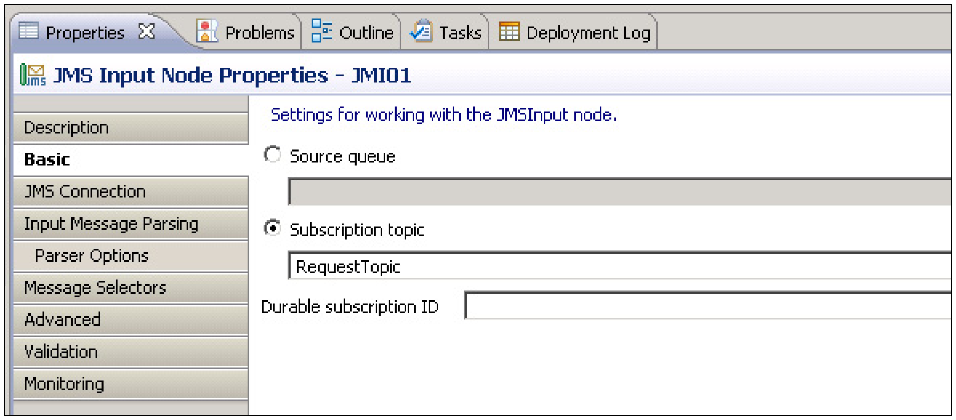
Scenario 3.1 This part of the messageflow will receive GeoLocation updates on topic /itso/driver/geolocation/+ and then insert details into database

Scenario 3.2 - This part of the messageflow will will run periodically and extract unique truck numbers from database, and send alerts to them on QoS2, Business rule illustrated here is very simple for demonstration purposes only.

Scenario 3.3 - QoS1 Broadcast will be done to all trucks which are there in field and have connectivity with MessageSight server. Topic on which the broadcast will be published is /itso/driver/notification/AllTrucks

Figure 3-11 JMS node usage in the IBM Integration Bus message flow

Figure 3-12 shows the Basic configuration of a `JMSInput` node. This input node is used in a publish/subscribe scenario. Therefore, the **Subscription topic** is selected. Note that the value, `RequestTopic`, is one of the administered objects created in the bindings file.



JMS Input Node Properties - JMI01

Settings for working with the JMSInput node.

○ Source queue

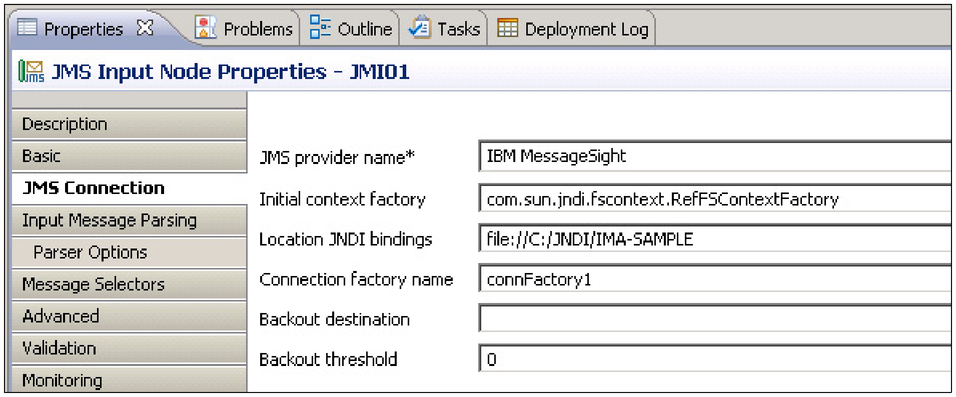
● Subscription topic: RequestTopic

Durable subscription ID:

Figure 3-12 JMSInput node Basic configuration

Figure 3-13 shows the configuration of the JMS Connection tab for the `JMSInput` node. The following list shows the parameters included in this tab:

- The name of the JMS provider, `IBM MessageSight`
- The fact that the node is using a file system context
- The location of the `.bindings` file
- The name of the JMS Connection Factory administered object, `connFactory1`, that was created earlier in the `.bindings` file

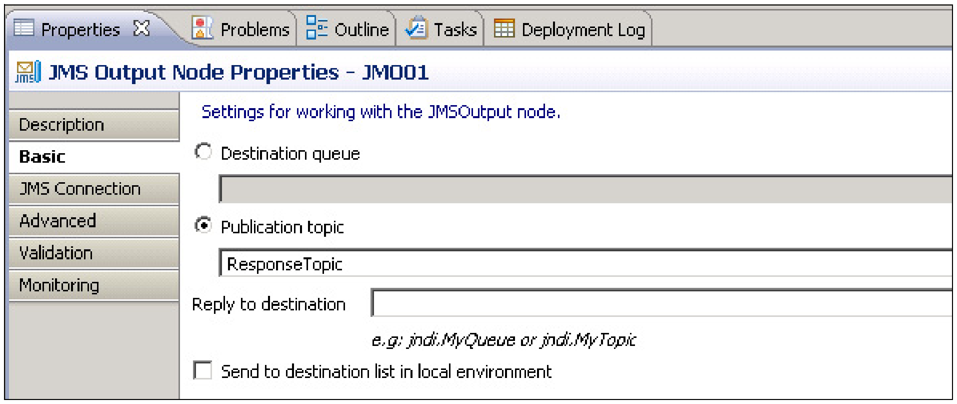


JMS provider name*	IBM MessageSight
Initial context factory	com.sun.jndi.fscontext.RefFSContextFactory
Location JNDI bindings	file://C:/JNDI/IMA-SAMPLE
Connection factory name	connFactory1
Backout destination	
Backout threshold	0

Figure 3-13 JMSInput node properties showing configuration of the JMS connection

Figure 3-14 shows the Basic configuration of a `JMSOutput` node. This output node is used in a publish/subscribe scenario. Therefore, the Publication topic is selected. Note that the value, `ResponseTopic`, is one of the administered objects created in the bindings file.

The JMS Connection tab for the `JMSOutput` node is similar to the tab described for the `JMSInput` node in Figure 3-13.



Settings for working with the JMSOutput node.

○ Destination queue

[✓ Publication topic]

ResponseTopic

Reply to destination

e.g: jndi.MyQueue or jndi.MyTopic

☐ Send to destination list in local environment

Figure 3-14 JMSOutput node Basic configuration

3.3 Developing stand-alone applications using the IBM MessageSight JMS classes

This section describes the development of JMS publisher and subscriber applications using IBM MessageSight JMS.

Before commencing development of JMS applications for IBM MessageSight, ensure that you have access to the IBM MessageSight JMS Client Pack.

It is assumed that the reader is familiar with, and comfortable with, JMS concepts and terminology. It is important to remember that the IBM MessageSight appliance provides the functionality of a JMS broker (or middle-man).

Important: This section does not contain the complete code for the sample applications. In the following sections, the code snippets illustrate the relevant methods used for connecting, publishing messages to, and subscribing from, the IBM MessageSight appliance. You can download the complete code from the IBM Redbooks website. See Appendix D, "Additional material" on page 341 for the download instructions.

3.3.1 IBM MessageSight JMS Client Pack

You can download the JMS Client pack from the IBM developerWorks website:

`https://www.ibm.com/developerworks/community/blogs/c565c720-fe84-4f63-873f-607d87787327/entry/download?lang=en`

The JAR files required for IBM MessageSight JMS are in the `jms/lib` directory. This directory must be included in your `CLASSPATH`.

For IBM MessageSight V1.0.0.1, there are five JARs in this directory:

- `fscontext.jar`
- `imaclientjms.jar`
- `jms.jar`
- `jmssamples.jar`
- `providerutil.jar`

The Javadoc for the IBM MessageSight JMS classes can be found in the `jms/doc` directory.

Additional sample codes are in the `jms/samples` directory.

3.3.2 IBM MessageSight JMS and JNDI

WebSphere MQ provides facilities by which Java Naming and Directory Interface (JNDI)-administered objects can be configured for the WebSphere MQ implementation of JMS. These facilities include:

- Use of the WebSphere MQ Explorer, specifically, the `JMS Administered Objects` folder. WebSphere MQ Explorer provides a user-friendly graphical approach to JNDI configuration.
- Use of the command-line utility, `JMSAdmin`.

The current release of IBM MessageSight JMS does not provide a tool or utility for configuring JNDI.

A sample `.bindings` file is supplied with the JMS Client Pack. This file shows the expected formats for the JNDI object configurations. Any text editor can be used to view and edit this file.

Additionally, a Java sample named `JMSSampleAdmin.java` is provided in the `jms/samples` directory. This sample program takes a configuration file as input and produces a required `.bindings` file. This sample application can also take a `.bindings` file as input, and produces a legible configuration file as output.

For more information about using JMS applications with IBM MessageSight, see *Creating administered objects* in the IBM MessageSight Information Center:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Developing/devjms_creatingadministeredobjects.html

3.3.3 Message delivery in IBM MessageSight

This section provides information about how IBM MessageSight handles message delivery, and compares and explains message reliability and delivery performance when using JMS and MQTT.

IBM MessageSight enables JMS message acknowledgement to be disabled with the use of the `DisableACK ConnectionFactory` property. This type of publication is the fastest, but is also the least reliable. Use it only when occasional message loss can be tolerated. Message loss can occur if the publisher is disconnected, the server fails, or transient network issues occur.

For more information about message delivery in IBM MessageSight, see the IBM MessageSight Information Center:

`https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/index.jsp?topic=%2Fcom.ibm.ism.doc%2FPlanning%2Fov00102_.html`

3.3.4 A sample JMS publisher

This section describes the implementation of a sample message publisher using IBM MessageSight JMS. This sample application uses a properties file for configuration information.

One key aspect that this sample illustrates is the ability to reconnect to an alternative IBM MessageSight appliance if one fails. IBM MessageSight can be configured in a high availability (HA) configuration using a pair of appliances. However, the individual appliances in the HA configuration use unique network addresses. Therefore, a JMS client must be aware of both addresses to use this HA capability.

The application does not use a `.bindings` file, but creates the required administered objects internally within the sample application. It uses the provided configuration properties to perform this object creation.

The sample publisher application carries out the following steps:

1. Parse the command-line arguments and process the properties file.
2. Create a connection to the JMS server (the IBM MessageSight appliance).
3. Set the Client identifier for the connection. If a user name and password are supplied, these are used to establish the connection.
4. Create a session.
5. Create a publisher for the required topic.
6. Publish the required number of messages.
7. If the connection to the JMS server is broken while sending the messages, the application attempts to reconnect to any specified alternative addresses (for example, the second appliance in an HA pair).
8. The application stops when the required number of messages are published.

Example 3-10 shows the required Java imports for this publisher sample.

Example 3-10 Publisher Java imports

```
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Topic;
import javax.jms.Session;
import javax.jms.TextMessage;

import java.io.*;
import java.util.Properties;

import com.ibm.ima.jms.ImaJmsException;
import com.ibm.ima.jms.ImaJmsFactory;
import com.ibm.ima.jms.ImaJmsObject;
import com.ibm.ima.jms.ImaProperties;
```

Example 3-11 shows the configuration file used by the JMS Publisher application.

Example 3-11 JMS publisher configuration file

```
#
# IBM MessageSight sample JMS Publisher
# Configuration file
#
ima.servers=192.168.121.135, 192.168.121.227
ima.port=16102
client.id=ITSOPUB01
topic=ITSO/Sample/Topic01
msg.count=10000
msg.body="This is an IMA JMS test message"
msg.rate=500
msg.pers=false
#
disable.ack=false
#
# Username and Password are used for additional security
#
#user.name=testuser
#user.passwd=testpassword
```

Example 3-12 shows the creation of the JNDI-administered objects required by the publisher application.

Example 3-12 Create administered objects

```
/**
* Get connection factory administered object.
*
* @return ConnectionFactory object.
*
* Note: In production applications, this method would retrieve a
*       ConnectionFactory object from a JNDI repository.
*/
public static ConnectionFactory getCF() throws JMSException
{
        ConnectionFactory cf = ImaJmsFactory.createConnectionFactory();
/*
* NOTE: For high availability configurations, the serverList
* must contain the list of IBM MessageSight server host names or
* IP addresses so that the client
* can attempt to connect to alternative hosts if connection
* (or reconnection) to the primary fails.
*/
((ImaProperties) cf).put("Server", serverList);
((ImaProperties) cf).put("Port", serverPort);
if (disableACK) {
   ((ImaProperties) cf).put("DisableACK", true);
   System.err.println("ACKs disabled in ConnectionFactory!");
}

/*
* After setting properties on an administered object, it is a
* best practice to run the validate() method to assure all
* specified properties are recognized by the IBM MessageSight
* JMS client.
*/
        ImaJmsException errors[] = ((ImaProperties) cf).validate(true);
        if (errors != null)
            throw new RuntimeException("Invalid properties provided for the
connection factory.");
        return cf;
}

/**
* Create topic administered object.
*
* @return Topic object.
*
* Note: In production applications, this method would retrieve a
*       Topic object from a JNDI repository.
*/
    public static Topic getTopic() throws JMSException
{
        Topic dest = ImaJmsFactory.createTopic(topicName);
        return dest;
}
}
```

Example 3-13 shows the connection being established to the JMS server (the IBM MessageSight appliance).

Example 3-13 Establish connection

```
/**
* Establish a connection to the server.  Connection attempts
* are retried until successful or until the
* specified timeout for retries is exceeded.
*/
public static boolean doConnect() {
    int connattempts = 1;
    boolean connected = false;
    long starttime = System.currentTimeMillis();
    long currenttime = starttime;

    /*
     *  Try for up to connectTimeout milliseconds to connect.
     */
    while (!connected && ((currenttime - starttime) < connectTimeout)) {
        try { Thread.sleep(5000); } catch (InterruptedException iex) {}
       if (debug) System.out.println("Attempting to connect to server (attempt
" + connattempts + ").");
        connected = connect();
        connattempts++;
        currenttime = System.currentTimeMillis();
    }
    return connected;
}

/**
* Connect to the server
*
* @return Success (true) or failure (false) of the attempt to connect.
*/
public static boolean connect()
{
    /*
     * Create connection factory and connection
     */
    try {
        fact = getCF();
        if (userPass != null)
            conn = fact.createConnection(userName, userPass);
        else
            conn = fact.createConnection();
        conn.setClientID(clientID);

    /*
     * Check that we are using IBM MessageSight JMS
     * administered objects before calling
     * provider specific check on property values.
     */
       if (fact instanceof ImaProperties)
       {
           /*
           * For high availability, the connection factory stores the
           * list of IBM MessageSight server host names or
           * IP addresses.  Once connected, the connection object
           * contains only the host name or IP address
           * to which the connection is established.
           */
            System.out.println("Connected to " +
          ((ImaProperties)conn).getString("Server"));
       }

        /*
         * Connection has succeeded.  Return true.
         */
        return true;
     } catch (JMSException jmse) {
        System.err.println(jmse.getMessage());
        /*
         * Connection has failed.  Return false.
         */
        return false;
    }
}
```

Example 3-14 shows the publication of the messages. Each message is appended with a sequence number. Appropriate delays are calculated to maintain the requested message rate, messages per second. If the connection is broken during message publication, attempts are made to re-establish the connection to a specified JMS server (appliance) and continue publishing messages until all requested messages are sent.

Example 3-14 Publish messages

```
/**
* Publish messages to specified Topic.
*/
public static void doSend() throws JMSException
{
        Session sess = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
        System.out.println("Client ID is: " + conn.getClientID());
        Topic haTopic = getTopic();
        // Create the message producer
        MessageProducer prod = sess.createProducer(haTopic);
        // Set the required message persistence.
        if (msgPers) {
        prod.setDeliveryMode(DeliveryMode.PERSISTENT);
        System.out.println("Using PERSISTENT messages");
        } else {
        prod.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
        System.out.println("Using NON-PERSISTENT messages");
        }
        conn.start();
        long startTime = System.currentTimeMillis();

        TextMessage tmsg;
        int msgcnt = 0;
        while ((sendcount < count) && !senddone) {
            try {
                tmsg = sess.createTextMessage(msgtext+" - "+sendcount);
                if (debug)
                    System.out.println("Sending message: " + sendcount + " " +
ImaJmsObject.toString(tmsg, "b"));
                prod.send(tmsg);
                sendcount++;
                msgcnt++;
                if (sendcount == count)
                    senddone = true;
            } catch (Exception ex) {
                ex.printStackTrace(System.err);
/*
* If an exception is thrown while messages are sent, check to see whether the
connection
* has closed.  If it has closed, then attempt to reconnect and continue
sending.
*/
                if (!senddone && isConnClosed()) {
                    reconnectAndContinueSending();
                } else {
/*
* If all messages have already been sent or if the connection remains active
despite
* the exception, the error should not be addressed with an attempt to
reconnect.
* Set senddone to true in order to exit this sample application.
*/
                    senddone = true;
                }
                break;
            }

            // Calculate required delay (sleep) to achieve the
            // requested message rate.
            long currTime = System.currentTimeMillis();
            double elapsed = (double) (currTime - startTime);
            double projected = ((double) msgcnt / (double) msgsPerSecond) *
1000.0;
            if (elapsed < projected) {
                double sleepInterval = projected - elapsed;
                try {
                    Thread.sleep((long) sleepInterval);
                } catch (InterruptedException e) {
                }
            }
        }
    }
```

Example 3-15 shows code to detect that the connection is lost, and to attempt to re-establish the connection.

Example 3-15   Broken connection detection and connection re-establishment

```
/**
* Check to see if connection is closed.
*
* When the IBM MessageSight JMS client detects that the connection
* to the server has been broken, it marks the connection object closed.
*
* @return True if connection is closed or if the connection object is null,
false otherwise.
*/
    public static boolean isConnClosed() {
        if (conn != null) {
/*
* Check the IBM MessageSight JMS isClosed connection property
* to determine whether the connection state is closed.
* This mechanism for checking whether the connection is closed
* is a provider-specific feature of the IBM MessageSight
* JMS client.  So check first that the IBM MessageSight JMS
* client is being used.
*/
            if (conn instanceof ImaProperties) {
                return ((ImaProperties) conn).getBoolean("isClosed", false);
            } else {
                /*
                 * We cannot determine whether the connection is closed so
return false.
                 */
                return false;
            }
        } else {
            return true;
        }
    }

    /**
     * Reconnect and continue sending messages.
     */
    public static void reconnectAndContinueSending() throws JMSException {
        System.out.println("Connection is closed.  Attempting to reconnect and
continue sending.");
        boolean reconnected = doConnect();

        if (reconnected) {
            doSend();
        } else {
/*
* If we fail to reconnect in a timely manner, then set senddone
* to true in order to exit from this sample application.
*/
            System.err.println("Timed out while attempting to reconnect to the
server.");
            senddone = true;
        }
    }
```

3.3.5 A sample JMS subscriber

This section describes the implementation of a sample message subscriber using IBM MessageSight JMS. This sample application uses a properties file for configuration information.

As for the publisher application, this sample illustrates the ability to reconnect to an alternative IBM MessageSight appliance if one fails. IBM MessageSight can be configured in an HA configuration using a pair of appliances. However, the individual appliances in the HA configuration use unique network addresses. Therefore, a JMS client must be aware of both addresses to use this HA capability.

The application does not use a `.bindings` file, but creates the required administered objects internally within the sample application. It uses the provided configuration properties for object creation.

The sample subscriber application carries out the following sequence of steps:

1. Parse the command-line arguments and process the properties file.
2. Create a connection to the JMS server (the IBM MessageSight appliance).
3. Set the Client identifier for the connection. If a user name and password are supplied, these are used to establish the connection.
4. Create a session.
5. Create a subscriber for the required topic.
6. Receive the required number of messages.
7. If the connection to the JMS server is broken while receiving the messages, the application attempts to reconnect to any specified alternative addresses (for example, the second appliance in an HA pair).
8. The application stops when the required number of messages are received.

Tip: This subscriber application uses synchronous message reception. Asynchronous message reception can also be implemented with the use of a message listener.

Example 3-16 shows the required Java imports for this subscriber sample.

Example 3-16 Subscriber Java imports

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;

import com.ibm.ima.jms.ImaJmsException;
import com.ibm.ima.jms.ImaJmsFactory;
import com.ibm.ima.jms.ImaProperties;
```

The steps to create the required administered objects (`connectionFactory` and `Topic`), and to establish a connection to the JMS server (IBM MessageSight appliance), are identical to the sample code snippets shown for the publisher sample in 3.3.4, “A sample JMS publisher” on page 63.

Example 3-17 shows the configuration file used by the JMS subscriber application.

Example 3-17 JMS subscriber configuration file

```
# IBM MessageSight sample JMS Subscriber
# Configuration file
#
ima.servers=192.168.121.135, 192.168.121.227
ima.port=16102
client.id=ITSOSUBS01
topic=ITSO/Sample/Topic01
msg.count=5000
# Receive timeout is in seconds - wait to receive first message.
rcv.timeout=15
# Durable subscription name
durasubs.name=DURASUBS01
#
# Username and Password are used for additional security
#
#user.name=testuser
#user.passwd=testpassword
```

Example 3-18 shows the reception of the messages. This message subscriber sample application can, optionally, use a durable subscription. If the connection is broken during message reception, attempts are made to re-establish the connection to a specified JMS server (appliance) and continue receiving messages until all requested messages have been received.

Example 3-18 Message reception

```
/**
* Receive messages via durable subscription.
*/
    public void doReceive() throws JMSException {

    // Create a session
        sess = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Topic haTopic = getTopic();
        if (duraSubsName != null) {
        // Create a Durable Subscriber
        cons = sess.createDurableSubscriber(haTopic, duraSubsName);
        System.out.println("Created a Durable Subscription named 
"+duraSubsName);
        } else {
        // Create a standard message consumer
        cons = sess.createConsumer(haTopic);
        }

        conn.start();
        int i, firstmsg;
        i = firstmsg = recvcount;

        do {
            int timeout = 10000;
            if (i == firstmsg)
                timeout = receiveTimeout;
            try {
                Message msg = cons.receive(timeout);
                if (msg == null) {
                    throw new JMSException("Timeout in receive message: " + i);
                } else {
                    synchronized (IMAJMSSubscriber.class) {
                        recvcount++;
                        lastrecvtime = System.currentTimeMillis();
                    }
                    if (msg instanceof TextMessage) {
                        TextMessage tmsg = (TextMessage) msg;
                        System.out.println("Received message " + i + ": " + 
tmsg.getText());
                    } else {
                        System.out.println("Received message " + i);
                    }
                    i++;
                }
            } catch (Exception ex) {
                ex.printStackTrace(System.err);
/*
* If an exception is thrown while messages are received, check to
* see whether the connection has closed.
* If it has closed, then attempt to reconnect and continue receiving.
*/
                if (!recvdone && isConnClosed()) {
                    this.reconnectAndContinueReceiving();
                } else {
                    synchronized (IMAJMSSubscriber.class) {
/*
* If all messages have already been received or if the
* connection remains active despite the exception,
* the error should not be addressed with an attempt to reconnect.
* Set recvdone to true in order to exit this sample application.
*/
                        recvdone = true;
                    }
                }
                break;
            }
        } while (recvcount < count);
        synchronized (IMAJMSSubscriber.class) {
            recvdone = true;
            if (recvcount < count)
                System.err.println("doReceive: Not all messages were received:
" + recvcount);
        }
// Close off cleanly, especially if a DurableSubscriber was used.
cons.close();
if (duraSubsName != null)
   sess.unsubscribe(duraSubsName);
}
```

Similar to the publisher sample application, the subscriber application detects a broken connection to the JMS server (appliance) and attempts to re-establish the connection.

3.4 IBM MessageSight resource adapter for Java Platform Enterprise Edition applications

The IBM MessageSight resource adapter is a new feature provided in version 1.1 that enables direct connection from IBM MessageSight to WebSphere Application Server. This simplified Java Platform, Enterprise Edition (Java EE) architecture supports faster configuration, easier monitoring, and easier scaling of Java EE solutions. The IBM MessageSight resource adapter supports inbound and outbound communication.

IBM MessageSight provides a resource adapter that enables Java EE applications to communicate directly with IBM MessageSight by using the Java EE Connector Architecture (JCA) V1.6 interfaces. In addition to the benefits listed in the previous paragraph, this Java EE architecture lowers operating costs.

The IBM MessageSight resource adapter supports two types of communication between an application and IBM MessageSight:

- Inbound communication

  A JMS message that arrives at a JMS destination is delivered to a message-driven bean (MDB), which processes the message asynchronously.

- Outbound communication

  An application starts a connection to IBM MessageSight, and then sends JMS messages to JMS destinations, and receives JMS messages from JMS destinations in a synchronous manner.

The IBM MessageSight resource adapter is supported on WebSphere Application Server Version 8.0 or later. For more detail about IBM MessageSight resource adapters, see the IBM MessageSight Information Center:

http://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/workingwithjmsra.html

In IBM MessageSight V1.1, you can configure durable and non-durable topic subscriptions to be shared by using the IBM MessageSight JMS client. In a non-shared subscription, each consumer of a particular topic receives a copy of all of the messages that are published to that topic string. In a shared subscription, each message is delivered to only one of the consumers for that subscription. This is similar to message sharing on a multi-consumer queue.

Unshared *durable* subscriptions must have a client ID specified. The client ID specifies the namespace for the subscription name. Unshared *non-durable* subscriptions do not require a client ID to be specified.

Shared subscriptions can be bound to a client ID, or can exist within a global namespace. If a client ID is specified, the subscription is bound to only that client ID. If no client ID is specified, the global namespace is used. By using a global namespace, it is possible to share a subscription between multiple connections. This configuration can be used to allow load balancing in an application server cluster.

For more detail about IBM MessageSight shared subscriptions, see the IBM MessageSight Information Center:

`http://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Developing/sharedsubscriptionsinjms.html`

4

Security in IBM MessageSight

This chapter describes a few of the security options available to set up secure messaging with IBM MessageSight. The focus is more on devices that try to connect to IBM MessageSight than on the back-end integration of the appliance.

This chapter provides information about the following topics:

- Authentication of a device connecting with IBM MessageSight
- Authorization of a device connecting with IBM MessageSight
- Transport Layer Security
- Combining TLS with connection and messaging policy

IBM MessageSight is configured to provide Federal Information Processing Standards (FIPS) 140-2 support. For more information about FIPS, see the following website:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Security/se00012_.html

4.1 Authentication of a device connecting with IBM MessageSight

This section describes how to set up user authentication with IBM MessageSight.

4.1.1 Users and groups

In IBM MessageSight, there are two types of users and groups:

- Administration users
- Messaging users

Administration users

The administration users have access to IBM MessageSight. There are three types of such users:

- System administrator
- Messaging administrator
- Appliance user

Among other actions, an administrator can change the time zone of servers, or configure Secure Sockets Layer/Transport Layer Security (SSL/TLS) connections.

Messaging users

The messaging users are handled by the client application. The authentication can be set up based on users and groups defined on IBM MessageSight. A user ID is set up on IBM MessageSight, and can be associated with a group for easier maintenance.

For example, you can define an `operatorE1` group that has a collection of user IDs belonging to operators of a particular piece of equipment. These user IDs might have an access control defined differently than other groups.

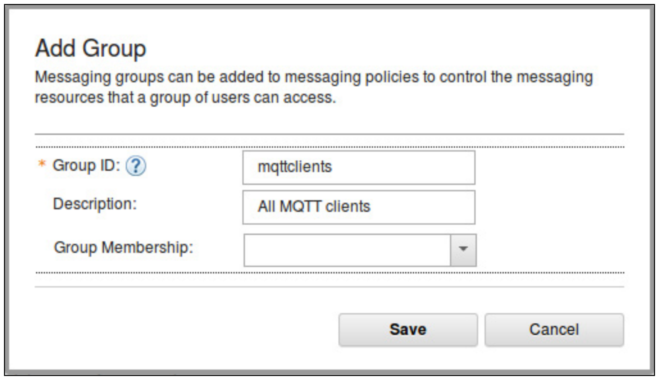
Similarly, you can create a group definition based on the user's geographic position. For example, all users of a device from a given warehouse in a large site can be designated to be in the `warehouseA` group.

In case of an MQTT client, the user ID defined on IBM MessageSight maps to the MQTT user name.

4.1.2 Creating a user ID and group with the web UI

To create users and groups with the IBM MessageSight web user interface (web UI), follow these steps:

1. Go to **Messaging** → **Users and groups**.
2. Start with creating a new group first. Under the Messaging Groups section, click the plus (**+**) icon to add a new group.
3. In the resulting dialog box, enter details, as shown in Figure 4-1.


Add Group

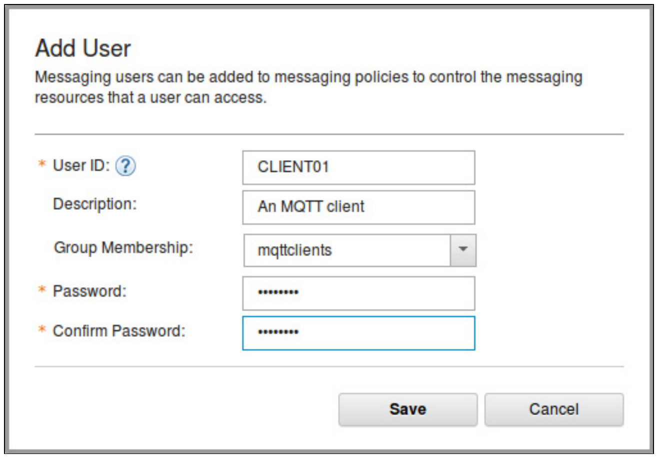
Messaging groups can be added to messaging policies to control the messaging resources that a group of users can access.

* Group ID:	mqttclients
Description:	All MQTT clients
Group Membership:	

Figure 4-1 Adding a new user group to IBM MessageSight

4. After this group is created, you can define and add users to this group. This user ID will be a messaging user and not a web UI user. On the same web UI page, go to the Messaging Users section and click the plus (**+**) icon to add a new user.

5. Enter the details, as shown in Figure 4-2 to add a new user. Note that, the group membership is set to `mqttclients` that was created earlier.


Add User

Messaging users can be added to messaging policies to control the messaging resources that a user can access.

* User ID:	CLIENT01
Description:	An MQTT client
Group Membership:	mqttclients
* Password:	••••••••
* Confirm Password:	••••••••

Figure 4-2 Adding a new user to the group

4.1.3 Creating a user ID and group on the command line

You can also create the user ID and group ID using the command-line interface (CLI), as shown in Example 4-1.

Example 4-1 Creating a new group and user ID on IBM MessageSight

```
Console> imaserver group add "GroupID=mqttclients" "Description=All
MQTT clients"
The requested configuration change has completed successfully.
Console> imaserver user add "UserID=CLIENT01" "Type=Messaging"
"Password=new00pwd" "GroupMembership=mqttclients" "Description=An MQTT
client"
The requested configuration change has completed successfully.
```

Note that the type of user mentioned, `Type=Messaging`, indicates that the user ID being created is a messaging user and not a web UI user.

For more information about authentication, see the IBM MessageSight Information Center:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Security/se00010.html

4.2 Authorization of a device connecting with IBM MessageSight

This section describes how to set up authorizations on IBM MessageSight, with a focus on MQTT connections.

4.2.1 Authorization schemes

You can set up the authorization on any combination of attributes related to network, client identification, and message data. For example, you can allow the connections on the basis of network attributes, such as certain IP addresses, or on the basis of the protocol (Java Message Service (JMS) or MQTT) used by a client and its identifier.

To set up this kind of access control, you must create a connection policy.

After a connection is established, another level of access control can be defined based on the action that the client can take. For example, users with certain types of equipment can be allowed to publish on topics as allocated to them. Similarly, they can be restricted to subscribe on certain topics on which broadcast messages are received.

To set up this kind of access control, you must create a messaging policy.

4.2.2 Authorizing MQTT clients

An MQTT client on a network is uniquely identified with its client ID. Optionally, an MQTT client can connect to IBM MessageSight with a user name and a password. You can set up this authorization based on the client identifier, user name, and password. The user name and password are supported by MQTT protocol.

However, the client and broker communication can also be secured using TLS, too. For more detail, see 4.3, “Transport Layer Security” on page 87.

For a full description about connecting to a server, see the MQTT specification:

http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#connect

Authorizing with a client identifier

According to the MQTT specification, an MQTT client is required to have a client ID. IBM MessageSight can be set up to authorize access based on this client ID. Figure 4-3 shows this scheme.

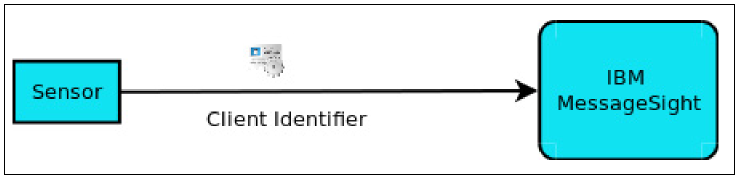


Figure 4-3 An MQTT client authenticating with a client identifier

With this scheme, it is possible to limit connections to only those MQTT clients who have a *known client ID* in a network. For example, a topology of MQTT clients can use the MAC address of the devices in the networks as their client ID.

Restriction: Note that such a connection policy is *not* secure if a malicious client can spoof a valid identifier.

Authorizing with a client ID, user name, and password

According to the MQTT specification, an MQTT client can optionally pass a user name and a password for authentication. This user name and password combination can also be used to authorize with IBM MessageSight. Figure 4-4 shows this scheme.

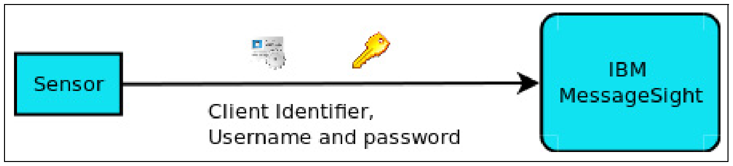


Figure 4-4 An MQTT client authenticating with a client ID, user name, and password

In this scheme, an additional layer of authorization is set up on the basis of an MQTT user name and password. Such a scheme is suitable for MQTT clients that can have multiple human operators over time.

For example, an electronic kit can have probes for temperature, a voltmeter, or an ammeter to read parameters of a piece of equipment. However, this kit might be used by different personnel over time. Therefore, an additional security layer can be added so that only *known personnel using known devices* can connect to IBM MessageSight.

4.2.3 Setting up authorization in IBM MessageSight: Connection policy

To set up authorization in IBM MessageSight, you can start with a connection policy. This connection policy is referred to in one or more endpoints on the appliance. In other words, a given endpoint on the appliance can have one or more connection policies applied to it. A client connection is allowed when at least one of the connection policies evaluates to `true`.

For complete information about connection policies, see the IBM MessageSight Information Center:

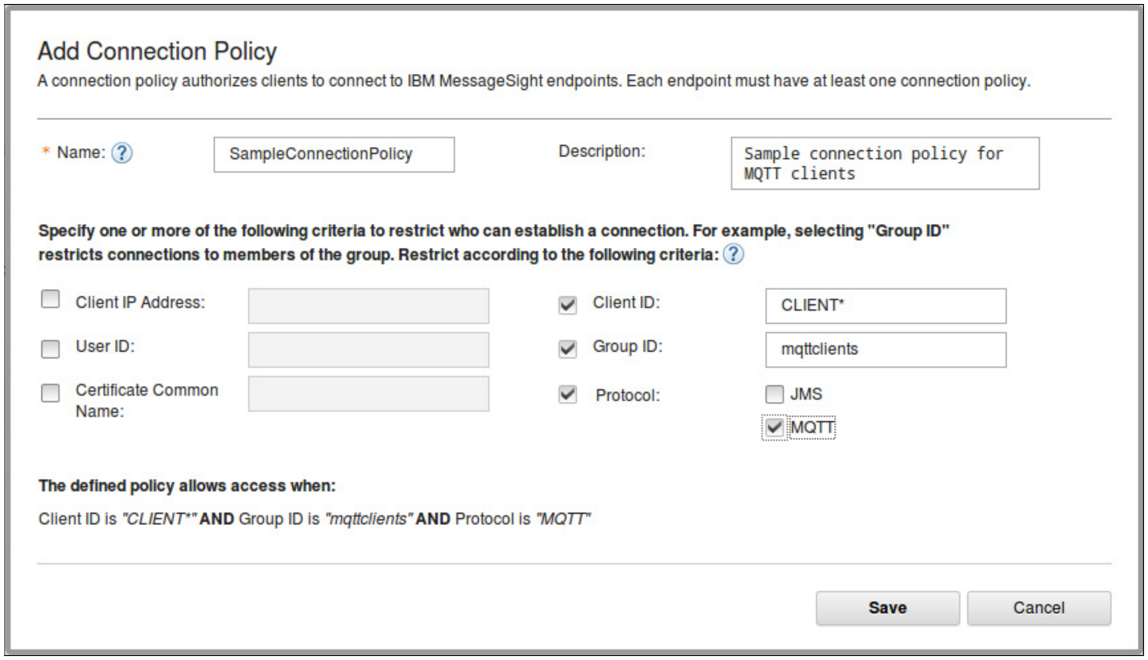
https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Security/se00003_.html

Creating a connection policy with the web UI

To create a connection policy with the web UI, follow these steps:

1. Go to **Messaging** → **Message Hubs**.
2. From the message hub, select a hub that you want to work with, and click the pencil icon to edit it.
3. The resulting page shows a page for connection and messaging policies, along with the end-points related with the selected hub. Click the **Connection Policies** pane, and click the green plus (**+**) icon to create a new connection policy.

Figure 4-5 shows a sample connection policy for MQTT clients, with criteria set for client IDs and group IDs. A group ID enables user IDs defined for that group. Alternatively, the policy can be set up for a given user ID only.



Add Connection Policy

A connection policy authorizes clients to connect to IBM MessageSight endpoints. Each endpoint must have at least one connection policy.

* Name:	SampleConnectionPolicy	Description:	Sample connection policy for MQTT clients

Specify one or more of the following criteria to restrict who can establish a connection. For example, selecting "Group ID" restricts connections to members of the group. Restrict according to the following criteria:

☐ Client IP Address:		✓ Client ID:	CLIENT*
☐ User ID:		✓ Group ID:	mqttclients
☐ Certificate Common Name:		✓ Protocol:	☐ JMS ✓ MQTT

The defined policy allows access when:

Client ID is *"CLIENT*"* **AND** Group ID is *"mqttclients"* **AND** Protocol is *"MQTT"*

Figure 4-5 Sample connection policy for MQTT clients

Creating a connection policy using the CLI

You can also create a connection policy using the CLI, as shown in Example 4-2.

Example 4-2 Command-line option to create a connection policy for MQTT clients

```
Console> imaserver create ConnectionPolicy
"Name=SampleConnectionPolicy" "Description=Sample connection policy for
MQTT clients" "ClientID=CLIENT*" "GroupID=mqttclients" "Protocol=MQTT"
The requested configuration change has completed successfully.
```

Note the following aspects of the code in Example 4-2:

- The client ID has the word `CLIENT` and an asterisk as the wildcard character to match zero or more characters. Therefore, MQTT clients with various client IDs, such as `CLIENT99`, `CLIENTA`, or `CLIENT`, are all considered valid.
- To limit connections to a set of specific client identifiers, individual connection policies for each one must be created.

- The group ID is set to `mqttclients`, which means that the client must connect to a user name that belongs to this group. The password for this user name must match with the name that was provided when creating the user name.

4.2.4 Setting up authorization in IBM MessageSight: Messaging policy

After a client is successfully connected to the appliance, another layer of access control exists, such that a given client has the appropriate level of access to correct resources. An endpoint is associated with one or more messaging policies. These policies are applied after the connection policy allows a connection to pass through.

For more information about messaging policies, see the IBM MessageSight Information Center:

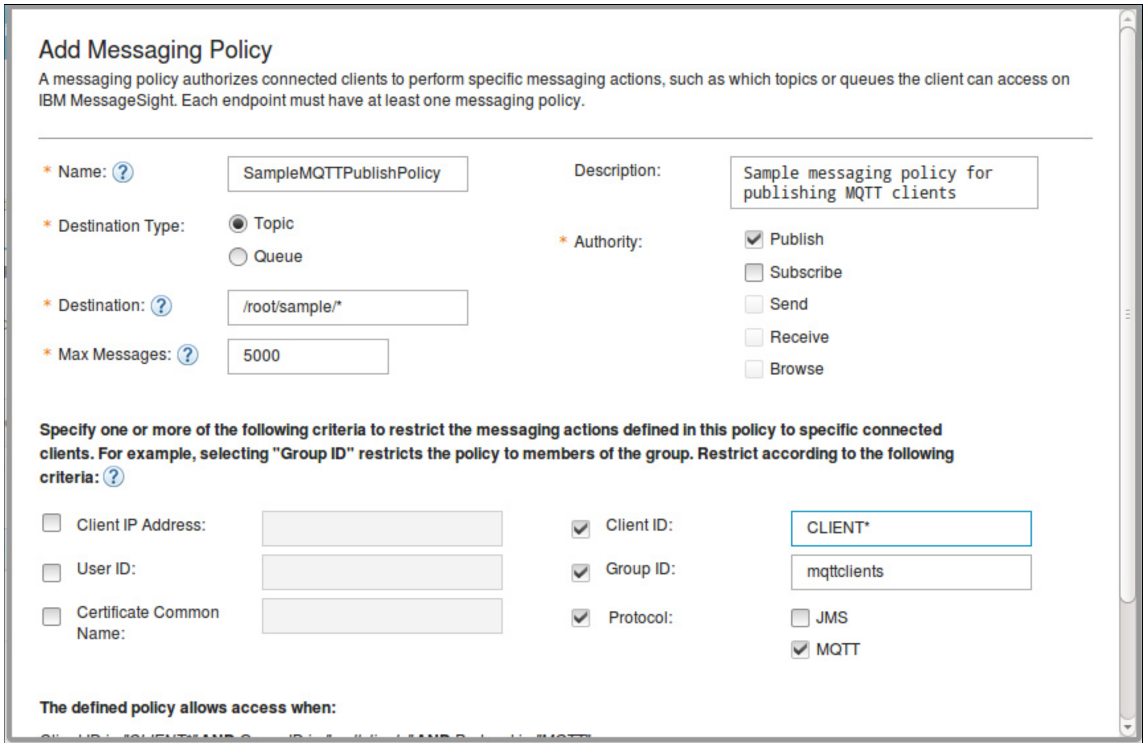
`https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Security/se00004_.html`

Creating a messaging policy with the web UI

To create a messaging policy with the web UI, follow these steps:

1. Go to **Messaging** → **Message Hubs**.
2. From the message hub, select a hub that you want to work with, and click the pencil icon to edit it.
3. The resulting page shows pages for connection and messaging policies, along with the endpoints related to the selected hub. Click the **Messaging Policies** pane, and click the green plus (**+**) icon to create a new messaging policy.

Figure 4-6 shows a sample messaging policy for MQTT clients.



Add Messaging Policy

A messaging policy authorizes connected clients to perform specific messaging actions, such as which topics or queues the client can access on IBM MessageSight. Each endpoint must have at least one messaging policy.

* Name:	SampleMQTTPublishPolicy	Description:	Sample messaging policy for publishing MQTT clients
* Destination Type:	[● Topic] ○ Queue	* Authority:	[✓ Publish] ☐ Subscribe ☐ Send ☐ Receive ☐ Browse
* Destination:	/root/sample/*		
* Max Messages:	5000		

Specify one or more of the following criteria to restrict the messaging actions defined in this policy to specific connected clients. For example, selecting "Group ID" restricts the policy to members of the group. Restrict according to the following criteria:

☐ Client IP Address:		✓ Client ID:	CLIENT*
☐ User ID:		✓ Group ID:	mqttclients
☐ Certificate Common Name:		✓ Protocol:	☐ JMS [✓ MQTT]

The defined policy allows access when:

Figure 4-6 Sample messaging policy for MQTT clients

Creating a messaging policy with the CLI

You can also create a connection policy using the CLI, as shown in Example 4-3.

Example 4-3 Command-line option to create a messaging policy for MQTT clients

```
Console> imaserver create MessagingPolicy "Name=SampleMessagingPolicy"
"Description=Sample messaging policy for publishing MQTT clients"
"DestinationType=Topic" "Destination=/root/sample/*"
"ClientID=CLIENT*" "GroupID=mqttclients" "Protocol=MQTT"
The requested configuration change has completed successfully.
```

4.3 Transport Layer Security

This section describes how to enable TLS for MQTT connections. With TLS, you can achieve (independently or in combination) the following security objectives:

- Encrypted communication with IBM MessageSight
- Client authentication based on certificates
- Client authentication based on user name and password

Enabling TLS in IBM MessageSight is a three-step process:

1. Creating a certificate profile
2. Creating a security profile
3. Associating a security profile to an endpoint

Additionally, a step for adding trusted certificates must be completed if the security profile is set to authenticate a client based on the client certificate.

Use TLS 1.1 and 1.2 as the default for secure communication with IBM MessageSight, MQTT clients, and JMS clients.

Important: In the rest of this section, all screen captures that refer to certificates are fictitious and self-signed. A self-signed certificate is not secure and ought to be avoided in security setup.

Note that IBM MessageSight expects that all certificates are in the privacy enhanced mail (PEM) format.

For more information about the `.pem` file extension, see the *OpenSSL* topic in the IBM WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r1/index.jsp?topic=%2Fcom.ibm.mq.doc%2Fq114050_.htm

For a background on digital certificates in general, see the *Digital certificates* topic in the IBM MessageSight Information Center:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Reference/_Topics/sy10530_.html

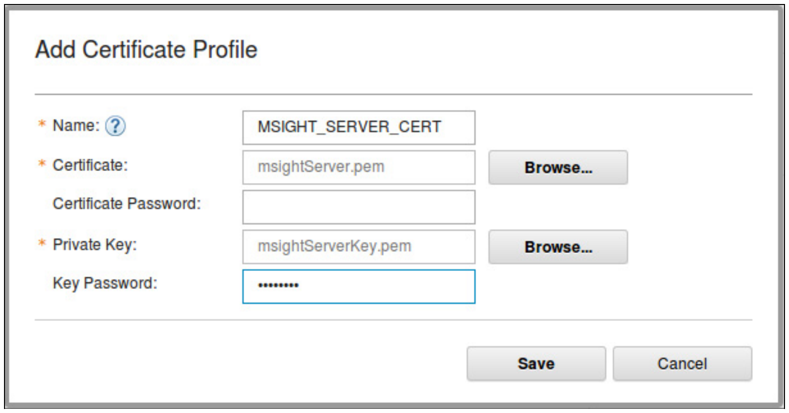
4.3.1 Certificate profile

To enable TLS, the first step is to create a certificate profile using the certificate and key pair designated for IBM MessageSight. You can have as many profiles as the number of endpoints you have in a message hub. Alternatively, you can use one certificate profile throughout the message hub.

Creating a certificate profile using the web UI

To set up certificate profile using the web UI, follow these steps:

1. Go to **Appliance** → **Security Settings**.
2. Under Certificate Profiles, click the green plus (**+**) icon to create a new certificate profile. Figure 4-7 shows the panel for creating a new certificate profile.



Add Certificate Profile

* Name:	MSIGHT_SERVER_CERT	
* Certificate:	msightServer.pem	Browse...
Certificate Password:		
* Private Key:	msightServerKey.pem	Browse...
Key Password:	••••••••	

Figure 4-7 Adding a new certificate profile using the web UI

3. The password fields are not marked as mandatory. However, at a minimum, the private key is expected to be secured with a password and so, in Figure 4-7, enter the password for the private key.

Note that IBM MessageSight expects that the certificate and the key selected are in the PEM format.

Creating a certificate profile using the CLI

To create a certificate profile using the CLI, complete the following tasks in order:

1. Import the certificate and key files to the IBM MessageSight appliance.
2. Apply the certificate and key.
3. Create the certificate profile.

In Example 4-4, you import the certificate and key from a server using the **`scp`** protocol.

Example 4-4 Importing certificate and key for the appliance

```
Console> file get scp://user@certs.repo.org/servers/msightServer.pem
user@certs.repo.org's password:
The requested configuration change has completed successfully.
```

```
Console> file get scp://user@certs.repo.org/servers/msightServerKey.pem
user@certs.repo.org's password:
The requested configuration change has completed successfully.
```

In Example 4-5, you apply the certificate and key.

Example 4-5 Applying the certificate and the key

```
Console> imaserver apply Certificate "CertFileName=msightServer.pem"
"CertFilePassword=" "KeyFileName=msightServerKey.pem"
"KeyFilePassword=redb00ks"
```

Note that the argument CertFilePassword is provided with no value after the equals sign (**=**) to indicate that no password is required for the certificate file.

In Example 4-6, you create the certificate profile using the certificate and the key file that you imported (Example 4-4 on page 88) and applied (Example 4-5).

Example 4-6 Creating a certificate profile

```
Console> imaserver create CertificateProfile "Name=MSIGHT_SERVER_CERT"
"Certificate=msightServer.pem" "Key=msightServerKey.pem"
```

For more information about importing certificates, see *Importing a server certificate using the command line*:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00341_.html

For more information about creating a certificate profile using the CLI, see *Creating a certificate profile on the command line*:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00333_.html

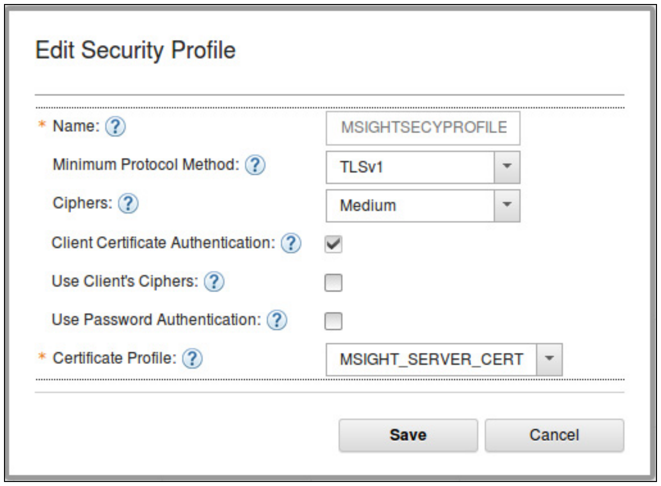
4.3.2 Security profile

After the certificate profile is created, IBM MessageSight knows the names of the certificate and keys to enable secure communication. To provide the rest of the parameters to set up secure communication, a security profile is created for the certificate profile.

Creating a security profile using the web UI

To set up a certificate profile using the web UI, follow these steps:

1. Go to **Appliance** → **Security Settings**.
2. Under Security Profiles, click the green plus (**+**) icon to create a new certificate profile. Figure 4-8 shows the panel for creating a new certificate profile.



Edit Security Profile

* Name:	MSIGHTSECYPROFILE
Minimum Protocol Method:	TLSv1
Ciphers:	Medium
Client Certificate Authentication:	✓
Use Client's Ciphers:	☐
Use Password Authentication:	☐
* Certificate Profile:	MSIGHT_SERVER_CERT

Figure 4-8 Adding a new security profile using the web UI

3. The options in the dialog box allow for various permutations of security parameters. For example, in Figure 4-8, the **Client Certificate Authentication** check box is selected. Therefore, when the communication is encrypted, IBM MessageSight also requests the client to send its certificate for identification.

   Similarly, security can be further enhanced by making the client provide a user name and a password by selecting the **Use Password Authentication** check box. If this box is selected, a valid user name and password must be passed by the client.

   Therefore, for the same certificate profile, multiple security profiles can be created by using different options as set when creating the security profile.

For more information about client authentication based on certificates, see *Client certificate authentication*:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Security/se00007.html

Creating a security profile using the CLI

To create a security profile on the command line, see Example 4-7.

Example 4-7 Creating a security profile

```
Console> imaserver create SecurityProfile "Name=MSIGHTSECYPROFILE"
"UseClientCertificate=True" "UsePasswordAuthentication=False"
"CertificateProfile=MSIGHT_SERVER_CERT"
```

Because `UseClientCertificate` is set to `True`, IBM MessageSight requests the certificate from the client for identification. Similarly, because `UsePasswordAuthentication` is set to `False`, the client does not need to pass the user name and password.

For more information about creating a security profile using the CLI, see *Creating a security profile on the command line*:

https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00337_.html

4.3.3 Associating a security profile with an endpoint

The certificate profile provides a certificate and associated key file, as designated for a certain entity to IBM MessageSight. This entity can be an entire message hub, a given endpoint, or any other configuration that you choose. Using the certificate profile, you created a security profile so that you can set parameters, such as client authentication based on certificates or user name and password.

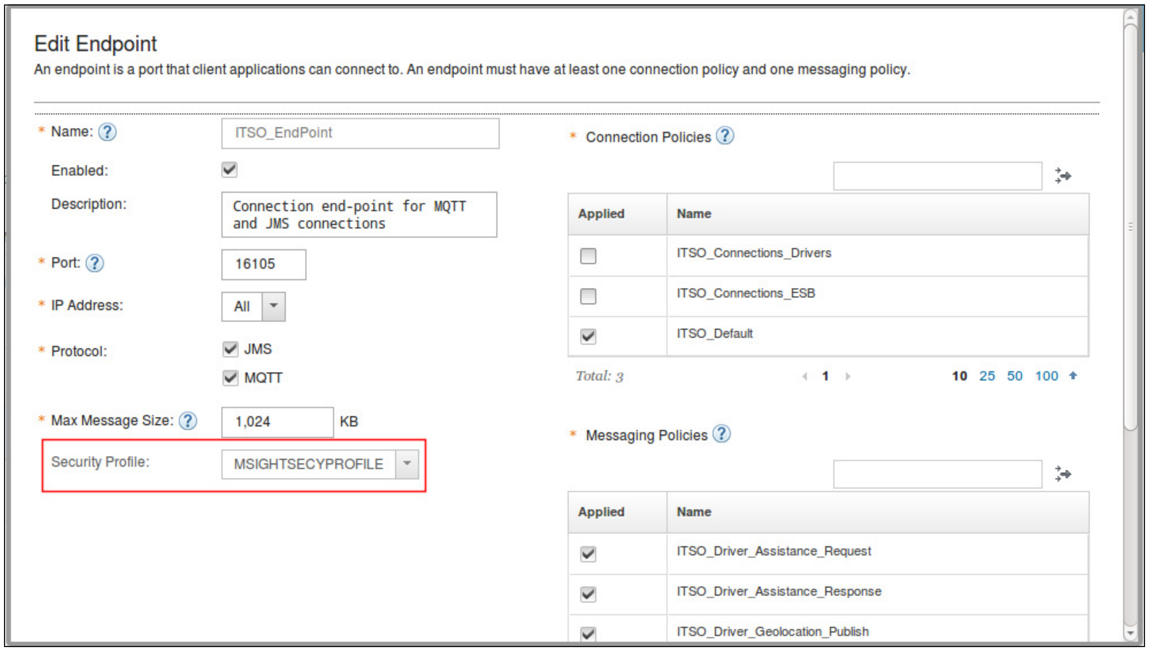
With both these profiles in place, associate the security profile with an endpoint.

Associating the security profile with an endpoint using the web UI

To associate a security policy with an endpoint using the web UI, follow these steps:

1. Go to **Messaging** → **Message Hubs**.
2. From the message hub, select a hub that you want to work with and click the pencil icon to edit it.
3. The resulting page shows a page for connection and messaging policies, along with the endpoints related with the selected hub. Click the endpoint that you want to work with, and click the pencil icon to edit it.

Figure 4-9 shows the panel for editing the endpoint. Note that the red rectangle shows the name of the security profile that this endpoint is associated with.


Edit Endpoint

An endpoint is a port that client applications can connect to. An endpoint must have at least one connection policy and one messaging policy.

* Name:	ITSO_EndPoint
Enabled:	✓
Description:	Connection end-point for MQTT and JMS connections
* Port:	16105
* IP Address:	All
* Protocol:	✓ JMS ✓ MQTT
* Max Message Size:	1,024 KB
Security Profile:	MSIGHTSECYPROFILE

* Connection Policies

Applied	Name
☐	ITSO_Connections_Drivers
☐	ITSO_Connections_ESB
✓	ITSO_Default

Total: 3 1 10 25 50 100

* Messaging Policies

Applied	Name
✓	ITSO_Driver_Assistance_Request
✓	ITSO_Driver_Assistance_Response
✓	ITSO_Driver_Geolocation_Publish

Figure 4-9 Associating a security profile with an endpoint

Associating the security profile using the CLI

Example 4-8 shows how to associate the security profile with an endpoint using the CLI.

Example 4-8 Associate security profile with an endpoint

```
Console>imaserver update Endpoint "Name=ITSO_EndPoint"
"SecurityProfile=MSIGHTSECYPROFILE"
```

4.3.4 Adding trusted certificates

To set up an infrastructure for client authentication through the certificates of the certificate authority (CA), the client certificate must be present as a trusted certificate (see Figure 4-11 on page 95). This is done by adding the certificate of the CA for a given security profile.

When a client connects to an endpoint, the client provides its certificate for authentication. The security profile associated with that specific endpoint contains the certificate of the CA of the client certificate. This setup ensures a successful client authentication.

Adding a trusted certificate using the web UI

To add a trusted certificate using the web UI, follow these steps:

1. Go to **Appliance** → **Security Settings**.
2. Under Security Profiles, click the security profile that you want to work with.
3. Click **Other Actions** → **Trusted Certificates**.
4. Figure 4-10 shows the panel for adding a trusted certificate. Browse the relevant `.pem` file and click **Upload**.

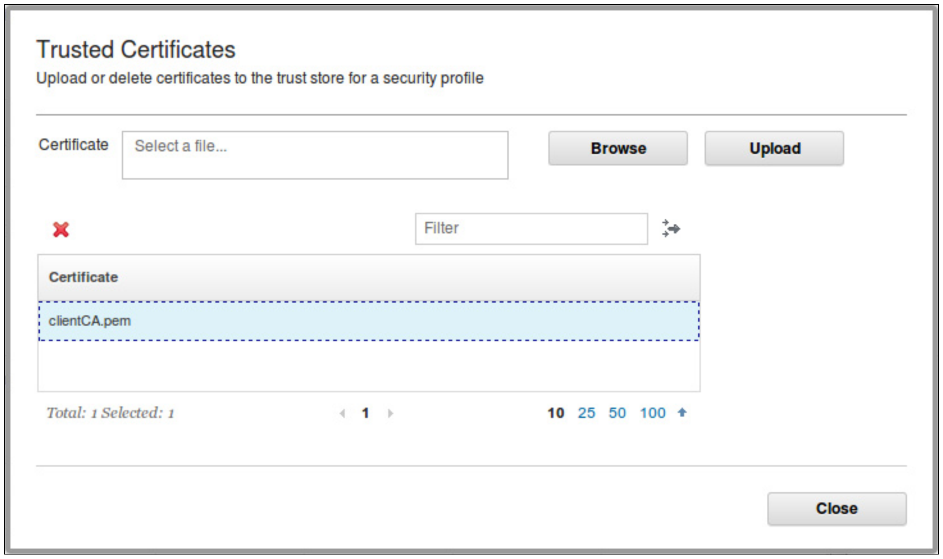


Figure 4-10 Upload the CA certificate as a trusted certificate

Adding a trusted certificate using the CLI

To add a trusted certificate using the CLI, complete the following tasks in order:

1. Import the certificate to the IBM MessageSight appliance.
2. Apply the certificate.

In Example 4-9, you import the certificate and key from a server using the **scp** protocol.

Example 4-9 Importing certificate and key for the appliance

```
Console> file get scp://user@certs.CA.org/servers/clientCA.pem
user@certs.repo.org's password:
The requested configuration change has completed successfully.
```

In Example 4-10, you apply the certificate and key.

Example 4-10 Applying the certificate as a trusted certificate

```
Console> imaserver apply Certificate "TrustedCertificate=clientCA.pem"
"SecurityProfileName=MSIGHTSECYPROFILE"
```

4.4 Combining TLS with connection and messaging policy

With TLS, you can encrypt the communication. Further, you can apply authentication to clients, as authenticated by their certificates. Another level of authentication is to require a client to pass a valid user name and password.

With *connection policy*, you can apply more constraints based on protocols, client identifiers, and so on. With *messaging policy*, you can limit access to only certain types of action and resources.

This section shows you how to make all of these work together.

4.4.1 Overview of TLS, connection policies, and messaging policies

Figure 4-11 shows an overview of the security implemented with TLS, connection policies, and messaging policies.

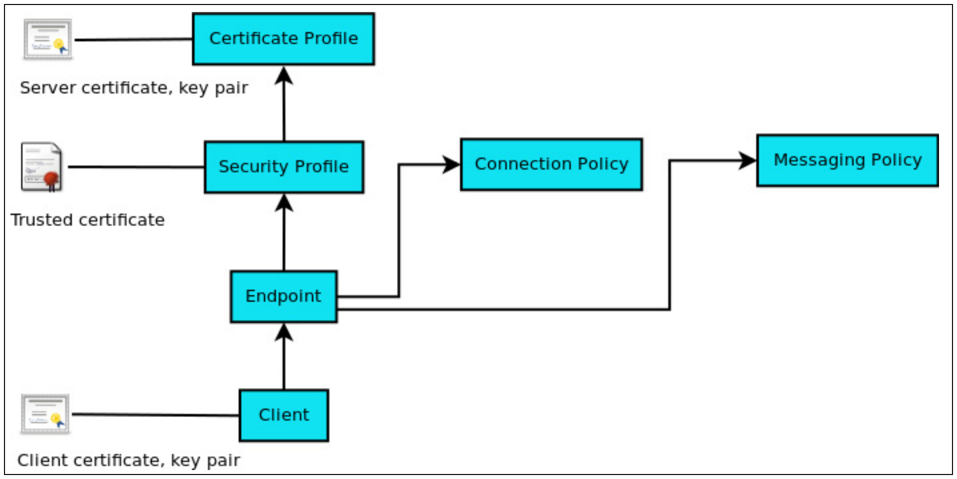


Figure 4-11 Security overview with TLS, connection policies, and messaging policies

In this setup, the security is implemented using the following steps:

1. The endpoint has a security profile associated with it. This security profile has a certificate profile associated with it, where the certificate profile has the server certificate and key pair. The client has its own certificate and key pair. Together, the client and endpoint establish an encrypted channel of communication.
2. If the security profile is set up accordingly, the endpoint sends a request to the client for its certificate for authentication.
3. The client then sends its certificate for authentication. With the trusted certificate set up with the security profile, the endpoint determines whether the client certificate is signed by a trusted CA.
4. After the authenticity of the client is established, the connection policy (or policies) associated with the endpoint comes into effect. With the connection policy, the connection is allowed if the constraints are met successfully. For example, the constraint can be on protocol (JMS or MQTT), IP address, client ID, and so on. The constraint can also be set on the common name of the client certificate.

Important: The *common name* of any digital certificate is a field in the subject of the certificate. This is one of the most important certificate fields, because it plays an important role in identification in a security setup.

5. After clearing the connection policy, the messaging policy associated with the endpoint comes into effect. With the messaging policy, the connection is allowed if the constraints are met successfully. For example, the constraint can be on an action (send, receive, publish, or subscribe), topic string, queue name, and so on. The constraint can also be set on the common name of the client certificate.

4.4.2 Identities in a security setup with TLS, connection policies, and messaging policies for an MQTT client

Figure 4-12 shows a sample implementation of a security setup for an MQTT client. In this sample, various identities are used in the security setup.

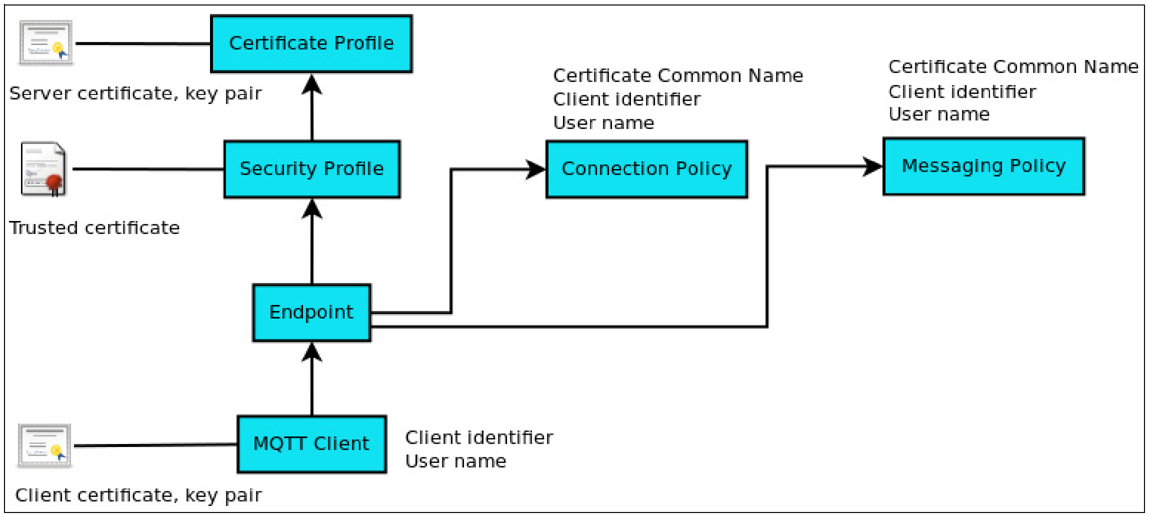


Figure 4-12 Identities in a security setup for an MQTT client with TLS, connection policies, and messaging policies

Identities in the security step

To secure any topology, it is important to know the entities and their identities involved. Table 4-1 on page 97 lists the entities and their identities for the security setup shown in Figure 4-12.

Table 4-1 Entities and identities for security setup

Entity	Identity	Remarks
MQTT client	► Client ID ► User name and password ► Certificate common name	Per the MQTT protocol specification, the client ID is mandatory, and the user name and password are optional.
Endpoint	► IP address ► Port number ► Name of the endpoint	The endpoint is identified by its IP address and port number. In IBM MessageSight, the endpoint also has a name.
Appliance	► Messaging user and group	The messaging user and group are used in the connection and messaging policies to apply a constraint on the connecting client.

Table 4-2 lists the certificates used for the security setup in Figure 4-12 on page 96.

Table 4-2 Certificate used in the security setup

Entity	Certificates	Usage
MQTT client	► Certificate and key pair	The MQTT client uses the pair of certificate and key pair for encrypting the communication, and also for authenticating itself to the endpoint.
Endpoint	► Certificate and key pair ► Trusted certificate	The endpoint uses the pair of certificate and key pair for encrypting the communication. The endpoint uses the trusted certificate to ensure that the client certificate was signed by a trusted authority.

The endpoint uses the trusted certificate to ensure that the client certificate was signed by a trusted authority.

The connection and messaging policies are created with respect to the following identities:

- Client ID of the MQTT client
- User name and password of the MQTT client
- Certificate common name of the client certificate

5

MQTT with mobile platforms

The number of different types of devices that can benefit from MQTT implementations is growing everyday. This chapter provides information about the available technologies to develop MQTT mobile applications.

This chapter describes how to develop a hybrid mobile application using IBM Worklight Studio for the Android platform. It contains an example application to show you how to send requests, receive responses, and push notifications on wanted quality of service (QoS) using MQTT.

This chapter includes the following topics:

5.1 Mobile application development considerations

Mobile communication devices are the fastest growing application segment. The traditional mobile phone has evolved into a powerful computing device, and new platforms and mobile operating systems are also now available.

Despite the wide variety of available technological capabilities, architects and developers must recognize the following factors when considering building applications for mobile devices:

- Preservation of battery life

  Mobile communication devices are designed, as the name implies, to be mobile. Therefore, as a general rule, the time interval between charges must be as long as possible.

  Independent of the device's platform or battery technology, running applications use resources (processor, memory, and network connectivity) and use battery power. Even when application management is generally a platform's choice, design applications to use the fewest resources possible.

- Processing capacity

  Even when mobile devices have a relatively good processing capacity, make applications as simple as possible in terms of processing, to speed up response time and to help preserve battery life.

- Network optimization

  Mobile devices must have at least one network connectivity option. In fact, most devices have a choice between several connection possibilities that vary in speed, connectivity range, battery consumption, and cost to the user. As a general rule, most types of connectivity are at some point limited in range.

  Mobile applications can switch between available communication options seamlessly, based on a set of user-defined preferences. By doing this, an application can use a pre-configured IEEE 802.11x (WiFi) connection when available, and then switch to a 3G or 4G cellular connection when the WiFi is out of range. Design the application to continue its functionality in a transparent manner.

- Platform capabilities

  Base requirements to build applications can vary from one platform to another. Usually, the platform owner provides a set of application programming interfaces (APIs) to develop applications for their platform. Sometimes, as with MQTT, freely available implementations of the protocol allow the developer to concentrate on the application functionality, and use the MQTT protocol as a service.

5.2 Mobile application development models

Several approaches for developing mobile applications exist. Figure 5-1 illustrates the capabilities of the four most important approaches.

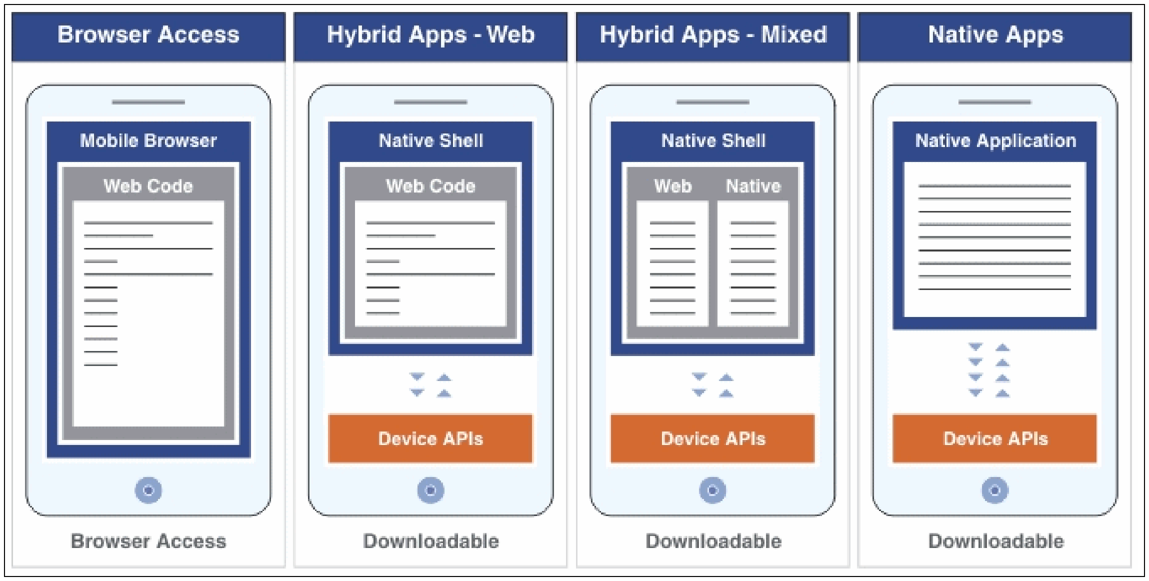


Figure 5-1 Mobile application development models

Mobile applications can be developed using any of the four different approaches:

- Web applications

  They are written entirely in HTML5, Cascading Style Sheets (CSS), and JavaScript code. Web applications are run by the mobile browser, and are cross-platform by default.

- Hybrid applications (web)

  The source code of the application consists of web code run within a native container, and consists of native libraries.

- Hybrid applications (mixed)

  The developer augments the web code with native language to create unique features, and access native APIs that are not available in JavaScript.

- Native applications

  This type of application is platform-specific, and requires expertise and knowledge of the platform.

5.2.1 Web application (browser access) development

If you use the web application development approach, your mobile application runs inside the browser of the mobile device, and uses standard technologies, such as HTML5, CSS3, and JavaScript. Mobile web applications are platform-independent, and no redevelopment is required to support a new mobile platform. However, modifications in the application might be required to support a different browser engine.

Mobile web applications cannot access the platform functions, because they rely only on the browser and the associated web standards. Mobile web applications are not distributed through an application store. They are accessed through a link on a website, or a bookmark in the mobile browser of the user.

Figure 5-2 illustrates an example of a web application (`http://m.ibm.com/us/en/`) opened through a Google Chrome browser.

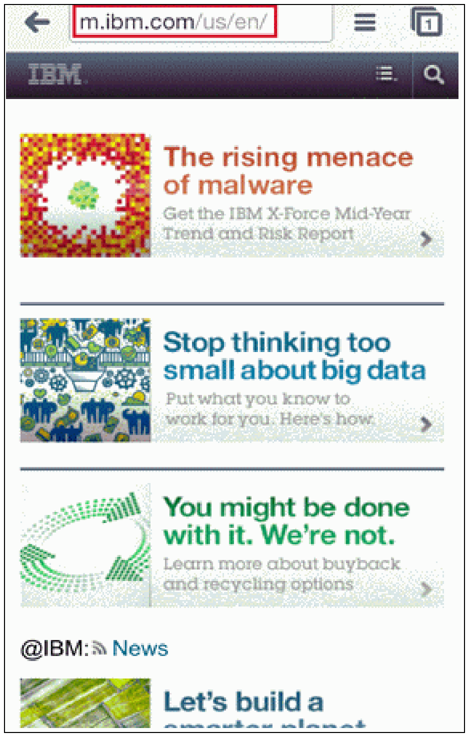

m.ibm.com/us/en/
IBM
The rising menace of malware
Get the IBM X-Force Mid-Year Trend and Risk Report
Stop thinking too small about big data
Put what you know to work for you. Here's how.
You might be done with it. We're not.
Learn more about buyback and recycling options
@IBM: News
Let's build a

Figure 5-2 Example of a web application

5.2.2 Hybrid application (web) development

Mobile applications implemented with the hybrid development approach use parts of both the native development and web development approaches. Mobile hybrid applications run inside a native container, and uses the browser engine to display the application interface, which is based on HTML and JavaScript.

With the native container, your mobile application can access device capabilities that are not accessible to web applications, such as the accelerometer, camera, and local storage on a smartphone. Similar to native applications, hybrid applications are distributed through the platform application store.

5.2.3 Hybrid application (mixed) development

Mobile applications developed using the hybrid mixed development approach use a container to access device capabilities, but also use other native, platform-specific components, such as libraries or specific user-interface elements, to enhance the mobile application.

5.2.4 Native development

Mobile applications developed using the native development approach are developed for a specific platform, and run on that platform only. These mobile applications can fully use all of the platform functions, such as accessing the camera or contact list, or interacting with other applications on the device.

To support different platforms, such as Android, iOS, BlackBerry, and Windows Phone, application developers must develop separate applications with different programming languages, such as Objective-C for iOS, or Java for Android. Typically, native applications are distributed through an application store.

5.2.5 Comparing the different approaches

Each of these development approaches has advantages and disadvantages. The specific requirements for an individual mobile solution drive the selection of an appropriate development approach.

In addition to the design and development of mobile applications, other topics are important also. Depending on the architecture of the application, running certain parts of the business logic as back-end services might be possible. These might have to be developed from the beginning (“from scratch”), but existing services typically can be used or adapted.

These services must be integrated into the mobile solution, and developing all of these components to create the solution involves various skills, not just mobile application development. Table 5-1 provides a comparison of different development approaches.

Table 5-1 Development approaches

Feature	Native application	Hybrid application	Web application
Development language	Native only	Native and web or web only	Web only
Code portability and optimization	None	High	High
Access device-specific features	High	Medium	Low
Use existing knowledge	Low	High	High
Advanced graphics	High	Medium	Medium
Upgrade flexibility	Low (Always by way of application store)	Medium (Usually by way of application store)	High
Installation experience	High (from application store)	High (from application store)	Medium (by way of mobile browser)

Another challenge in developing mobile applications is testing. If the application is intended for multiple platforms, it must be tested on all of them. For each platform, developers might have to test different devices with different functionality and display sizes.

In addition, to run end-to-end tests, the devices must have access to a test environment. This poses a problem, because in traditional web development, a test environment is usually protected behind firewalls and not connected to the Internet.

5.3 IBM Worklight

IBM Worklight is a mobile application platform containing all of the tools required to develop a mobile application, including implementation, testing, and distribution. IBM Worklight facilitates the creation and delivery of rich, secure mobile applications. It simplifies end-to-end security and service integration between mobile applications and back-end systems.

It makes cross-platform application development easier through the use of standards-based technologies, such as HTML5 or JavaScript, and tools that can access native device capabilities. IBM Worklight consists of five core components, as illustrated in Figure 5-3.

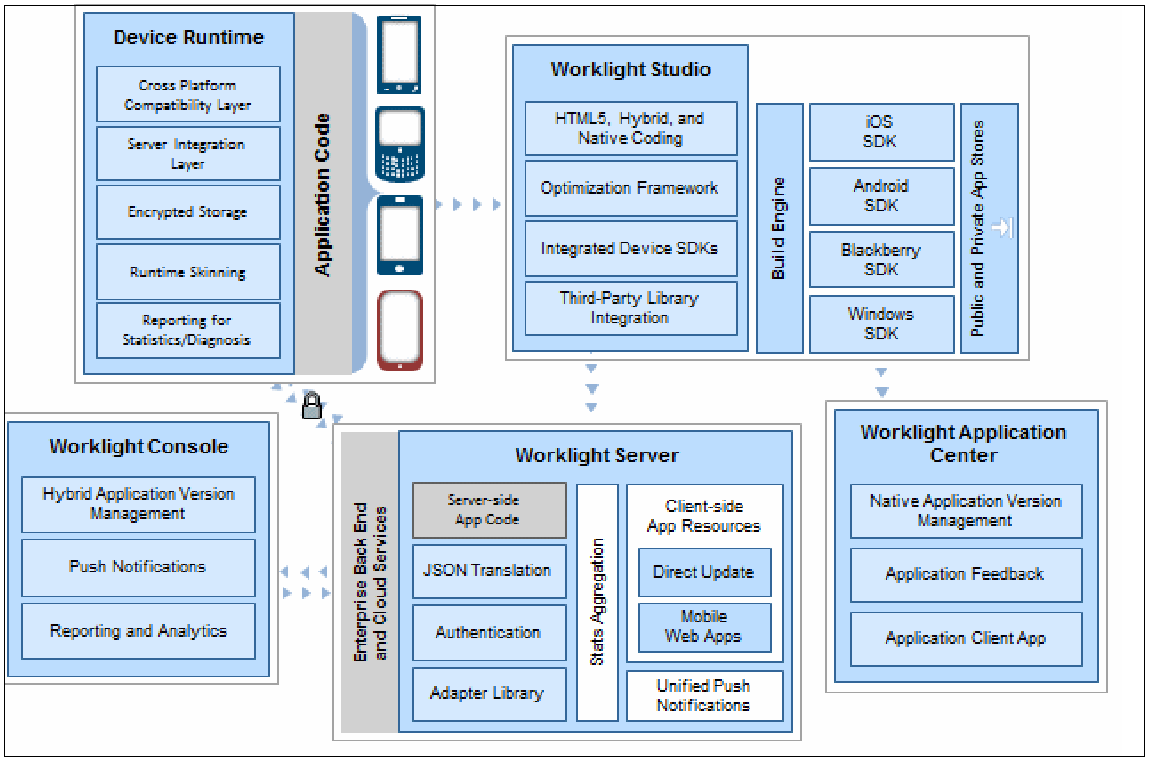


Figure 5-3 Five components of IBM Worklight

Each component provides a specific set of functions and supports a different stage in the lifecycle of a mobile application:

Worklight Studio	An Eclipse-based development environment with a rich set of tools for developing cross-platform applications.
Worklight Device Runtime Components	A runtime environment with consistent APIs that can work across different mobile platforms to access native device functionality, and integrate with existing services and security.
Worklight Server	A runtime server that enables secure access to enterprise services, application version management, unified push notifications, and direct application updates.
Worklight Console	A web-based user interface for real-time analytic, push notification authoring, and mobile application version management.
Worklight Application Center	A private, cross-platform mobile application store that is tailored for the specific needs of an application development team.

Note: Android applications developed in this book use Worklight Studio and Worklight Device Runtime Components. More details about application development are included in section 5.8, “Developing an MQTT hybrid application for Android” on page 143.

5.3.1 IBM Worklight Studio

Worklight Studio is an Eclipse-based integrated development environment (IDE) that provides a set of tools to create a fully operational mobile application for various mobile operating systems, and to integrate applications with existing services. By enhancing Eclipse with custom plug-ins, Worklight enables developers to use a single IDE to build enterprise applications, from server applications to applications for different mobile device operating systems.

Figure 5-4 illustrates an overview of IBM Worklight Studio.

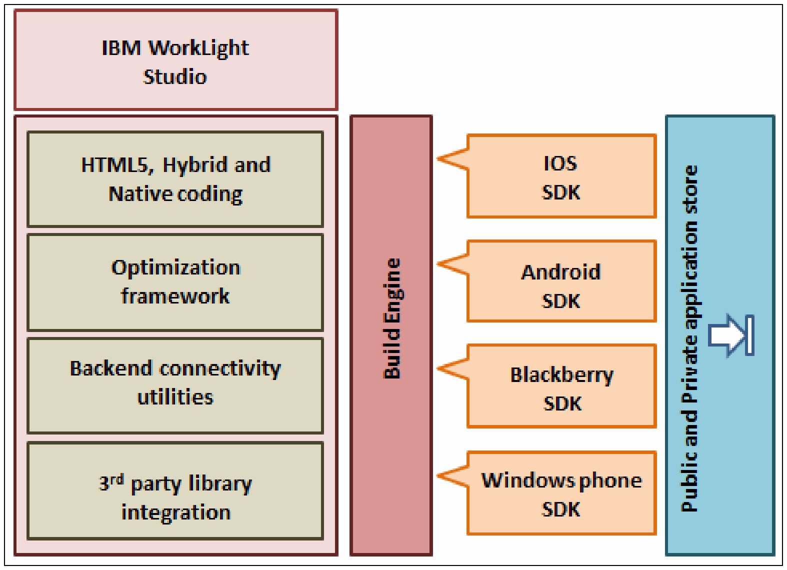

IBM WorkLight Studio
HTML5, Hybrid and Native coding
Optimization framework
Backend connectivity utilities
3rd party library integration
Build Engine
IOS SDK
Android SDK
Blackberry SDK
Windows phone SDK
Public and Private application store

Figure 5-4 IBM Worklight Studio overview

Worklight Studio provides powerful capabilities mobile applications developers. The following list includes a few of these capabilities:

- Pure native development and web development technologies

  Worklight Studio supports pure native development and web development technologies, such as HTML5, Apache Cordova, and Java. Developers can develop mobile applications with pure HTML5, or use a compatible JavaScript framework, such as jQuery Mobile, Dojo, or Sencha Touch. This simplifies development with the reusable user interface widgets and commonly used functions that are provided by the frameworks.

  Developers can use Apache Cordova to enable mobile application to access native device functionality. Also, if the application needs to access a special device module, such as one for near field communication (NFC), developers can develop a native extension that can be made available to JavaScript through an Apache Cordova plug-in (a small native-to-JavaScript wrapper).

- Shell development

  For hybrid mobile applications, Worklight uses a default hybrid shell that offers all of the capabilities necessary to use web and native technologies. Shell development enables organizations to separate native component implementations from web-based implementations, and to split this work between different developers.

For instance, by creating a custom shell, developers can add third-party native libraries, implement custom security, or provide extended features specific to the organization. Shells can also be used to restrict or enforce specific corporate guidelines, such as design or security rules. For example, a shell can be used to add a default style to the mobile application, or to disable the device's camera.

- Optimization framework

  With IBM Worklight Studio, a common environment is used as a central development point, and all environments can share the same base code. Then, to create an optimized version for an Apple iPad tablet, for instance, developers can create a variant of the base and implement only the required changes. An optimization framework called *runtime skinning* enables mobile applications to switch at run time between different sets of customization.

- Integration of device-specific software development kits (SDKs)

  Each vendor of mobile devices supplies its own development environment as part of an SDK. Worklight Studio generates a project for each supported SDK (for example, Xcode for iOS development). Some vendors require developers to use their SDK for specific tasks, such as building the binary application.

  The integration of device-specific SDK with Worklight Studio links the Worklight Studio project with the native development environment (such as Xcode), enabling the developer to seamlessly switch between a native development environment and Worklight Studio. Any change in the native development environment is mirrored back to the developer's Worklight Studio project, reducing the need for manual copying.

- Third-party library integration

  Depending on the developer's programming approach, these mobile applications can include several JavaScript frameworks, such as Sencha Touch, jQuery, or Dojo. This third-party library integration promotes code reuse and reduces implementation times.

  Many companies underestimate the effort involved in developing a framework, because testing efforts for quality assurance are not fully taken into account. In the case of a shell project, different kinds of compatible native code or libraries can be included.

- Integrated build engine

  The build chain of Worklight Studio combines common implementation code (used on all target platforms) with platform-unique code that is used on specific target platforms. At build-time, the integrated build engine of Worklight Studio combines these two implementations into a complete mobile application. This reduces IT development costs by using a single, common implementation for as much of the mobile application as possible, rather than a unique implementation for every supported platform.

- Integrated development tools

  By extending the Eclipse IDE with custom plug-ins, Worklight Studio can be used to develop all components of a mobile application, including the mobile application and integration modules (called Worklight *adapters*) from within the same development environment. Integrated development tools allow developers to develop and test these adapters seamlessly in Worklight Studio.

- Mobile browser simulator

  IBM Worklight Studio offers a mobile browser simulator, which can be used during the development cycle to test mobile web and hybrid applications using a desktop browser. Many of today's desktop and mobile browsers are based on the WebKit engine as the underlying core technology, providing a common platform for developing applications that support HTML5, CSS3, and JavaScript.

  This consistent browser engine enables developers who host their mobile browser simulator on a WebKit-based desktop browser, such as Chrome or Safari, to validate the application behavior before deploying it on the actual device.

  When the developer is ready to test on the actual device or mobile emulator, they can verify that the core WebKit engine resident on many mobile devices, including Android and iOS, provides the same consistent user experience that was verified when testing using the browser.

  In addition to supporting cross-platform browser testing for mobile devices, the mobile browser simulator also provides implementations of the various Apache Cordova APIs. With these APIs, users can test hybrid applications that use device features without having to run on the actual device, decreasing development time and effort required to repeatedly deploy the application to a device.

Worklight studio is based on Eclipse, so it gives developers all of the flexibility and extensibility that Eclipse provides, such as adding new functionality with plug-ins. For instance, an IBM Rational® Team Concert plug-in can be used to control application source code, track changes, and create daily builds without installing an additional development application.

Worklight Studio also provides a set of Ant tasks that can be used to run a mobile application build for various platforms. Developers can distribute build tasks to a variety of build machines running Apple OS X (for an Apple iOS binary) or Microsoft Windows (for a Microsoft Windows Phone 8 binary), among others.

Using this mechanism can reduce IT resource requirements, because developers no longer need access to multiple build machines to create binaries for specific mobile platforms.

5.3.2 IBM Worklight Device Runtime Components

Worklight Device Runtime Components ensure that the mobile applications have access to a variety of Worklight features during run time. Its components are on the mobile device, and consist of libraries that are bundled into the application to enable additional runtime capabilities.

These libraries can help integrate mobile applications with Worklight Server using predefined communication interfaces, and simplify application development by providing unified access to native device functionality.

The native, hybrid, mixed hybrid, or web-based APIs in Worklight Device Runtime Components provide support for all mobile development approaches, with enhanced security and integration features. For hybrid and mixed hybrid applications, the included Apache Cordova plug-ins add native capabilities and cross-platform user interface controls.

Worklight Device Runtime Components deliver a uniform bridge between web technologies (HTML5, CSS3, and JavaScript) and the native functions available on various mobile platforms. Figure 5-5 illustrates how the architecture of Worklight Device Runtime Components combines native and web technology to drive mobile application development.

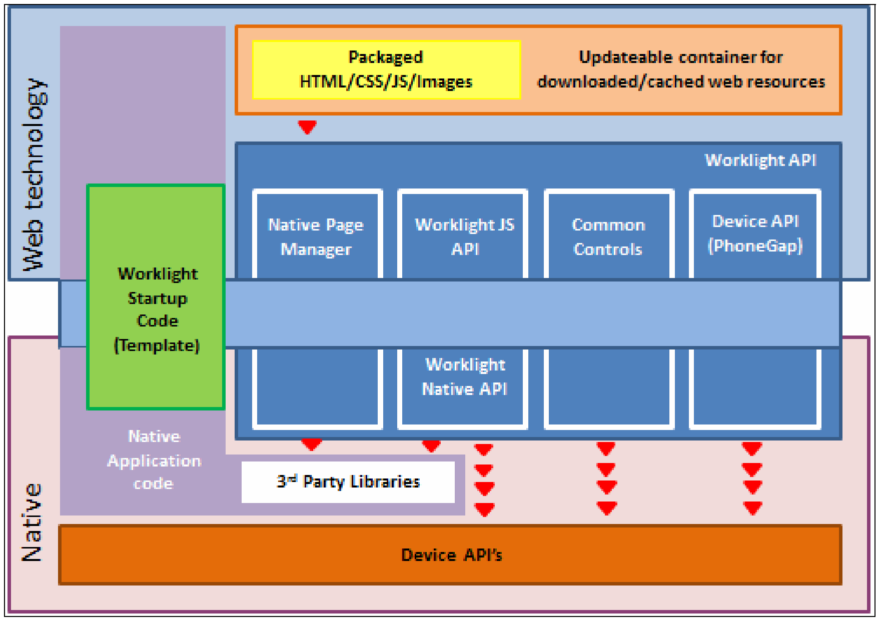


Figure 5-5 Worklight Device Runtime Components: Architecture overview

Worklight Device Runtime Components support a wide variety of mobile operating systems and release levels (some are only supported at certain fix pack levels). See the most recent list of supported mobile operating systems and release levels at the following website:

http://publib.boulder.ibm.com/infocenter/prodguid/v1r0/clarity-reports/report/html/softwareReqsForProductByComponent?deliverableId=1343665214557&duComponent=Mobile%20Device_CBF56230C6611E28BB5885E0925FE36

In addition to Apache Cordova, Worklight Device Runtime Components provide several important features that can improve application development by reducing complexity and implementation time:

- Cross-platform compatibility layer

  A cross-platform compatibility layer supports development for all supported platforms. Developers of hybrid mobile applications can access common control elements, such as tab bars and clipboards, and native device capabilities, such as the location service or the camera. These functions can be extended for Android and iOS using a custom shell.

- Client-to-server integration

  This feature enables transparent communication between a mobile application (built using Worklight technology) and Worklight Server. Worklight forces mobile applications to use a Secure Sockets Layer (SSL)-enabled connection to the server at all times, including for authentication. This integration enables advanced management and security features, such as remotely disabling a specific application version, or updating the web resources of a hybrid application.

- Encrypted data store

  Mobile devices are inherently insecure, so malicious application users can sometimes get access to private application data or other information. To help prevent this access, an application can store private data in an encrypted data store on the device, and can access that data using a simple API. Using an encrypted data store prevents malicious users from reading the information; all they get is highly encrypted data.

  Using International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 18033-3 security standards, such as Advanced Encryption Standard (AES) 256 or Public Key Cryptography Standards (PKCS) #5, the encryption complies with the United States National Security Agency (NSA) regulations for transmitting confidential or secret information.

The key that is used to encrypt the information is unique to the current user of the application and the device. Worklight Server issues a special key when a new encrypted data store is created. That makes it almost impossible to reconstruct the key used to encrypt the critical information.

- JavaScript Object Notation (JSON) Store

  To synchronize mobile application data with related data on the back end, Worklight includes a JSON Store that provides a synchronizable, offline-capable, key-value database. It implements the application's local read, write, update, and delete operations, and can use Worklight adapter technology to synchronize the related back-end data.

  Imagine a sales representative who is making a sales call in an area without reliable network connectivity. The representative can download all customer information, sales documents, and marketing presentations before leaving the office. Then, when the deal is complete and network connectivity is re-obtained, the back-end systems can be updated with all changes related to the new sale.

- Runtime skinning

  This feature helps developers incorporate an adaptive design that can be tailored to each mobile device. The Worklight runtime skin is a user interface (UI) variant that can be applied during application run time based on device properties, such as the operating system, screen resolution, and form factor. This type of UI abstraction enables mobile application development for multiple mobile device models at the same time.

- Reporting

  Worklight Device Runtime Components have a reporting feature for application usage and user behavior. The mobile application sends events to Worklight Server, which records the information in a separate database where it can be used by Worklight Console or an external analytic tool, such as IBM Tealeaf® CX Mobile. This enables developers to study which features of the mobile application are being used, and how they are being used.

- Worklight APIs

  These APIs provide access to Worklight functions across multiple device platforms. Applications built using web technologies can access the Worklight Server through these APIs using JavaScript, but applications that use native components can access the APIs directly using Java and Objective-C.

  This enables mobile applications developed with the hybrid and native development approaches, including those running Android, iOS, or Java ME, to benefit from the simplified application security and integration features of Worklight.

5.4 Mobile technologies

This section of the chapter describes various mobile technologies that are referred to when building mobile hybrid applications for MQTT.

5.4.1 Android software stack

The Open Handset Alliance (OHA) that developed Android consists of well-known companies, such as Google, Motorola, Intel, and so on. Android is a software stack for mobile devices. It includes an operating system, middleware, and key applications.

The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform, using the Java programming language. These include a device emulator, tools for debugging, memory and performance profiling, and a plug-in for the Eclipse development environment.

Figure 5-6 illustrates the major components of the Android operating system within the application framework, libraries, Android run time, and Linux kernel.

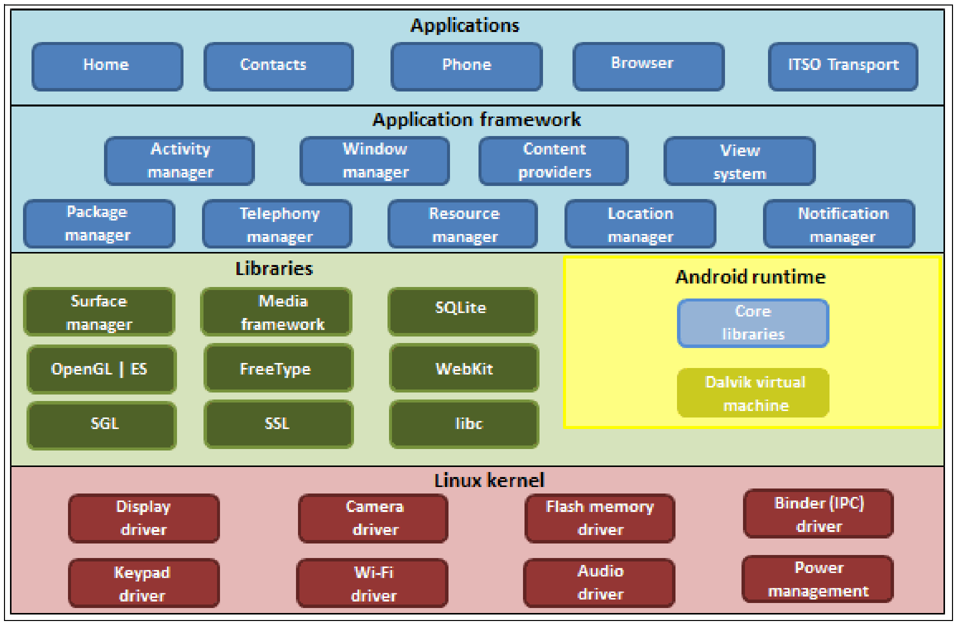


Figure 5-6 Android software stack

5.4.2 Device API specification

Device APIs enable you to develop mobile applications and widgets that interact with device services, such as calendar, contacts, and camera. For more details about the device APIs specification, see the w3c specification website:

`http://www.w3.org/2009/dap/`

5.4.3 Apache Cordova

Web technologies based on JavaScript are the distinct candidates for the web side of a typical mobile application, and are platform-independent by their nature. Unfortunately, their scope is confined within the web part of the application.

When dealing with the native functionality running on the device platform (Camera, Internal Contacts List, and so on), JavaScript does not prove to be helpful. When the need arises for a web page to obtain or run native functionality, the following alternatives are available:

- Implement the scenario in the platform-specific manner (using Android or iOS, for example).
- Use a bridge between web pages and the native page, where requests and responses to and from web and native pages are translated by both sides of the bridge.

Choosing the first option, you can create platform-dependent applications. This is a poor choice for enterprise mobile solutions, because the need to implement mobile applications on different platforms leads to cost-ineffective solutions, which are also poorly extensible for future needs. Using the second option, you benefit from overall control on the web-native communication flow, but still keep the solution platform-independent.

Apache Cordova is an open source mobile development framework that enables the creation of cross-platform mobile applications to access native device features over a uniform API using web technologies (HTML5, CSS, and JavaScript).

Targeting multiple mobile platforms and a variety of form factors in a cost-effective manner is a major consideration when developing a mobile platform. To help achieve this capability, Apache Cordova framework comes embedded in Worklight.

The following list includes some advantages of the Apache Cordova Approach:

- An open, standards-based approach to cross-platform application development
- Cost-effective development accessing existing in-house knowledge and skills
- Compatibility with JavaScript toolkits, such as JQuery Mobile, Sencha Touch, and Dojo Mobile
- Access to native device features using a uniform JavaScript bridge
- An extensible architecture that simplifies integration with native device code

For more details about Apache Cordova, see the following website:

`http://cordova.apache.org/`

5.4.4 HTML5

HTML5 is a language designed to organize web content. It is intended to make web design and development easier by creating a standardized and intuitive user interface markup language. HTML5 provides the means to analyze and compartmentalize your pages. It enables you to create discrete components that are designed to organize your site logically, and created to give your site syndication capabilities.

HTML5 might be called the *information mapping approach to website design*, because it incorporates the essence of information mapping, dividing and labeling information to make it easy to use and understand. This is the foundation of HTML5's dramatic semantic and aesthetic utility. HTML5 gives designers and developers of all levels the ability to publish everything from simple text content to rich, interactive multimedia to the internet.

HTML5 provides effective data management, drawing, video, and audio tools. It facilitates the development of cross-browser applications for the web and portable devices. HTML5 is one of the technologies driving the advances in mobile cloud computing services, because it provides for greater flexibility, enabling the development of exciting and interactive websites.

It also introduces new tags and enhancements, including an elegant structure, form controls, APIs, multimedia, database support, and significantly faster processing speed.

The new tags in HTML5 are highly evocative, encapsulating their role and use. Past versions of HTML used rather nondescript tags. However, HTML5 has highly descriptive, intuitive labels. It provides rich content labels that immediately identify the content.

For example, the overworked <div> tag has been supplemented by the <section> and <article> tags. The addition of the <video>, <audio>, <canvas>, and <figure> tags also provides a more precise description of the specific type of content.

HTML5 provides the following elements and capabilities:

- Tags that describe exactly what they are designed to contain
- Enhanced network communications
- Greatly improved general storage
- Web Workers for running background processes
- The WebSocket interface to establish continuous connection between the resident application and the server
- Better retrieval of stored data
- Improved page saving and loading speeds
- Support for CSS3 to manage the GUI, which means that HTML5 can be content-oriented
- Improved browser form handling
- An SQL-based database API that supports client-side, local storage
- Canvas and video, for adding graphics and video without installing third-party plug-ins
- The Geolocation API specification, which uses smartphone location capabilities to incorporate mobile cloud services and applications
- Enhanced forms that reduce the need to download JavaScript code, allowing more efficient communication between mobile devices and cloud servers

For more details about HTML5, see the following website:

http://www.ibm.com/developerworks/training/kp/wa-kp-html5/

Web storage

Cookies have been around from the inception of JavaScript, so storing data on the web is not a new concept. However, web storage is a much more powerful version of data storage that offers more security, speed, and ease of use. Users can also store large amounts of data in web storage. The exact amount is based on the web browser.

Another advantage of using web storage is that this data is not loaded with every server request. The only limitation is that you cannot share web storage between browsers.

There are two types of web storage objects built into HTML5:

- The `sessionStorage` object stores data for a single session. If the user closes the page or browser, the data is deleted.
- The `localStorage` object stores data with no expiration date. The data remains stored when the web page or browser is closed, depending on the storage amount set for the user's browser.

Browsers that support web storage have the global variables <*sessionStorage*> and <*localStorage*> declared at the window level. Example 5-1 and Example 5-2 illustrate JavaScript examples for <*sessionStorage*> and <*localStorage*>.

Example 5-1 SessionStorage JavaScript snippet

```
// Store value on browser for duration of the session
sessionStorage.setItem('key', 'value');
```

Example 5-2 LocalStorage JavaScript snippet

```
// Store value on the browser beyond the duration of the session
localStorage.setItem('key', 'value');
```

WebSockets

WebSockets provides full-duplex communication over a single TCP socket. WebSockets is a component of HTML5, designed to replace existing push-based technologies with the aim of reducing latency and network traffic between client and server.

The protocol operates in two parts:

1. First, a handshake is exchanged between client and server (as an HTTP Upgrade request).
2. Next, data is transferred back and forth in messages that are composed of one or more frames, each having a specified type (for example, UTF-8 text, binary data, or control frames) that is the same for each frame in a message.

The WebSockets protocol is designed on the principle that there ought to be minimal framing (the only framing that exists is to make the protocol frame-based rather than stream-based, and to support a distinction between Unicode text and binary frames). It is expected that metadata will be layered on top of WebSockets by the application layer, in the same way that metadata is layered on top of TCP by the application layer (for example, HTTP).

Conceptually, WebSockets is a layer on top of TCP that performs the following functions:

- Adds a web origin-based security model for browsers
- Adds an addressing and protocol naming mechanism to support multiple services on one port, and multiple host names on one IP address
- Layers a framing mechanism on top of TCP to get back to the IP packet mechanism that TCP is built on, but without length limits
- Includes an extra closing handshake in-band that is designed to work in the presence of proxies and other intermediaries

WebSockets is ideal for the following types of applications:

- Gaming
- Stock ticker
- Traffic updates
- Telemetry data (MQTT over WebSocket)
- Mobile payments
- Mobile application gateway
- Scoring

5.4.5 Eclipse Paho

The Eclipse Paho project was developed to provide scalable open source implementations of open and standard messaging protocols for various emerging applications in the machine-to-machine (m2m) and Internet of Things (IoT) spaces. One of the major objectives of the Paho project is to provide an effective level of decoupling between devices and applications.

Paho initially started with MQTT publish/subscribe client implementations for use on embedded platforms, but in the future will provide corresponding server support as determined by the community. The current efforts of the Paho project are to support the requirements of m2m integration with web and enterprise middleware and applications.

In order for m2m devices and client developers to integrate, develop, and test messaging components end to end, Paho provides a framework to support testing and development of end to end device connectivity with a server. The MQTT application developed in this chapter uses the Paho Java client implementation.

You can download these client implementation source files from the following website:

`http://git.eclipse.org/c/paho/org.eclipse.paho.mqtt.java.git`

For more details about the Eclipse Paho project, see the following website:

http://www.eclipse.org/paho

5.4.6 SQLite database

SQLite is an open source database. SQLite supports standard relational database features, such as SQL syntax, transactions, and prepared statements. The database requires limited memory at run time, which makes it a good candidate for being embedded into other lightweight runtime engines.

SQLite does not support static data typing, but instead uses column affinity. This means that rather than a data type being a property of a table column, it is a property of the data. When a value is inserted into the database, SQLite examines its type.

If the type does not match that of the associated column, an attempt is made to convert the value to the column type. If this is not possible, the value is stored as its native type. SQLite supports the NULL, INTEGER, REAL, TEXT, and BLOB data types.

SQLite is embedded into every Android device. Using an SQLite database in Android does not require a setup procedure, or administration of the database. Developers only have to define the SQL statements for creating and updating the database. Later, the database is automatically managed by the Android platform.

Access to an SQLite database involves accessing the file system. This can be slow. Therefore, performing database operations asynchronously is advisable. If the Android application creates a database, this database is by default saved in the `data/data/APP_NAME/databases/FILENAME` directory.

To read more details about SQLite database, see the following website:

http://www.sqlite.org

5.5 MQTT hybrid application for Android using IBM Worklight

This section describes how Worklight Studio can be used to develop a hybrid Android application that sends and receives messages using MQTT.

The approach used is to develop the application in HTML and JavaScript, and to use the Apache Cordova (PhoneGap) libraries provided with Worklight to interface with native code that implements the MQTT protocol. This native code includes an Android Service that manages communications with MQTT when the application is not running in the foreground.

Figure 5-7 illustrates the architecture of the mobile application developed in this chapter. This application is developed as an Worklight hybrid application. User interface and application logic is coded in HTML, CSS, and JavaScript.

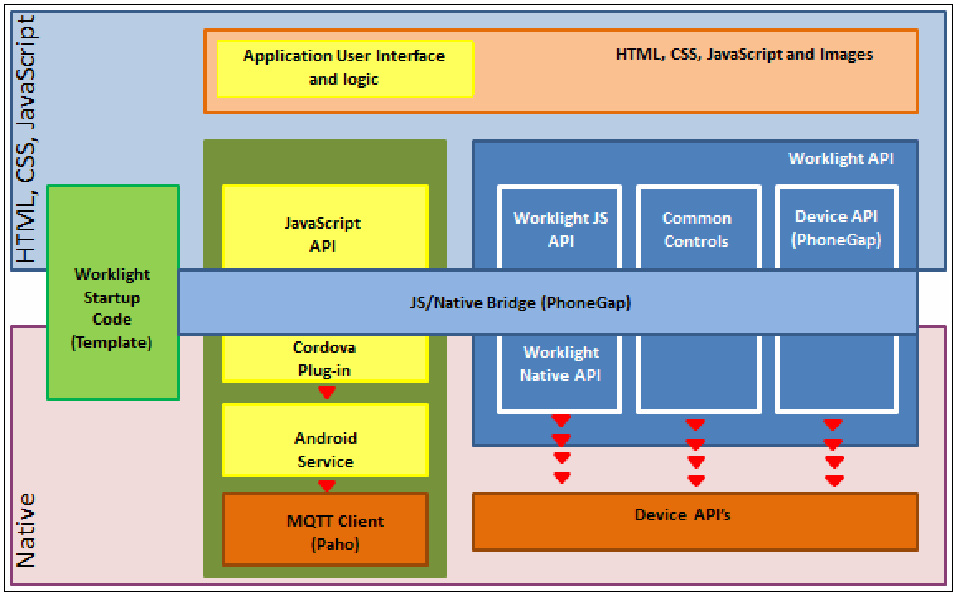


Figure 5-7 Architecture of hybrid mobile application

The Worklight Client API enables the mobile application JavaScript code to use Android platform-specific user interface controls, and to access device-specific hardware. The Worklight API uses Apache open source Cordova project (Also known as PhoneGap) to bridge JavaScript to the native Java code required to interface with the Android platform and device hardware.

The same approach has been extended to send and receive messages using MQTT. This way, mobile application programmers can develop their application logic and user interface using regular JavaScript, but the detailed code to handle communications with the messaging service is implemented in native Java code.

To handle the specifics of MQTT protocol, this mobile application uses Java MQTT client implementation from the Eclipse Paho project. This is a general-purpose Java library. It is wrapped with an Android Service to adapt it for use in the Android platform.

One of the major benefits this approach provides is to enable incoming messages to be handled when the application is not running in the foreground. The Cordova plug-in and JavaScript API components serve as the interface between the Android Service and the JavaScript application logic.

Note: It is advisable to keep all mobile application-specific logic in the JavaScript portion of the application, rather than developing it in the Cordova plug-in or Android Service components. This approach enables the native and platform-specific components to be reused for other mobile applications.

The integration between MQTT and Worklight is performed in the *Device Runtime* level. Because Worklight uses the Apache Cordova (PhoneGap) framework to provide cross-platform run time for supported mobile devices, a PhoneGap plug-in must be developed for supporting MQTT.

For more details about developing Apache Cordova plug-ins for MQTT applications, see 5.8.4, “Creating a Cordova plug-in” on page 158.

5.6 Configuring the Android SDK and test environment

The Android SDK provides developers the API libraries and developer tools necessary to build, test, and debug applications for Android.

The Android SDK includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials.

Currently supported development platforms include computers running Linux, Mac OS X 10.5.8 or later, and Windows XP or later. The officially supported IDE is Eclipse using the Android Development Tools (ADT) Plug-in. This section describes the setup of Android SDK on Windows.

Enhancements to Android's SDK are closely integrated with the overall Android platform development. The SDK also supports older versions of the Android platform, in case developers want to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

Android applications are packaged in the `.apk` format, and stored under the `/data/app` folder on the Android OS (the folder is accessible only to the root user for security reasons).

Developers can download the Android SDK from the following website:

`http://developer.android.com/sdk/index.html`

5.6.1 Android SDK tools

The Android SDK includes a variety of tools that help developers develop mobile applications for the Android platform. The tools are classified into two groups:

- SDK tools
- Platform tools

The SDK tools are installed with the SDK starter package and are periodically updated. The SDK tools are required by developers for developing Android applications. The following list includes some of the most important SDK tools:

- The Android SDK Manager (Android SDK)
- The Android Virtual Devices Manager (AVD Manager)
- The Android Emulator (emulator)
- The Dalvik Debug Monitor Server (DDMS)

Table 5-2 summarizes a list of frequently-used SDK tools.

Table 5-2 Android SDK tools

SDK tools	Description
Android SDK and AVD Manager	Helps you manage AVDs, projects, and the installed components of the SDK
Dalvik Debug Monitor Server (DDMS)	Helps you debug Android applications
dmtracedump	Helps you generate graphical call-stack diagrams from trace log files
Draw 9-patch	Helps you easily create a NinePatch graphic using a what you see is what you get (WYSIWYG) editor

SDK tools	Description
Android Emulator (emulator)	A QEMU-based device emulation tool that you can use to design, debug, and test your applications in an actual Android runtime environment
Hierarchy Viewer (hierarchyviewer)	Helps you debug and optimize an Android application's user interface
hprof-conv	Converts the `HPROF` file that is generated by the Android SDK tools to a standard format, so you can view the file in a profiling tool of your choice
layoutopt	Helps you quickly analyze your application layouts to optimize them for efficiency
mksdcard	Helps you create a disk image that you can use with the emulator, to simulate the presence of an external storage card (such as an SD card)
Monkey	Runs on your emulator or device and generates pseudo-random streams of user events, such as clicks, touches, or gestures, and several system-level events, and can be used to stress test the application in a random yet repeatable manner
monkeyrunner	Provides an API for writing programs that control an Android device or emulator from outside of Android code
ProGuard	Shrinks, optimizes, and obfuscates application code by removing unused code and renaming classes, fields, and methods with semantically obscure names
Systrace	Helps you analyze the execution of applications in the context of system processes, to help diagnose display and performance issues
sqlite3	Helps you access the SQLite data files created and used by Android applications
traceview	Provides a graphical viewer for execution logs saved by application
zipalign	Optimizes `.apk` files by ensuring that all decompressed data starts with a particular alignment relative to the start of the file

Platform tools

The platform tools are installed with the SDK platform. Platform tools are compatible with earlier platform versions. Developers typically use only one of the platform tools: The Android Debug Bridge (ADB). ADB is an important tool that helps developers manage the state of an emulator instance or Android powered devices.

Developers can also install an Android application (`.apk`) file on a device using ADB. The other platform tools are typically called by the Android build tools or ADT, so developers rarely need to start these tools directly.

Note: Before installing the Android SDK, verify the system requirements for this tool. You can obtain the Android SDK system requirements from the following website:

`http://developer.android.com/sdk/index.html`

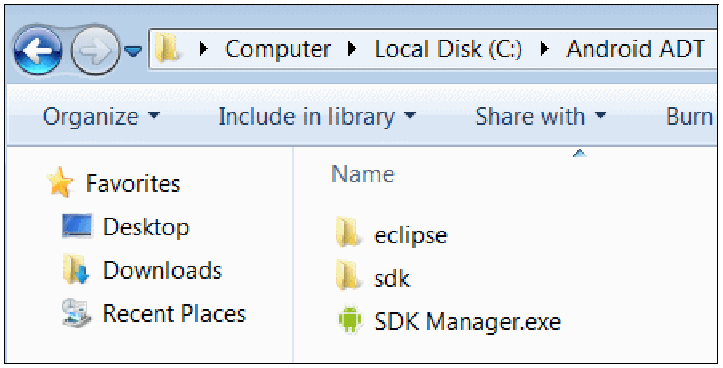
5.6.2 Downloading and installing the ADT bundle

Complete the following steps to install ADT:

1. Download the Android SDK bundle from the following website:

   `http://developer.android.com/sdk/index.html`

2. Extract the bundle in a dedicated directory on the file system.
3. After the `.zip` file is extracted, you see the files and folders, as shown in Figure 5-8.



eclipse
sdk
SDK Manager.exe

Figure 5-8 Android SDK bundle

4. Click **eclipse** → **eclipse.exe**.
5. Provide the workspace location to open a new development workspace.

6. A new workspace is started for application development.
7. Click the **Android SDK Manager** icon in the Java perspective, as shown in Figure 5-9.

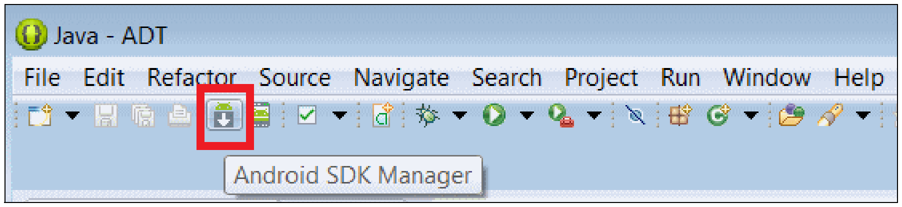
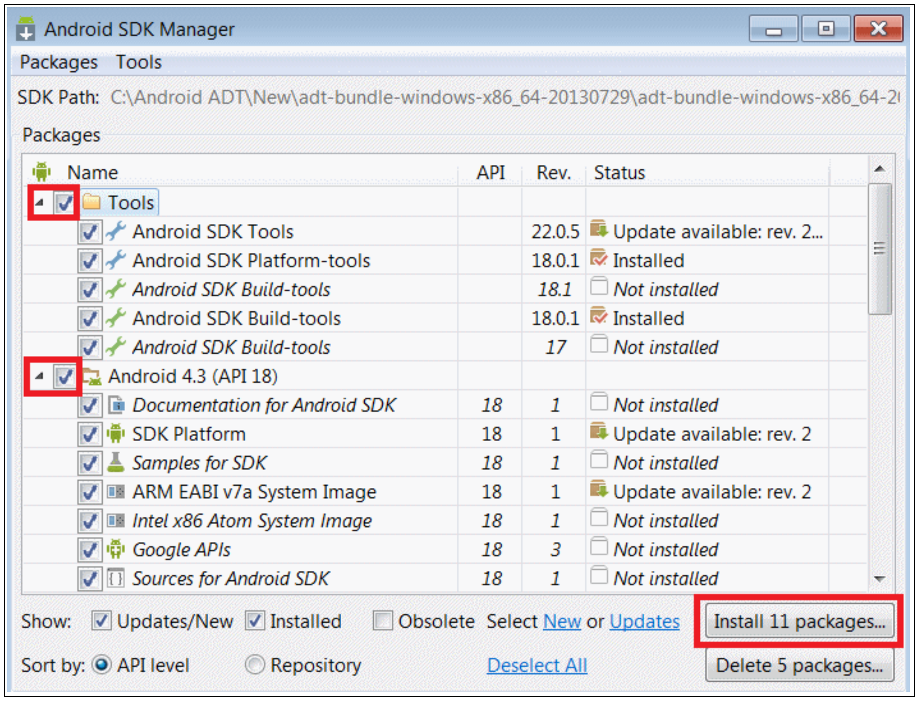

Java - ADT
File Edit Refactor Source Navigate Search Project Run Window Help
Android SDK Manager

Figure 5-9 Android SDK Manager

8. Select packages in the Android SDK Manager tool and click **Install <*number*> Packages**, as highlighted in Figure 5-10. After the packages are installed, Android application on API level 18 can be developed.



SDK Path: C:\Android ADT\New\adt-bundle-windows-x86_64-20130729\adt-bundle-windows-x86_64-2

Packages

Name	API	Rev.	Status
✓ Tools			
✓ Android SDK Tools		22.0.5	Update available: rev. 2...
✓ Android SDK Platform-tools		18.0.1	Installed
✓ *Android SDK Build-tools*		*18.1*	*Not installed*
✓ Android SDK Build-tools		18.0.1	Installed
✓ *Android SDK Build-tools*		*17*	*Not installed*
✓ Android 4.3 (API 18)			
✓ *Documentation for Android SDK*	*18*	*1*	*Not installed*
✓ SDK Platform	18	1	Update available: rev. 2
✓ *Samples for SDK*	*18*	*1*	*Not installed*
✓ ARM EABI v7a System Image	18	1	Update available: rev. 2
✓ *Intel x86 Atom System Image*	*18*	*1*	*Not installed*
✓ *Google APIs*	*18*	*3*	*Not installed*
✓ *Sources for Android SDK*	*18*	*1*	*Not installed*

Show: ✓ Updates/New ✓ Installed ☐ Obsolete Select New or Updates [Install 11 packages...]

Sort by: ● API level ○ Repository Deselect All [Delete 5 packages...]

Figure 5-10 Android SDK Manager and package updates

9. Now you can install Worklight Studio and create Android SDK test emulator.

5.6.3 Installing IBM Worklight studio in Android SDK

You can install and configure the Worklight Developer Edition within Android SDK. You must install Eclipse Marketplace client first.

Configuring Eclipse Marketplace client

Complete these steps to configure the Eclipse Marketplace client:

1. In Android SDK, click **Help** → **Install New Software**.
2. Select the Eclipse Juno repository:

   `http://download.eclipse.org/releases/juno`
3. Select **General Purpose Tools** → **Marketplace Client**, and then click **Next**.
4. Accept the license and install Eclipse Marketplace client.
5. When the Eclipse Marketplace client is installed, restart Android SDK.

   Figure 5-11 shows the Eclipse Marketplace client, now available in the Help menu of Android SDK.

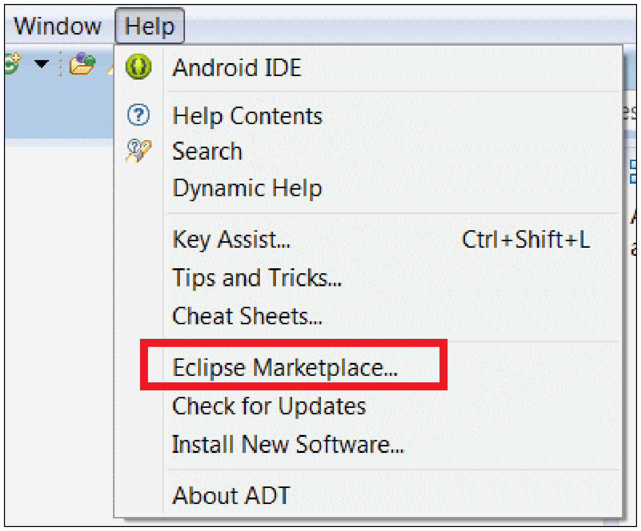


Figure 5-11 Marketplace client in Android SDK

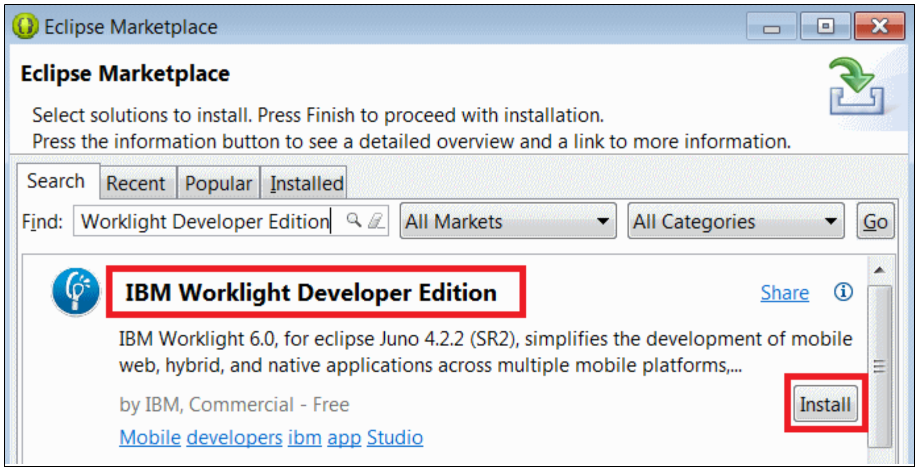
Installing IBM Worklight Developer Edition

Complete the following steps to install IBM Worklight Developer Edition:

1. In Android SDK, click **Help** → **Eclipse Marketplace**.
2. Click **Search** → **Find text box**.

3. Enter `Worklight Developer Edition` then click the **Search** icon.

   Figure 5-12 shows the search result returned after searching for the text `Worklight Developer Edition`.



Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Find: Worklight Developer Edition

IBM Worklight Developer Edition

IBM Worklight 6.0, for eclipse Juno 4.2.2 (SR2), simplifies the development of mobile web, hybrid, and native applications across multiple mobile platforms,...

by IBM, Commercial - Free

Mobile developers ibm app Studio

Figure 5-12 Search results for IBM Worklight Developer Edition

4. Click **Install**.
5. In the Confirm Selected Features dialog, leave everything selected, and click **Next**.
6. In the Review License dialog, accept the license terms and click **Finish**.
7. In the dialog prompting to restart Eclipse, click **Yes**.
8. When the IBM Worklight Developer Edition is installed, and Android SDK is restarted, switch to the Design perspective.
9. Select **Window Open Perspective** → **Other** → **Design**. Click **OK**.

Note: All development steps for this hybrid application are performed in the Design perspective of Worklight Studio. If you are required to change the perspective in the development tool, that will be clearly stated.

5.6.4 Configuring the Android emulator

Android emulator is a QEMU-based device-emulation tool that developers can use to design, debug, and test their applications in an actual Android runtime environment. This section describes steps to configure an Android device emulator:

1. In the Android SDK tool Java perspective, click **Android Virtual Device Manager**, as shown in Figure 5-13.

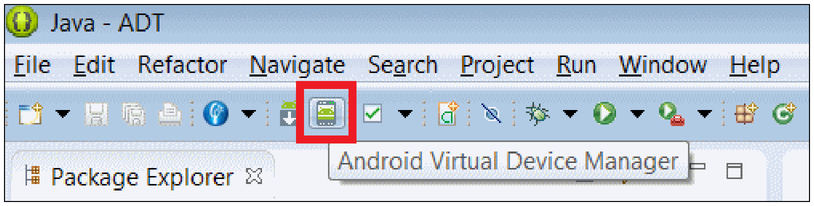
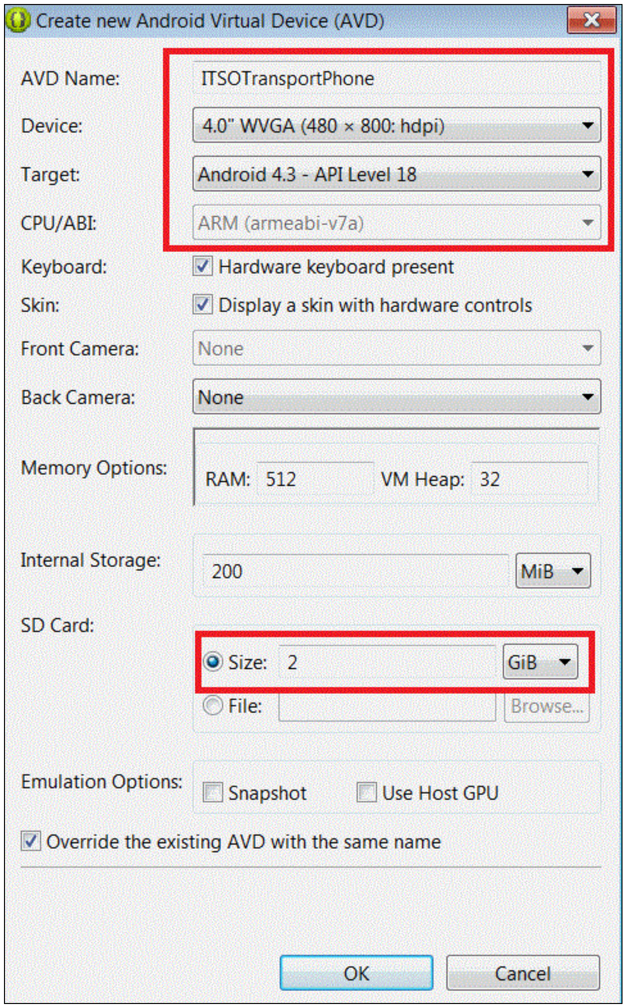


Figure 5-13 Android Virtual Device Manager

2. Complete the Android Virtual Device creation wizard, as shown in Figure 5-14 on page 129:
   a. Enter `ITSOTransportPhone` as the AVD Name.
   b. Set the Device to **4.0” WVGA (480 X 800: hdpi)**.
   c. Set the Target to **Android 4.3 - API Level 18**.
   d. Set the CPU/ABI to **ARM (armeabi-v7a)**.
   e. Select **Size** in **GiB** and enter `2` (this is required because the MQTT plug-in uses the SD card for persistence).
   f. Click **OK**.



AVD Name: ITSOTransportPhone
Device: 4.0" WVGA (480 × 800: hdpi)
Target: Android 4.3 - API Level 18
CPU/ABI: ARM (armeabi-v7a)
Keyboard: [✓] Hardware keyboard present
Skin: [✓] Display a skin with hardware controls
Front Camera: None
Back Camera: None
Memory Options: RAM: 512 VM Heap: 32
Internal Storage: 200 MiB
SD Card: [● Size: 2 GiB] ○ File:
Emulation Options: ☐ Snapshot ☐ Use Host GPU
[✓] Override the existing AVD with the same name

Figure 5-14 Android Virtual Device creation wizard

When the Android Virtual Device creation completes, a new device named `ITSOTransportPhone` is available in Android Virtual Device Manager.

5.7 MQTT hybrid application use case and requirements

5.8, “Developing an MQTT hybrid application for Android” on page 143, describes the process and procedure to develop the MQTT hybrid application for Android. However, it is important to understand the functional requirements and use cases for the application developed. This section describes company’s business profile, their business problems, requirements and use case for a mobile application.

5.7.1 About the company

This mobile application is developed for a hypothetical company called ITSO Transport Corporation. This company provides transport services to grocery retailers around the country. Trucks from this company are scheduled to pick up groceries from centralized cold storage and deliver groceries to the end retail stores using trucks enabled with cold storage containers.

Timely delivery of groceries is the key success attribute for ITSO Transport Corporation business, and it helps build their customer satisfaction. Every truck scheduled for delivery is equipped with two crew members, a truck driver and an assistant.

5.7.2 Business problem

Timely deliver of groceries is critical to ITSO Transport company. Any delay during the delivery can cause groceries to get stale. The company observes that deliveries are obstructed by a variety of problems on the highway. The following list includes a few of the key problems:

- Trucks get into problems, such as flat tires, problems with container cold storage, road accidents, and so on, for which truck drivers need immediate roadside assistance. If truck drivers need to schedule and coordinate assistance accurately and correctly on their own, that takes much of their time and can cause further delivery delay.
- Truck drivers who are not aware of traffic congestion and weather problems on their delivery route can get engaged with unexpected problems and cause delivery delays.

5.7.3 Application requirements

ITSO Transport corporation wants to develop an Android mobile application for the Android devices installed in delivery trucks. This mobile application must fulfill the following functional requirements:

- Authenticate the truck driver using a driver user name and password.
- Authorize the mobile application with the connection and messaging policies of IBM MessageSight.
- Truck drivers log in to this application before starting on a delivery, and log out from this application when they get the truck back to the station. The mobile application makes durable subscriptions on topics when drivers log in to the application, and unsubscribes from the same topics when a logout is performed.
- The mobile application captures changes in the geographic location of the truck, and transmits that data to back-end IBM MessageSight, with the longitude and latitude of the truck's location.
- The truck's assistant can request roadside assistance using the mobile application.
- The roadside assistance request message includes the type of assistance required, and the geographic location of the truck where assistance needs to be dispatched.
- The roadside assistance request and response message must be sent with the highest quality of service (QoS2), for *exactly once* delivery of these critical messages.
- Based on the truck's location as captured by back-end services, back-end services send push notifications to the mobile application, alerting drivers for any predicted problems on their destination route. These push notifications might be weather notifications, traffic-related alerts, or weigh station reminders. Example 5-3 shows examples of these critical alerts.

Example 5-3 Push notification example

```
– Inclement weather in your area, drive safely and inform main
  office if you expect delivery to be delayed
– Highway construction on I-75, use alternative route if possible
– Tornado Warnings issued in your area, take cover immediately
– Truck malfunction in your area, inform front office if you are
  available to pick up load for delivery
– Traffic Jam reported on Highway 5, use alternative route
– Accident on Interstate 10, take alternative route
– Heavy snow - use precaution and engage inclement weather driving
  gear
```

```
– Bay bridge is closed, plan alternative route
– Highway 92 closed, take alternative route
– Return to station, additional cargo is ready for delivery
```

- Push notifications must only be sent to relevant trucks, whose truck location matches the business event.
- Push notifications are also sent with QoS2 quality of service, for exactly once delivery of notification to the mobile application in the truck.
- Broadcast of important messages is possible, which enables the back-end office team to broadcast important communication to all of the trucks currently delivering or scheduled for delivery. Example 5-4 shows examples of these broadcast messages.

Example 5-4 Broadcast messages to all trucks

```
- Drive safely and abide by state traffic laws.
- It is month end and your salary is deposited in your bank account.
- Bring truck to nearest service station, for routine service check.
```

- Broadcast messages must be sent with *at least once* delivery quality of service (QoS1). These messages are marked as retained publication, and when drivers log in to the mobile application before starting on a delivery, they will receive the previously broadcast message.

5.7.4 Application use cases

Figure 5-15 shows actors and use cases for ITSO Transport mobile application.

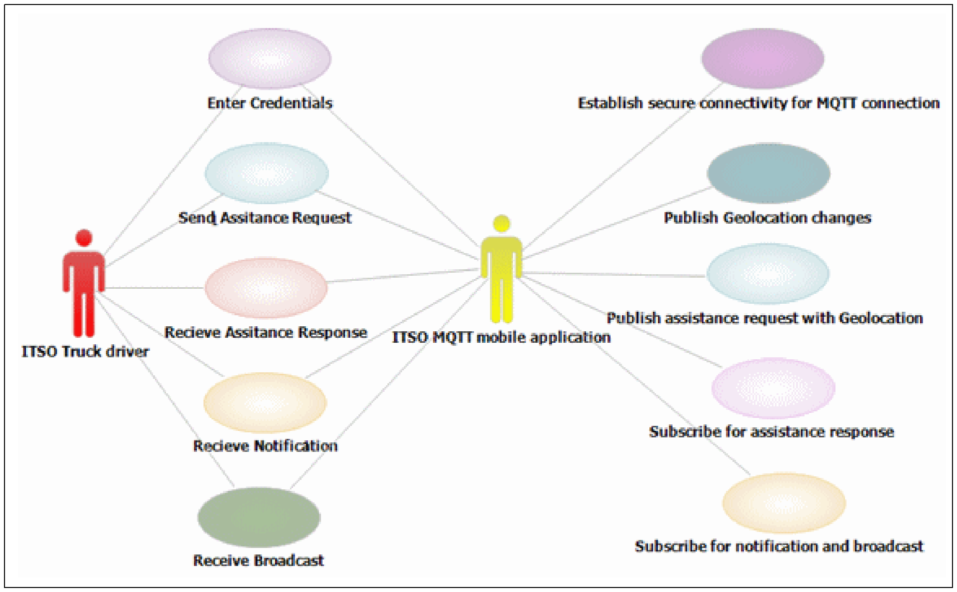

Enter Credentials
Send Assitance Request
Recieve Assitance Response
Recieve Notification
Receive Broadcast
ITSO Truck driver
ITSO MQTT mobile application
Establish secure connectivity for MQTT connection
Publish Geolocation changes
Publish assistance request with Geolocation
Subscribe for assistance response
Subscribe for notification and broadcast

Figure 5-15 Use case diagram for ITSO Transport mobile application

Actors

The following list includes the main actors in context of the ITSO Transport mobile application:

- ITSO truck driver and truck assistant staff
- ITSO Transport mobile application
- IBM MessageSight server

Use cases for ITSO truck driver and truck assistant staff

The truck driver and the truck assistant staff can use the application to perform the following tasks:

- Enter credentials

  Truck driver logs in to the mobile application before starting for a delivery, and log out when back to the station after completing the delivery. The truck driver submits a user name and password for authentication, and a truck number for authorization. These credentials are used by the mobile application to establish secure connectivity with IBM MessageSight server. The truck number is used as a client identifier.

- Send roadside assistance request

  Truck staff must be able to request roadside assistance for any problems with the truck. The assistance message includes the type of request needed, and the geographic location where the assistance needs to be dispatched.

- Receive roadside assistance response

  A response message as an acknowledgment to an assistance request is received on the mobile application for the staff in truck.

- Receive notification messages

  Based on the geographic location, the truck staff receives notifications from the back-end office. These notifications make truck staff aware of any problems predicted on their delivery route.

- Receive broadcast messages

  The back-end office might want to send broadcast messages to all of its truck staff delivering groceries. The truck staff receives these broadcast messages through the mobile application.

Use cases for the ITSO Transport mobile application

The ITSO Transport mobile application include the following functions:

- Secure connection

  The mobile application establishes secure connectivity using an IBM MessageSight message on an IBM MessageSight server. To establish this connectivity, the mobile application provides driver credentials (user name and password) for authentication, and the truck number for authorization. The truck number is used as a client identifier for this mobile application.

- Publish geographic location change

  The mobile application publishes any changes to the geographic location of the truck to the back-end IBM MessageSight server on a specific topic string.

  More details about the topic string used for publish and subscribe in the ITSO Transport mobile application are included in 5.7.6, “Topic strings used for publish and subscribe” on page 138.

- Publish roadside assistance request

  The mobile application publishes a roadside assistance request, from the truck driver to the back-end IBM MessageSight server, as a publication on a specific topic string.

- Subscribe roadside assistance response

  The mobile application subscribes to the topic string on which the roadside assistance response will be received. The roadside assistance response message received from the IBM MessageSight server is alerted and displayed through the assistance request tab in the mobile application.

- Subscribe notification messages

  The mobile application subscribes to the topic string on which the push notification specific to this truck will be received. The push notification message received from the IBM MessageSight server is alerted and displayed through the notification tab in the mobile application.

- Subscribe broadcast messages

  The mobile application subscribes to the topic string on which the broadcast messages from the back-end office will be received. The broadcast message received from the IBM MessageSight server is alerted and displayed through the notification tab in the mobile application.

Use case for the IBM MessageSight server

The application uses IBM MessageSight server for the following functions:

- Secure connection

  The IBM MessageSight server receives a secure MQTT connection request from the ITSO Transport mobile application. After authentication and authorization is completed, a secure connectivity is established between the mobile application and the IBM MessageSight server.

- Geographic location changes

  The IBM MessageSight server receives publications from mobile applications, on a specific topic, for changes to the geographic location of the truck. These messages are later published to the back-end application, which has subscribed on a matching topic string.

- Roadside assistance request

  The IBM MessageSight server receives publications from the mobile application, on specific topics, for roadside assistance requests from the truck staff. These messages are later published to the back-end application, which has subscribed on a matching topic string.

- Roadside assistance response

  The IBM MessageSight server receives a publication from the back-end enterprise application, on a specific topic, for a response to a corresponding roadside assistance request received from the mobile application. This response message is later published to the mobile application that sent the assistance request and has subscribed on a matching topic string.

- Notification message

  The IBM MessageSight server receives a publication from the back-end enterprise application, on a specific topic, to send a notification message to a specific truck. The notification message is later published to the mobile application, which has subscribed on a matching topic string.

- Broadcast message

  The IBM MessageSight server receives a publication from the back-end enterprise application, on a specific topic, to send broadcast messages to all trucks currently delivering groceries. The broadcast message is later published to the mobile application, which has subscribed on a matching topic string.

5.7.5 Application visual blueprint

The ITSO Transport mobile application provides several features:

- Secure connectivity of the application to the IBM MessageSight server
- Road assistance request and response for truck staff
- Notification and broadcast message from the back-end office for truck staff
- History log of publication, subscription, and connection to IBM MessageSight

These features are provided through different tabs in this mobile application. Figure 5-16 and Figure 5-17 on page 138 show the different tabs and user interfaces to be developed in the ITSO Transport hybrid mobile application.

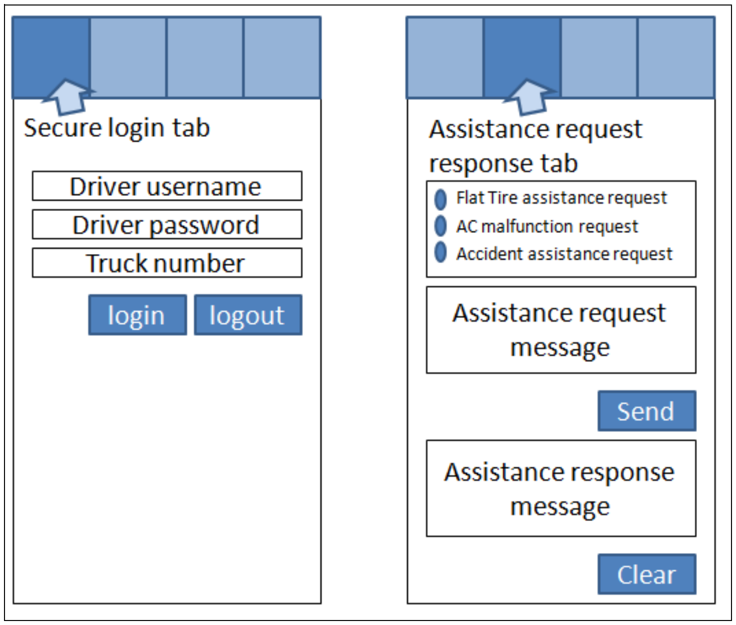

Secure login tab
Driver username
Driver password
Truck number
login
logout
Assistance request response tab
Flat Tire assistance request
AC malfunction request
Accident assistance request
Assistance request message
Send
Assistance response message
Clear

Figure 5-16 Secure login and assistance request tabs in the mobile application

Figure 5-17 shows the mobile application notifications and history.

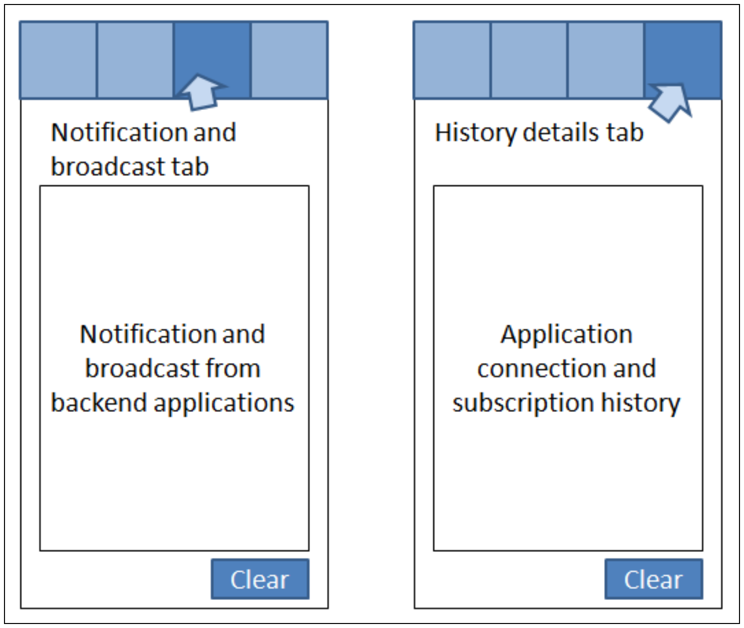

Notification and broadcast tab
Notification and broadcast from backend applications
Clear
History details tab
Application connection and subscription history
Clear

Figure 5-17 Push notification and broadcast and log history tabs in the mobile application

5.7.6 Topic strings used for publish and subscribe

The topic string is the label attached to an application message that is matched against the subscriptions known to the IBM MessageSight server. The application message is sent to each client with a matching subscription. To read more details about the MQTT topic name, see Appendix A, “MQTT protocol” on page 303, or the MQTT specification document on the following website:

http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

The ITSO Transport mobile application uses various topics to publish application messages to the back-end IBM MessageSight server, and to subscribe on multiple topics to receive messages from the back-end enterprise applications. This section provides information about a list of these topics.

Note: The truck number is not part of the published message payload from the mobile application, because the truck number is already included in the topic.

Geographic location change message

The ITSO Transport mobile application publishes a message when there is a geographic location change for the truck. This message is in comma-separated value (CSV) format, as shown in Example 5-5. The message contains the current time stamp, driver user name, current longitude coordinates, and current latitude coordinates.

Example 5-5 Message format for geographic location changes

```
TimeStamp,driver username,longitude,latitude
```

This message is published on the topic mentioned in Table 5-3.

Table 5-3 Topic string for capturing geographic location changes

ITSO Transport mobile application	**Direction of message delivery through IBM MessageSight**	**Back-end enterprise application**

Action: PUBLISH
`Topic:`
`/itso/driver/geolocation/`
`<Truck Number>`
QoS of messages: 0
Retained: No
Protocol: MQTT
Note: This topic is used to publish changes in the geographic location of truck.
	---------------------------->	Action: SUBSCRIBE
		Topic:
		`/itso/driver/geolocation`
		`/+`
		Protocol: JMS

Roadside assistance request message

The ITSO Transport mobile application enables its drivers to send roadside assistance requests for several scenarios:

- Flat truck tires, and truck staff needs truck mechanical assistance
- Container AC malfunction, need assistance from an engineer
- Truck in accident situation and truck staff needs emergency assistance

The roadside assistance request is in CSV format, as shown in Example 5-6.

Example 5-6 Message format for roadside request assistance

```
TimeStamp,driver username,longitude,latitude,text message from driver
```

The roadside assistance request message contains the current time stamp, driver user name, current longitude coordinates, current latitude coordinates, and a customized text message that the driver can type in the Android application.

Table 5-4 provides details about various topics that are used for sending roadside assistance requests. There are three different topics about which a request can be sent, depending on the assistance required.

Table 5-4 Topic string for roadside assistance request

ITSO Transport mobile application	**Direction of message delivery through IBM MessageSight**	**Back-end enterprise application**
Action: PUBLISH Topic: `/itso/driver/assistance/request /FlatTire/<`*`TruckNumber`*`>` QoS of messages: 2 Retained: No Protocol: MQTT Note: This topic is used to publish an assistance request for a flat tire problem.	---------------------->	Action: SUBSCRIBE Topic: `/itso/driver /assistance/request/#` Protocol: JMS
Action: PUBLISH Topic: `/itso/driver/assistance/request /ACMalfunction/<`*`TruckNumber`*`>` QoS of messages: 2 Retained: No Protocol: MQTT Note: This topic is used to publish an assistance request for an AC malfunction problem.	---------------------->	
Action: PUBLISH Topic: `/itso/driver/assistance/request /Accident/<`*`TruckNumber`*`>` QoS of messages: 2 Retained: No Protocol: MQTT Note: This topic is used to publish an assistance request for an accident-related emergency.	---------------------->	

Roadside assistance response message

For a roadside assistance request sent by the ITSO Transport mobile application, there will be a corresponding response received from the back-end enterprise application. To get a response, the ITSO Transport mobile application must subscribe on a topic string, as shown in Table 5-5. The data received is in plain UTF-8 characters with no specific format.

Example 5-7 shows examples of a few response messages sent back to the mobile application.

Example 5-7 Sample messages for a roadside assistance response

```
- <driver username>, Vehicle mechanic dispatched to your nearest location.
- <driver username>, AC Engineer dispatched to your nearest location.
- <driver username>, Emergency staff notified and dispatched to your nearest
location.
```

Table 5-5 Topic string for roadside assistance response

ITSO Transport mobile application	Direction of delivery through IBM MessageSight	Back-end enterprise application
Action: SUBSCRIBE Topic: `/itso/driver/assistance/response/<TruckNumber>` Subscription's maximum QoS: 2 Protocol: MQTT Note: This topic is used to receive responses for roadside assistance request.	<------------------------	Action: PUBLISH Topic: `/itso/driver/assistance/response/<TruckNumber>` Retained: No Protocol: JMS

Push notification messages from the back-end application

Based on the geographic location of trucks as captured by the back-end application, truck drivers can automatically notify the truck staff about any emergency or business-related situations through notification messages. There is no specific format for these messages, and a few examples of them were provided in Example 5-3 on page 131.

Table 5-6 shows the topic string on which the ITSO Transport mobile application receives push notifications from the back-end enterprise applications.

Table 5-6 Topic string for push notification messages

ITSO Transport mobile application	**Direction of delivery through IBM MessageSight**	**back-end enterprise application**

Action: SUBSCRIBE
Topic:
`/itso/driver/notification`
`/<TruckNumber>`
Subscription's maximum QoS: 2
Protocol: MQTT
Note: This topic is used to receive push notification message from back-end enterprise applications.
	<-------------------------	Action: PUBLISH
		Topic:
		`/itso/driver`
		`/notification/`
		`<TruckNumber>`
		Retained: No
		Protocol: JMS

Broadcast messages from the back-end application

The ITSO Transport back-end office team can send broadcast messages to the mobile application in trucks. These messages are intended for all of the drivers currently delivering groceries or who are scheduled for later delivery. These are generally non-critical messages.

Therefore, the messages are sent with QoS1 and marked as retained, so when a driver logs in to the application before delivery, he still receives the previously broadcast message. Example 5-4 on page 132 showed a few examples of these broadcast messages.

Table 5-7 shows the topic string on which the ITSO Transport mobile application receives broadcast messages from the back-end office team.

Table 5-7 Topic string for broadcast messages

ITSO Transport mobile application	**Direction of delivery through IBM MessageSight**	**Back-end enterprise application**

Action: SUBSCRIBE
Topic:
`/itso/driver/notification`
`/AllTrucks`
Subscription's maximum QoS: 1
Protocol: MQTT
Note: This topic is used to receive broadcast message from back-end office team.
	<-------------------------	Action: PUBLISH
		Topic:
		`/itso/driver`
		`/notification/AllTrucks`
		Retained: Yes
		Protocol: JMS

5.8 Developing an MQTT hybrid application for Android

This section describes the process and procedure to develop an MQTT hybrid application for Android platform using IBM Worklight Studio. You learn about the development process through an example. The steps described in this section are applicable for other similar application development, not just specific for the example provided in this chapter.

You can download this example from the IBM Redbooks website. For the download instructions, see Appendix D, "Additional material" on page 341.

Figure 5-7 on page 120 illustrates the architecture of a hybrid mobile application. The components that are primarily described in this section have a yellow background.

Start by creating a Worklight hybrid project in IBM Worklight Studio. This application is developed using the bottom-up approach, and the architecture components are designed, configured, developed in the following order:

1. Create an IBM Worklight project and Worklight environment.
2. Create MQTT client layer using Eclipse Paho client.
3. Create Android services.
4. Create Cordova plug-in.
5. Create application logic and user interface.
6. Create Android manifest.
7. Build and deploy the application.
8. Run and test the application.

5.8.1 Creating an IBM Worklight project and Worklight environment

In this section, you create a new IBM Worklight project for a hybrid application, and then create a Worklight environment for Android platform. Follow these steps to create a Worklight project and environment:

1. Go to the directory where Android SDK was extracted previously.
2. Click **eclipse** → **eclipse.exe**.
3. Provide the workspace location to open a new development workspace.
4. In the new workspace started for application development, click **Windows** → **Open Perspective** → **Others**.

5. Select **Design** perspective and click **OK**, as shown in Figure 5-18.

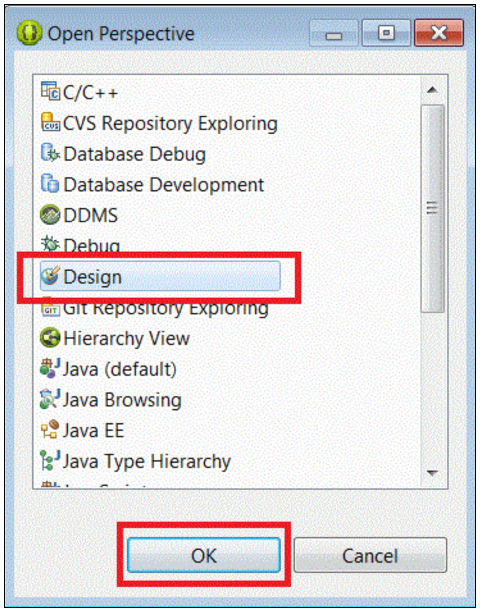


Figure 5-18 Design perspective

6. In project explorer view, right-click and select **New** → **Worklight Project**, as shown in Figure 5-19.

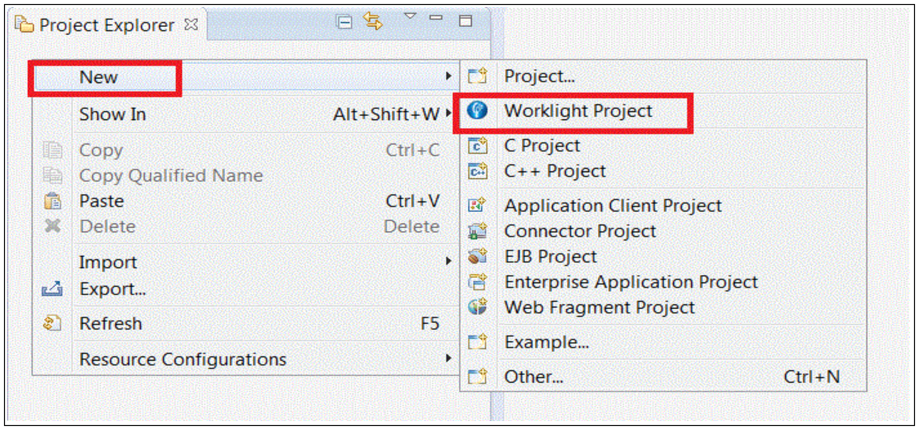
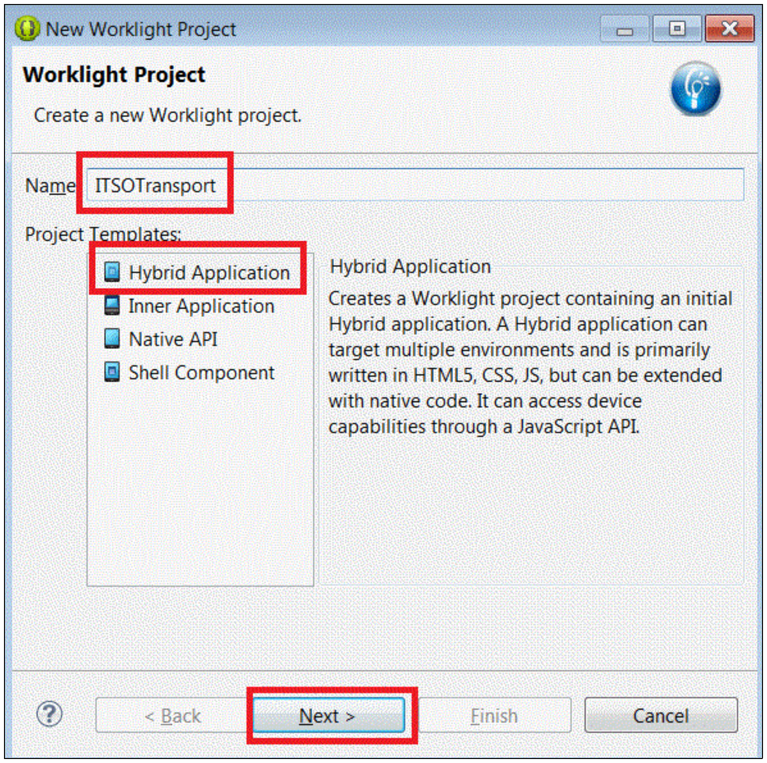


Figure 5-19 Worklight project

7. In the new Worklight project wizard, enter `ITSOTransport` as the project name, select **Hybrid Application** under Project Templates, and then click **Next** (Figure 5-20).



Name: ITSOTransport

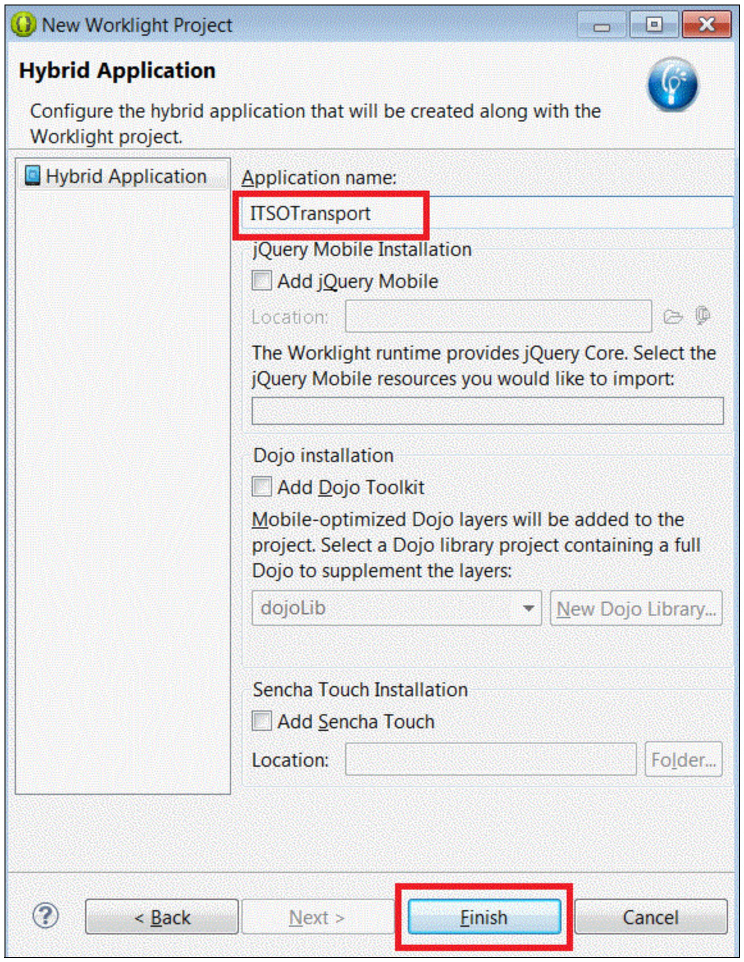
Project Templates: Hybrid Application

Hybrid Application

Creates a Worklight project containing an initial Hybrid application. A Hybrid application can target multiple environments and is primarily written in HTML5, CSS, JS, but can be extended with native code. It can access device capabilities through a JavaScript API.

Figure 5-20 Worklight project name and application type

8. On the next window in the New Worklight Project wizard, enter `ITSOTransport` as the application name, and then click **Finish** (Figure 5-21).



New Worklight Project

Hybrid Application

Configure the hybrid application that will be created along with the Worklight project.

Hybrid Application

Application name: ITSOTransport

jQuery Mobile Installation

Add jQuery Mobile

Location:

The Worklight runtime provides jQuery Core. Select the jQuery Mobile resources you would like to import:

Dojo installation

Add Dojo Toolkit

Mobile-optimized Dojo layers will be added to the project. Select a Dojo library project containing a full Dojo to supplement the layers:

dojoLib

Sencha Touch Installation

Add Sencha Touch

Location:

Figure 5-21 Worklight application name

9. Open the `application-descriptor.xml` file from the Project Explorer view (Figure 5-22).

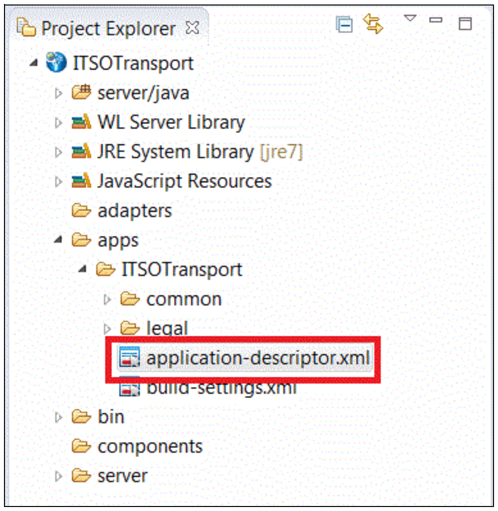
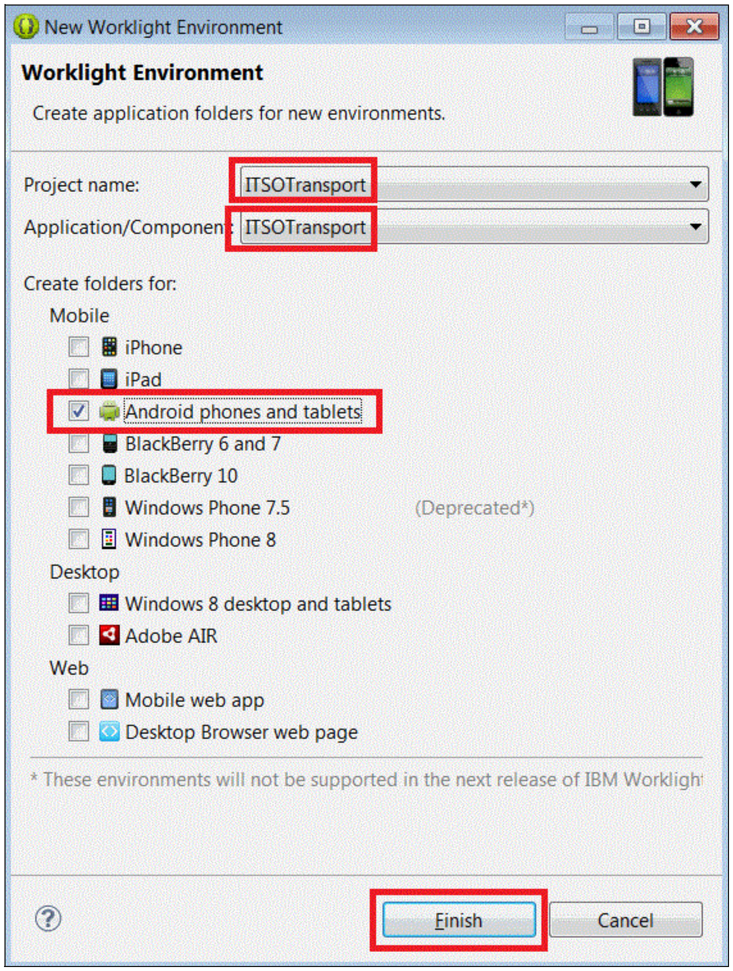


Figure 5-22 The application-descriptor.xml file

10. Update the application display name to `ITSO Transport` and add a description of the application. Save the changes made in the `application-descriptor.xml` file.
11. Create a Worklight Environment for the application:
    a. In the Project Explorer view, expand the **ITSOTransport** → **apps** → **ITSOTransport** project.
    b. Right-click the **ITSOTransport** → **New** → **Worklight Environment** application.
12. A new Worklight Environment wizard opens. Select **ITSOTransport** as the Project name and Application/Component.

13. Select **Android phones and tablets** and then click **Finish** (Figure 5-23).



New Worklight Environment

Worklight Environment

Create application folders for new environments.

Project name: ITSOTransport

Application/Component: ITSOTransport

Create folders for:

Mobile
- [ ] iPhone
- [ ] iPad
- [x] Android phones and tablets
- [ ] BlackBerry 6 and 7
- [ ] BlackBerry 10
- [ ] Windows Phone 7.5 (Deprecated*)
- [ ] Windows Phone 8

Desktop
- [ ] Windows 8 desktop and tablets
- [ ] Adobe AIR

Web
- [ ] Mobile web app
- [ ] Desktop Browser web page

* These environments will not be supported in the next release of IBM Worklight

Figure 5-23 Worklight Environment for Android platform

14. After the new Worklight Environment wizard completes the environment creation, a new project for Android named ITSOTransportITSOTransportAndroid is seen in the Project Explorer view.

5.8.2 Creating an MQTT client layer using the Eclipse Paho client

You can download the Eclipse Paho client from the Eclipse Paho website. The client is also included with the examples for download on the IBM Redbooks website. See the download instructions in Appendix D, “Additional material” on page 341. Download the `Ch05.zip` file and extract it in your local file system.

This Java library implements the client half of the MQTT V3.1 protocol implementation. The Java client runs on any suitable Java run time, from Java 1.5 on, including Android. Separate interfaces are provided for synchronous and asynchronous styles of operation.

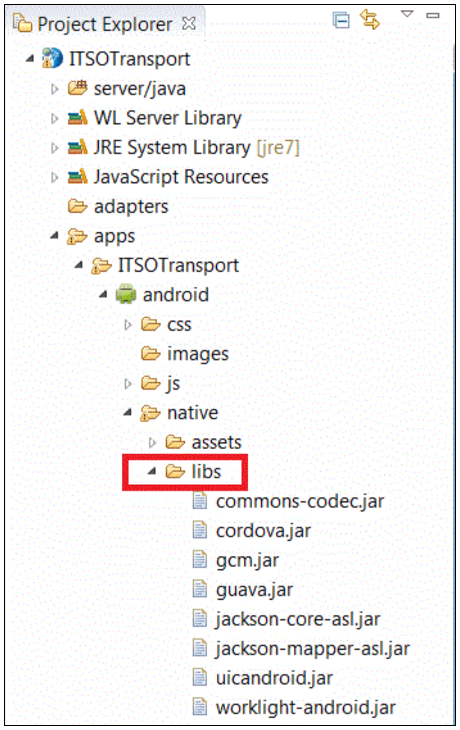
The synchronous API supports a style where an operation only returns to the caller when the operation has completed. This is a traditional style that can be used to implement a simple client. However, the blocking nature of this API limits its usage in environments where threads are not allowed to block, or when high performance or large numbers of clients are required.

The asynchronous API supports a different style in which a call returns immediately. The application can then either be notified through a callback when the operation completes, or it can use a token returned to the application to block until the operation completes.

This style of API is better suited to mobile, event-oriented, and high-performing applications. Applications where it might take time to complete an operation, or where a thread must not be blocked, are good candidates for the asynchronous API. This client is a version of the open source MQTT client that is available from the Eclipse.org Paho project.

Import this library in the Android environment of the ITSOTransport hybrid application. Follow these steps to import this library in the project:

1. In Project Explorer view, expand the **ITSOTransport** project → **apps** → **ITSOTransport** application → **android** → **native** → **libs** (Figure 5-24).


Project Explorer

```
ITSOTransport
  server/java
  WL Server Library
  JRE System Library [jre7]
  JavaScript Resources
  adapters
  apps
    ITSOTransport
      android
        css
        images
        js
        native
          assets
          libs
            commons-codec.jar
            cordova.jar
            gcm.jar
            guava.jar
            jackson-core-asl.jar
            jackson-mapper-asl.jar
            uicandroid.jar
            worklight-android.jar
```

Figure 5-24 Location to import the Eclipse Paho client

2. Right-click **libs** and click **Import**.

3. In the Import wizard, select **File System** and click **Next** (Figure 5-25).

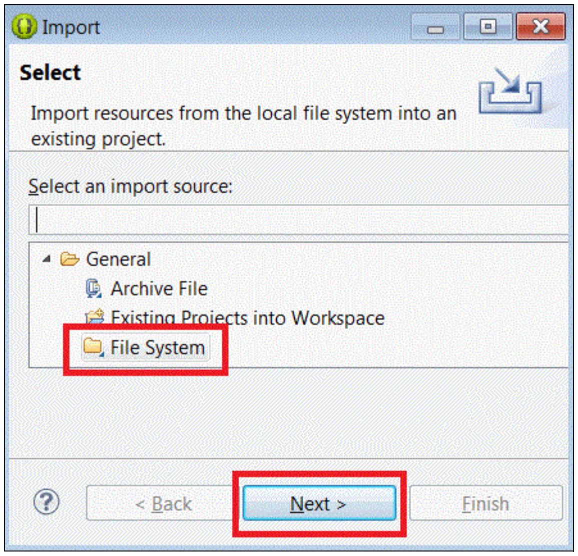
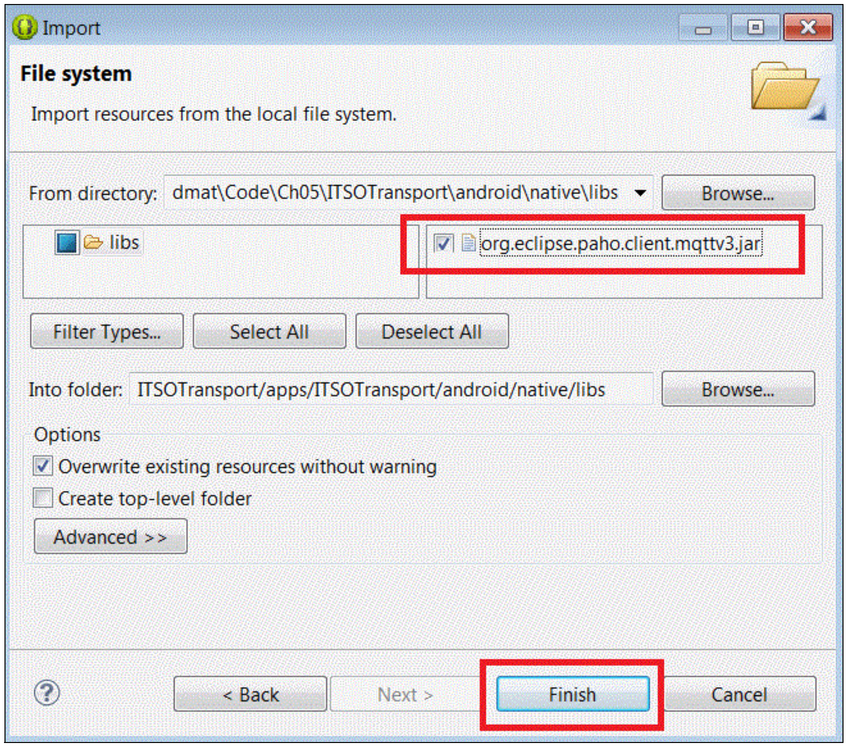

Import
Select
Import resources from the local file system into an existing project.
Select an import source:
General
Archive File
Existing Projects into Workspace
File System
< Back
Next >
Finish

Figure 5-25 Import from file system

4. On the next window in the Import wizard, browse to the `../Ch05/ITSOTransport/android/native/libs` directory where the downloaded example for this chapter was extracted.

5. Select the `org.eclipse.paho.client.mqttv3.jar` file, and then click **Finish** (Figure 5-26).



File system

Import resources from the local file system.

From directory: dmat\Code\Ch05\ITSOTransport\android\native\libs

☑ libs

☑ org.eclipse.paho.client.mqttv3.jar

Into folder: ITSOTransport/apps/ITSOTransport/android/native/libs

Options

☑ Overwrite existing resources without warning

☐ Create top-level folder

Figure 5-26 Import the Eclipse Paho Java client library into the project

6. The Eclipse Paho client library is imported into the `ITSOTransport/apps/ITSOTransport/android/native/libs` folder in the ITSOTransport project (Figure 5-27).

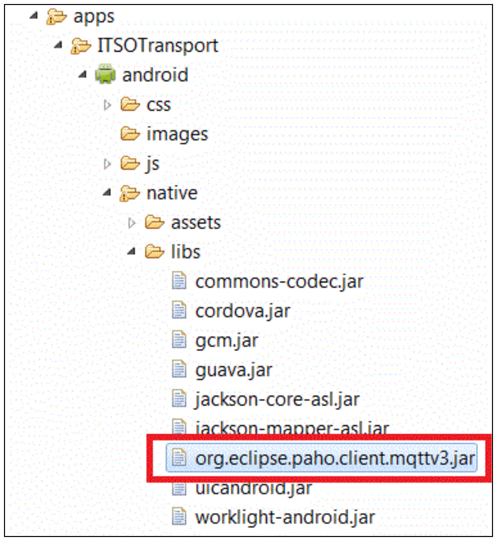


Figure 5-27 Eclipse Paho Java client library available in the project

To read more details about the packages and classes included in the Eclipse Paho Java client library, see the Javadoc website:

`http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp?topic=%2Fcom.ibm.mq.javadoc.doc%2FWMQMQxrClasses%2Findex.html&org%2Feclipse%2Fpaho%2Fclient%2Fmqttv3%2Fpackage-summary.html`

5.8.3 Creating Android services

The function of the Android Service is to take the Eclipse Paho MQTT client and adapt it to run in the Android environment. Complete the following steps to import Android service classes into the ITSOTransport Worklight hybrid application project:

1. In the Project Explorer view, expand **ITSOTransport** project → **apps** → **ITSOTransport** application → **android** → **native** → **src** (Figure 5-28).

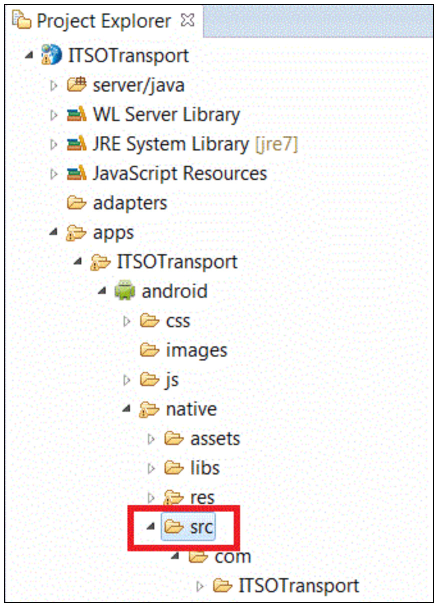


Figure 5-28 Location to import the Android service Java classes

2. Right click **src** and then click **Import**.

3. In the Import wizard, select **File System** and click **Next** (Figure 5-29).

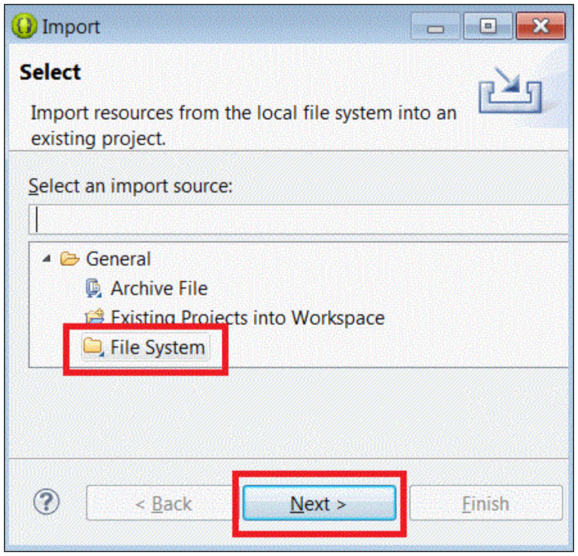

Import
Select
Import resources from the local file system into an existing project.
Select an import source:
General
Archive File
Existing Projects into Workspace
File System
< Back
Next >
Finish

Figure 5-29 Import from File System

4. On the next window in the Import wizard, browse to the `../Ch05/ITSOTransport/android/native/src` directory where the downloaded example for this chapter was extracted.
5. Navigate to the service folder in the left tree view, and select all of the Java classes in this folder, and then click **Finish**, as shown in Figure 5-30.

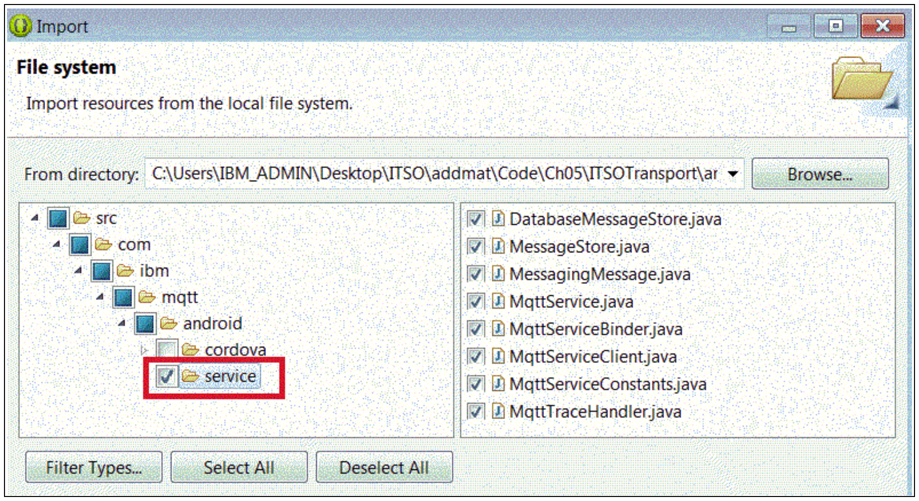


Figure 5-30 Import the Android Java service class

The function of the Android Service is to take the Eclipse Paho MQTT client and adapt it to run in the Android environment. This implementation is structured into several classes in the `com.ibm.mqtt.android.service` package, as shown in Table 5-8.

Table 5-8 Android service class

Android service class	Description
`DatabaseMessageStore`	This is a class that uses the Android SQLite database to hold a store of incoming messages that are en route to being delivered to the application.
`MessageStore`	This Interface has the mechanism for persisting messages until you know they have been received by the application. `DatabaseMessageStore` implements this interface.
`MqttService`	This is the implementation of the Service. It extends the Android Service class.
`MqttServiceBinder`	This is an ancillary class supporting the `MqttService` class.
`MqttServiceClient`	This class contains the bulk of the logic that interfaces to the Paho client.
`MqttServiceConstants`	This class defines constants that are used by the Service (and also by the Cordova plug-in when it interfaces to the service).
`MqttTraceHandler`	This is an ancillary class supporting the `MqttService` class.
`MessagingMessage`	This Java class is a representation of the `Messaging.Message` class from JavaScript.

6. After the Android service classes are imported into the project, you see them available in the workspace, as shown in Figure 5-31.

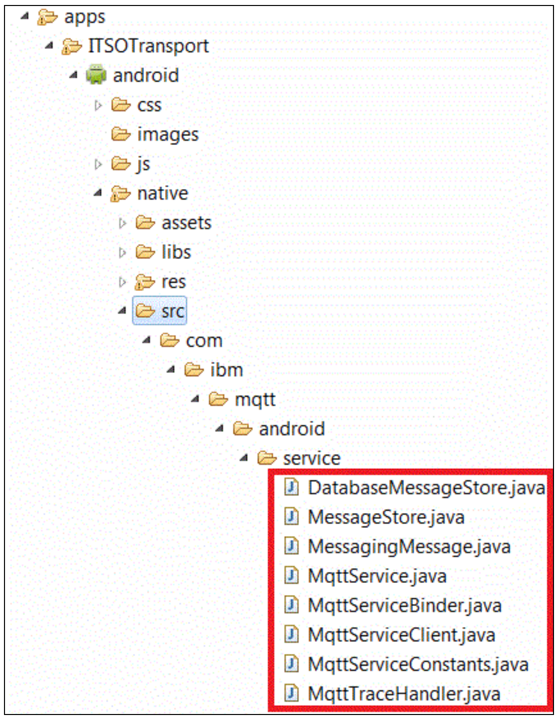


Figure 5-31 Android service class imported in ITSOTransport project

The Android service implements an Android *bound service*. If the application calls the Eclipse Paho MQTT client code directly from the Cordova plug-in, that code is run as part of the application's Android activity. An Android phone only has one activity running at any one time, and when the user switches to a new application, or starts talking on the phone, the current activity is stopped and is placed on a stack when another activity runs in its place.

By placing the MQTT client in an Android service, you can ensure that it continues to run even when the application is stopped. It is important for having the split between the Cordova MQTT plug-in (which is part of the activity, and therefore only runs when the application owns the phone's user interface) and the `MqttService`, which can continue to run in the background.

These two components communicate with each other using Android's message passing mechanism (Android refers to these messages as *intents*), and the asynchronous nature of this mechanism fits well with the asynchronous nature of the JavaScript API described later in this book.

The main reason for wanting a service running in the background is to be able to handle messages that arrive when the application's activity is stopped. The service queues up these messages in the DatabaseMessageStore, and notifies any Activities that it is associated with that it has a message for them.

If and when an activity restarts, its MQTT plug-in will attempt to deliver these messages to the application. `MqttService` also includes a rudimentary interface to the Android Status notification service, to alert the user that there are messages waiting.

5.8.4 Creating a Cordova plug-in

The Cordova plug-in is made up of two parts:

- The JavaScript portion, which you can find in the `android/js/mqttCordovaAndroidClient.js` file
- The Java component, which you can find in the `com.ibm.msg.android.cordova.plugin.MQTTPlugin.Java` class

The JavaScript code implements the API outlined earlier in this book. It collects the parameters to be passed across to the Java code, and handles callbacks from the Java code, following the standard pattern for a Cordova plug-in.

The JavaScript portion communicates to the Java portion through a set of Actions (messages sent between JavaScript and Java), as shown in Example 5-8.

The CONNECT, DISCONNECT, SUBSCRIBE, UNSUBSCRIBE, and SEND actions are sent from JavaScript to Java when the corresponding function in the JavaScript API is called by the application. The ACKNOWLEDGE_RECEIPT action is sent when the JavaScript application's `onMessageArrived` callback has completed processing an incoming message.

Example 5-8 Cordova plug-in actions

```
var PLUGIN_ACTION = {
   SEND_ACTION : "send",
   ACKNOWLEDGE_RECEIPT_ACTION : "acknowledgeReceipt",
   UNSUBSCRIBE_ACTION : "unsubscribe",
   SUBSCRIBE_ACTION : "subscribe",
   DISCONNECT_ACTION : "disconnect",
   CONNECT_ACTION : "connect",
   GET_CLIENT_ACTION : "getClient",
   START_SERVICE_ACTION : "startService",
   STOP_SERVICE_ACTION : "stopService",
   MESSAGE_ARRIVED_ACTION : "messageArrived",
   MESSAGE_DELIVERED_ACTION : "messageDelivered",
   ON_CONNECTION_LOST_ACTION : "onConnectionLost",
   TRACE_ACTION : "trace",
   SET_ON_CONNECT_CALLBACK : "setOnConnectCallbackId",
   SET_ON_CONNECTIONLOST_CALLBACK : "setOnConnectionLostCallbackId",
   SET_ON_MESSAGE_ARRIVED_CALLBACK : "setOnMessageArrivedCallbackId",
   SET_ON_MESSAGE_DELIVERED_CALLBACK : "setOnMessageDeliveredCallbackId",
   SET_TRACE_CALLBACK : "setTraceCallbackId",
   SET_TRACE_ENABLED : "setTraceEnabled",
   SET_TRACE_DISABLED : "setTraceDisabled"
};
```

Remember: The `mqttCordovaAndroidClient.js` Cordova plug-in implementation and `MQTTPlugin.Java` are available for download from the IBM Redbooks website. For download instructions, see Appendix D, "Additional material" on page 341.

To set the MQTT Cordova plug-in for ITSOTransport application, follow these steps:

1. Select **js** (the JavaScript folder) in the Worklight environment of the ITSOTransport application.
2. Right-click it and select **Import**, as shown in Figure 5-32.

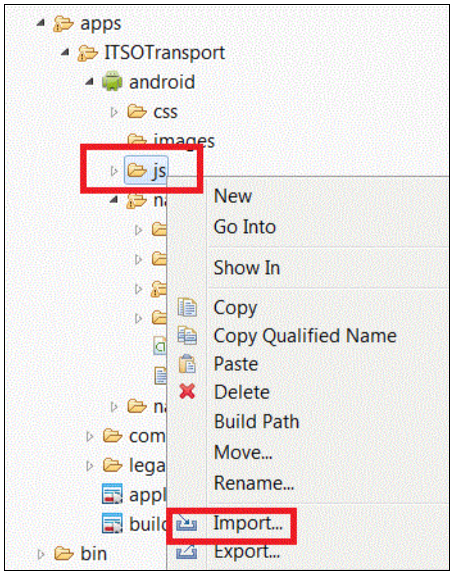


Figure 5-32 Import the Cordova plug-in JavaScript portion

3. Select **File System** and click **Next**.

4. Browse to the `../Ch05/ITSOTransport/android/js` directory where the downloaded material for this chapter was extracted.
5. Select **mqttCordovaAndroidClient.js** and click **Finish**, as shown in Figure 5-33.

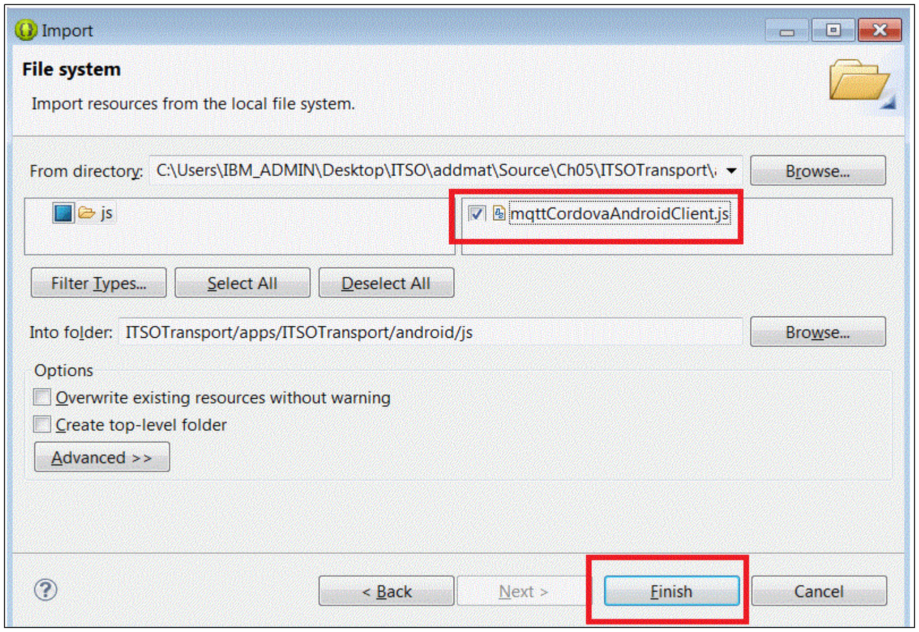

Import
File system
Import resources from the local file system.
From directory: C:\Users\IBM_ADMIN\Desktop\ITSO\addmat\Source\Ch05\ITSOTransport\
Browse...
js
mqttCordovaAndroidClient.js
Filter Types...
Select All
Deselect All
Into folder: ITSOTransport/apps/ITSOTransport/android/js
Browse...
Options
Overwrite existing resources without warning
Create top-level folder
Advanced >>
< Back
Next >
Finish
Cancel

Figure 5-33 Import mqttCordovaAndroidClient.js in Android Worklight environment

6. After `mqttCordovaAndroidClient.js` is imported into the workspace, it is available in the JavaScript folder of Android (`js`), as shown in Figure 5-34.

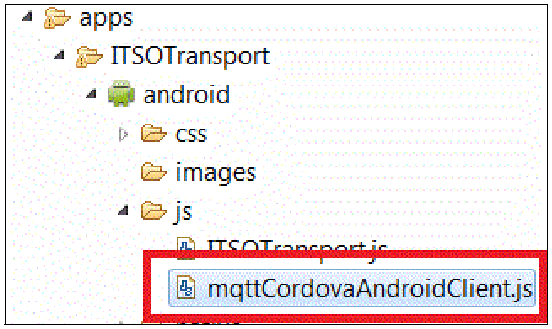


Figure 5-34 The mqttCordovaAndroidClient.js file available in the Worklight project

7. To create the Java component of Cordova plug-in, select the `src` folder in the ITSOTransport application Worklight environment (Figure 5-35), right-click, and select **Import**.

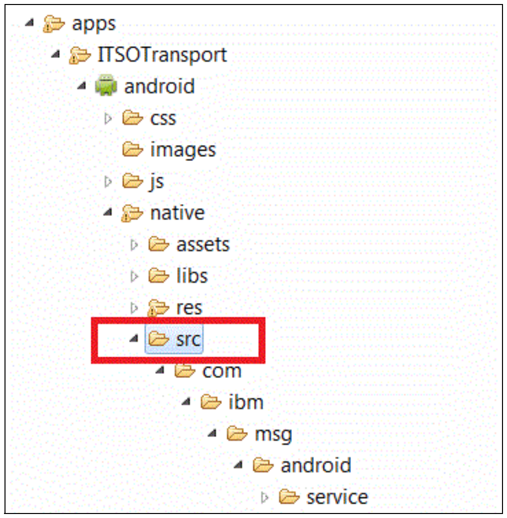
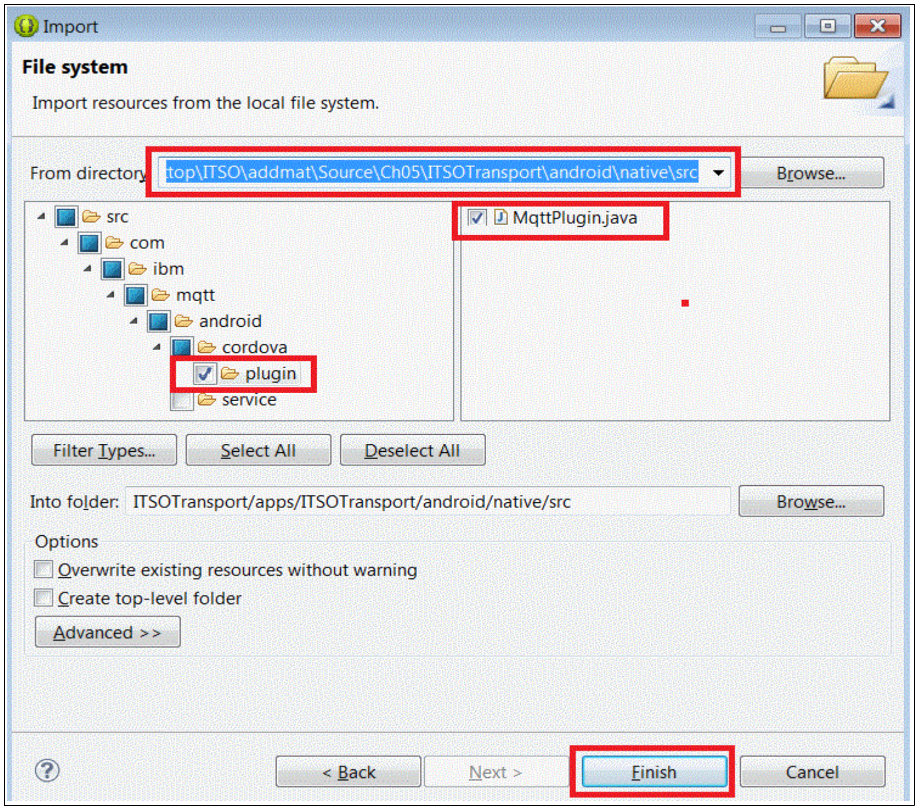


Figure 5-35 Import Java component of Cordova plug-in

8. Select **File System** and click **Next.**
9. On the next window in Import wizard, browse to the `../Ch05/ITSOTransport/android/native/src` directory where the example application is extracted.

10. Select the `plugin` folder in the left tree view, select the `MqttPlugin.Java` classes from this folder, and then click **Finish**, as shown in Figure 5-36.



From directory: top\ITSO\addmat\Source\Ch05\ITSOTransport\android\native\src

- src
  - com
    - ibm
      - mqtt
        - android
          - cordova
            - [✓] plugin
            - service

[✓] MqttPlugin.java

Into folder: ITSOTransport/apps/ITSOTransport/android/native/src

Options
- [ ] Overwrite existing resources without warning
- [ ] Create top-level folder

Figure 5-36 Import MqttPlugin.Java into the Android Worklight environment

11. After `MqttPlugin.java` is imported into the workspace, it is available in the `native/src` folder of Android, as shown in Figure 5-37.

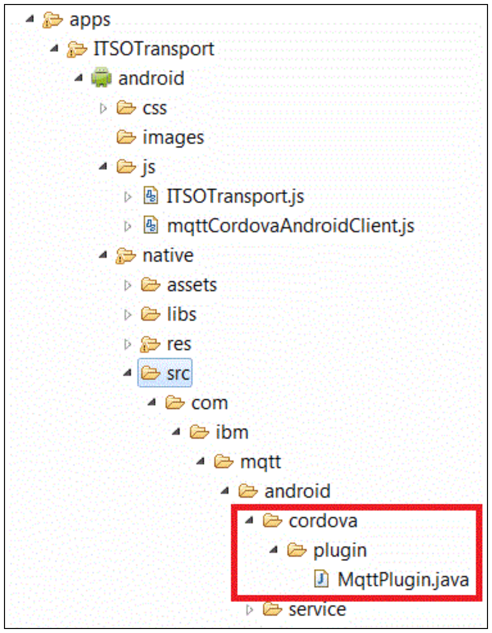


Figure 5-37 MqttPlugin.Java available in the Worklight project

5.8.5 Creating application logic and the user interface

To implement application logic and the user interface, start by importing CSS, JavaScript images, and HTML files from the downloaded example files:

1. Select the `common` folder in the Worklight environment of the ITSOTransport application.

2. Right-click it and select **import**, as shown in Figure 5-38.

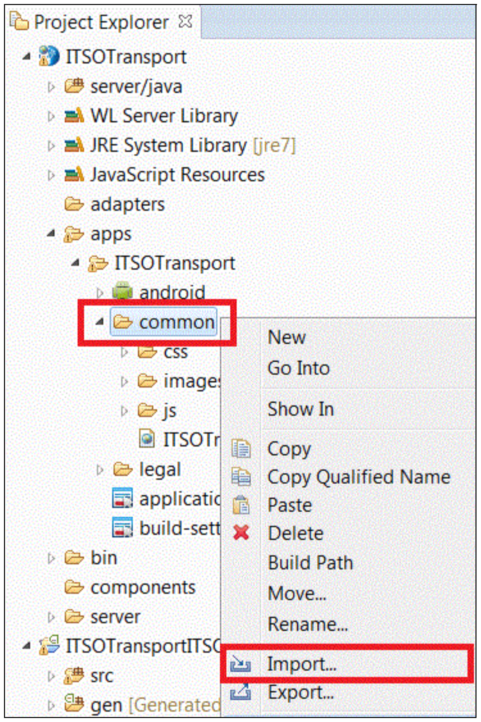
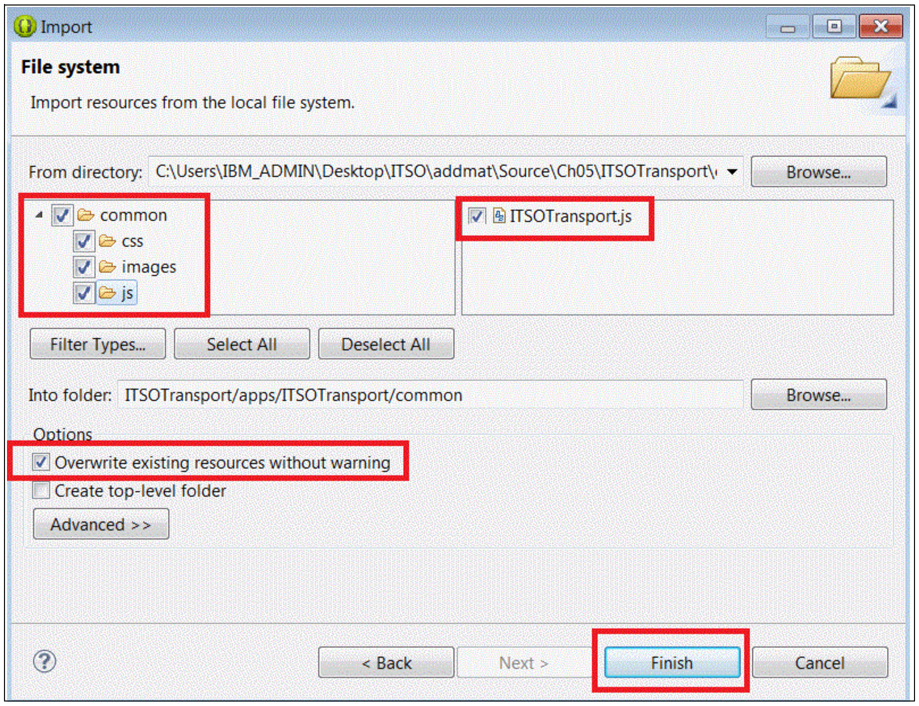


Figure 5-38   Import CSS, HTML, JavaScript, and image files

3. Select **File System** and click **Next.**
4. On the next window in the Import wizard, browse to the `../Ch05/ITSOTransport/android/common` directory where the additional material for this chapter was extracted.

5. Select the `common` folder in the left tree view to import everything. Select **Overwrite existing resources without warning**, and then click **Finish**, as shown in Figure 5-39.



Import

File system

Import resources from the local file system.

From directory: C:\Users\IBM_ADMIN\Desktop\ITSO\addmat\Source\Ch05\ITSOTransport\

- [✓] common
  - [✓] css
  - [✓] images
  - [✓] js
- [✓] ITSOTransport.js

Into folder: ITSOTransport/apps/ITSOTransport/common

Options

- [✓] Overwrite existing resources without warning
- [ ] Create top-level folder

Figure 5-39 Import CSS, HTML, JavaScript, and image files for the ITSOTransport application

6. After the HTML, JavaScript, CSS, and image files are imported, they are available in the ITSO Transport Worklight application workspace, as shown in Figure 5-40.

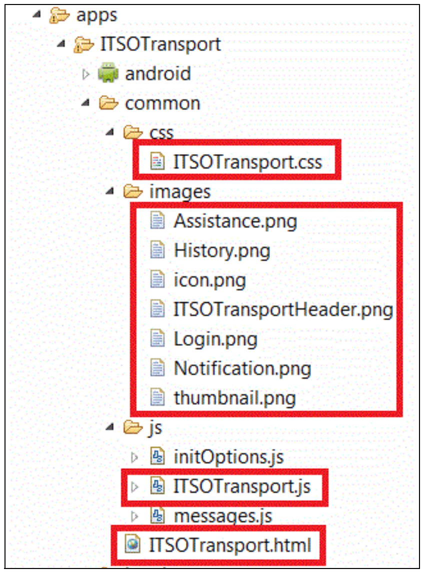


Figure 5-40   HTML, JavaScript, CSS, and image files for the ITSO Transport application

ITSOTransport CSS file

CSS is a style sheet language used for describing the presentation semantics of a document written in a markup language. CSS is designed primarily to enable the separation of document content from document presentation, including elements, such as the layout, colors, and fonts. Example 5-9 shows the contents of the `ITSOTransport.css` file.

Example 5-9   ITSOTransport.css

```
/* Reset CSS */
a, abbr, address, article, aside, audio, b, blockquote, body, canvas, caption,
cite, code, dd, del, details, dfn, dialog, div, dl, dt, em, fieldset,
figcaption, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, html,
i, iframe, img, ins, kbd, label, legend, li, mark, menu, nav, object, ol, p,
pre, q, samp, section, small, span, strong, sub, summary, sup, table, tbody,
td, tfoot, th, thead, time, tr, ul, var, video {
   margin: 0;
   padding: 0;
}

/* Worklight container div */
```

```
#content {
   height: 460px;
   margin: 0 auto;
   width: 320px;
}

form, fieldset, table, tr, td {
   background: transparent;
   border: 0 none;
   font-size: 94%;
   font-family: inherit;
   outline: 0;
   margin: 0;
   padding: 0;
   vertical-align: baseline;
}

form.tabContent {
   border: 1px solid #c9c3ba;
   padding: 0.5em;
   background-color: #f1f0ee;
}

form.tabContent.hide{
   display: none;
}
```

ITSOTransport HTML user interface

The user interface code is provided in Appendix C, “MQTT hybrid application for Android code example” on page 323.

ITSOTransport JavaScript application logic

The JavaScript application logic is provided in Appendix C, “MQTT hybrid application for Android code example” on page 323.

5.8.6 Creating the Android manifest file

Every Android application must have an `AndroidManifest.xml` file in its root directory. The manifest presents essential information about the application to the Android system. This file enables you to configure the permissions needed by the application to access platform services. To read more details about the Android manifest XML file, see the following websites:

http://developer.android.com/guide/topics/manifest/manifest-intro.html
http://developer.android.com/reference/android/Manifest.permission.html

You can import the `AndroidManifest.xml` file from the `../Ch05/ITSOTransport/android/native` directory, and overwrite the existing file in the `android/native` folder of the ITSOTransport application, as shown in Figure 5-41.

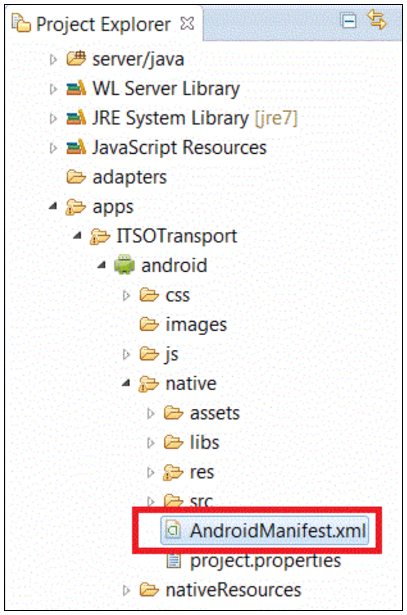


Figure 5-41 AndroidManifest file for ITSOTransport application

Example 5-10 shows the contents of the `AndroidManifest.xml` file, and the permissions defined for accessing Android services.

Example 5-10 AndroidManifest.xml file permissions

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.ITSOTransport" android:versionCode="2"
android:versionName="1.0.0.1">
<supports-screens android:smallScreens="false" android:normalScreens="true"
android:largeScreens="false" android:resizeable="false"
android:anyDensity="false"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<permission android:name="com.ITSOTransport.permission.C2D_MESSAGE"
android:protectionLevel="signature"/>
<uses-permission android:name="com.ITSOTransport.permission.C2D_MESSAGE"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

```
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.USE_CREDENTIALS"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"/>
<application android:label="@string/app_name" android:debuggable="true"
android:icon="@drawable/icon">
<activity android:name=".ITSOTransport" android:label="@string/app_name"
android:configChanges="orientation|keyboardHidden|screenSize"
android:launchMode="singleTask">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
<intent-filter>
<action android:name="com.ITSOTransport.ITSOTransport.NOTIFICATION"/>
<category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
<activity android:name="com.worklight.common.WLPreferences"
android:label="Worklight Settings"/>
<service android:name=".GCMIntentService"/>
<service android:name=".ForegroundService"/>
<receiver android:name="com.google.android.gcm.GCMBroadcastReceiver"
android:permission="com.google.android.c2dm.permission.SEND">
<intent-filter>
<action android:name="com.google.android.c2dm.intent.RECEIVE"/>
<category android:name="com.ITSOTransport"/>
</intent-filter>
<intent-filter>
<action android:name="com.google.android.c2dm.intent.REGISTRATION"/>
<category android:name="com.ITSOTransport"/>
</intent-filter>
</receiver>
<service android:name="com.ibm.mqtt.android.service.MqttService"></service>
</application>
<uses-sdk android:minSdkVersion="8"/>
</manifest>
```

5.8.7 Building and deploying the application

To build and deploy this IBM Worklight hybrid application for MQTT, right-click **ITSOTransport** and select **Run As** → **Build All and Deploy**, as shown in Figure 5-42.

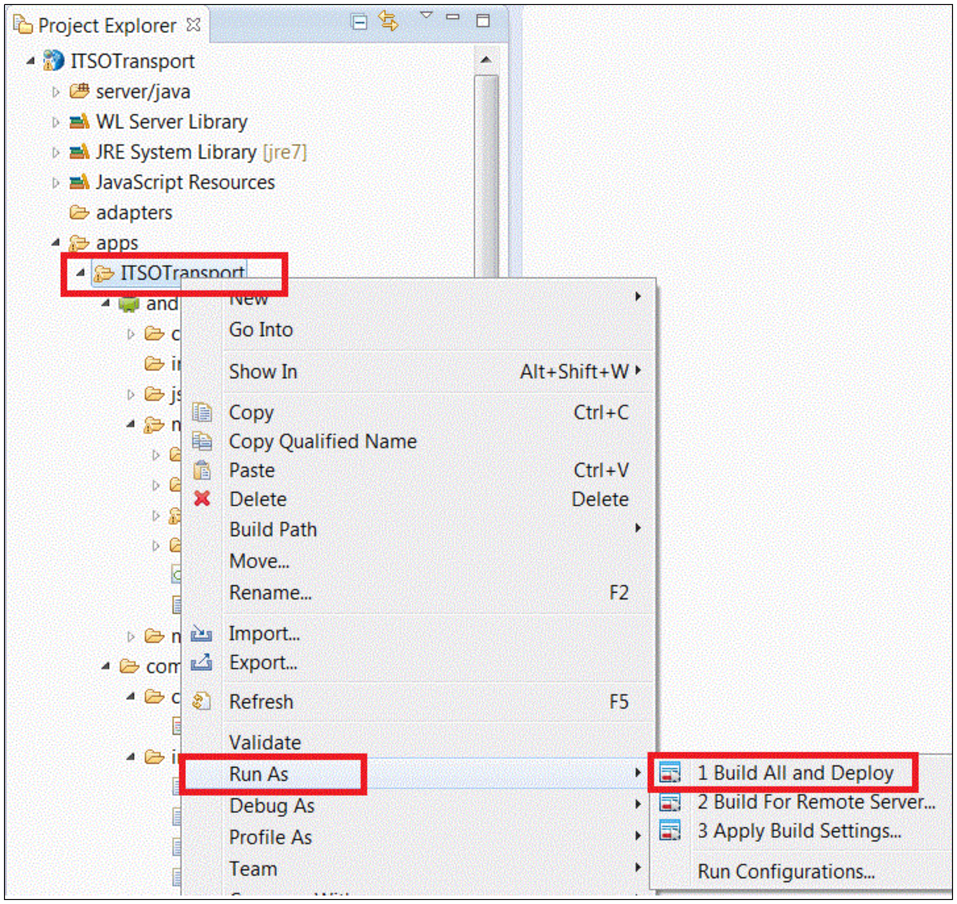


Figure 5-42 Build and deploy ITSOTransport MQTT hybrid application

Deploying an Android application using the Android debug bridge

If you do not want to build the complete project from scratch, you can use the `ITSOTransportITSOTransportAndroid.apk` binary file available in the `../Ch05/ITSOTransport/bin` directory where the additional material for this chapter was extracted.

You can deploy the binary file using the Android debug bridge (**adb**) SDK tool, which is available in the `sdk/platform-tools` directory of the SDK installation location. Example 5-11 shows the command to deploy the binary file.

Example 5-11 Deploy Android application using adb

```
C:\AndroidSDK\sdk\platform-tools>adb install
ITSOTransportITSOTransportAndroid.apk
73 KB/s (1635214 bytes in 21.663s)
pkg: /data/local/tmp/ITSOTransportITSOTransportAndroid.apk
Success
```

After the application is built and deployed, you are now ready to run and test the application.

5.8.8 Running and testing the application

To run this application, right-click the **ITSOTransportITSOTransportAndroid** project and select **Run As** → **Android Application**, as shown in Figure 5-43.

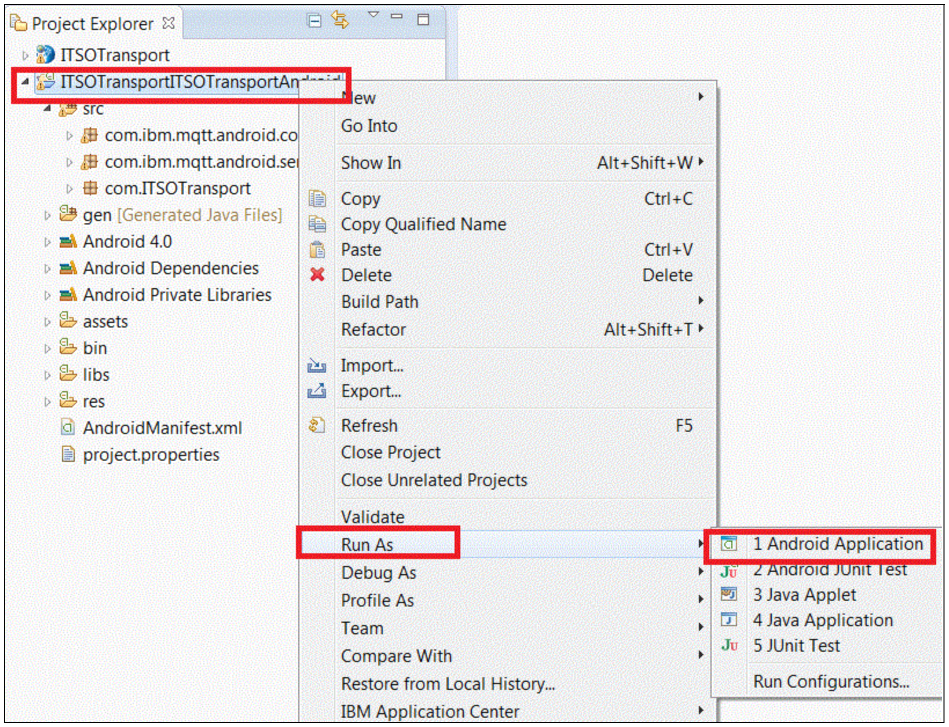


Figure 5-43 Run the ITSOTransport application

After the application is started, it opens the emulator ITSOTransportPhone created in section 5.6.4, "Configuring the Android emulator" on page 128.

Figure 5-44 shows the ITSOTransport Worklight hybrid application deployed and available in the ITSOTransportPhone emulator.

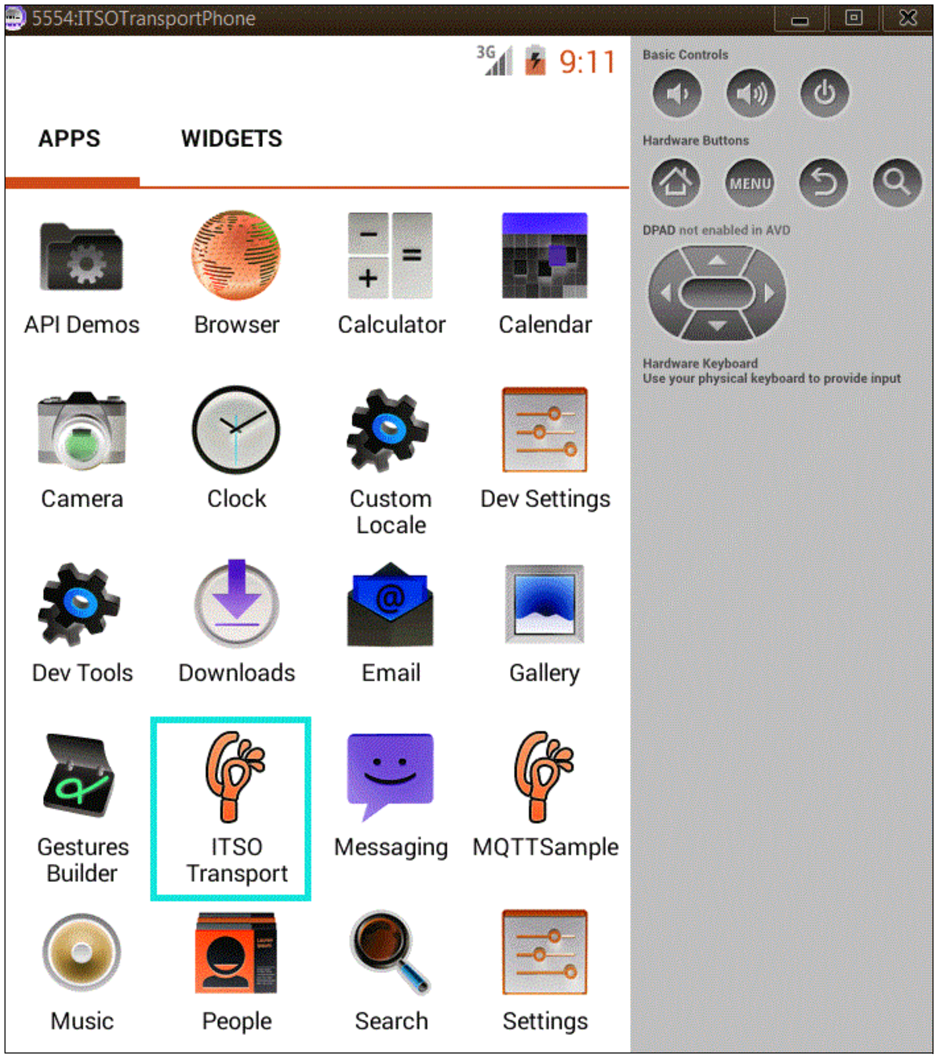
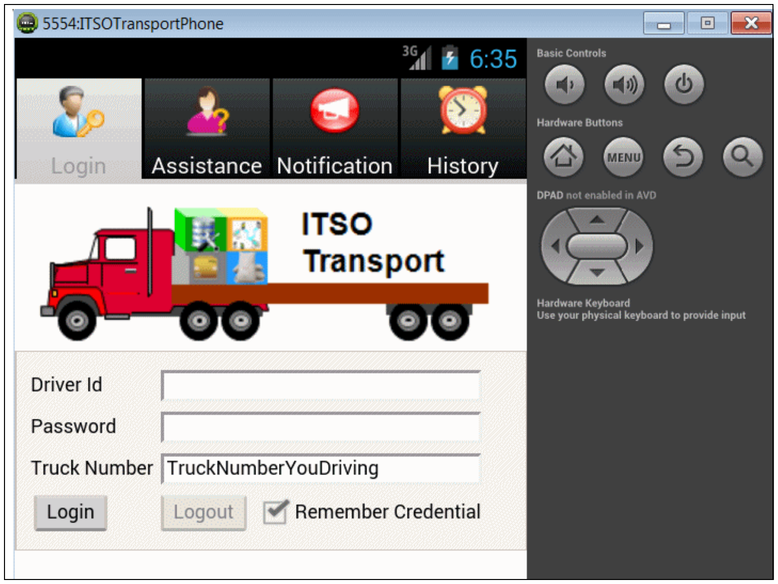


Figure 5-44 The ITSOTransport Worklight hybrid application deployed

Figure 5-45 shows the ITSOTransport Worklight hybrid application started in the ITSOTransportPhone emulator.



Driver Id:
Password:
Truck Number: TruckNumberYouDriving

Figure 5-45 The ITSOTransport application started in the Android emulator

This application provides all of the functionality described in section 5.7.4, “Application use cases” on page 133. Use case testing for this hybrid mobile application is performed in the scenario chapters of this book. Chapter 6, “Scenarios overview” on page 175, provides a brief overview of all of the scenarios described in this book.

Note: This application is based on the MQTTSample application described in the *Using MQ Telemetry Transport Protocol in IBM Worklight Mobile applications* IBM technote:

`http://www-01.ibm.com/support/docview.wss?uid=swg24033580`

You can also see this technical article to read more details about another IBM Worklight hybrid application developed for MQTT.

The Worklight projects are available for download from the IBM Redbooks website. The download instructions are in Appendix D, “Additional material” on page 341. You can import these projects into IBM Worklight studio and run the ITSOTransport MQTT application developed for IBM MessageSight.

6

Scenarios overview

This chapter provides an overview of the scenarios that are implemented in later chapters of this book. The company profile, business problems, and the application requirements are described in 5.7, “MQTT hybrid application use case and requirements” on page 130. These scenarios provide a solution to the business problems of the fictional company, and demonstrate how a mobile application can be integrated with the back-end system to achieve the required solution.

This chapter includes the following sections:

- Scenario 1 overview
- Scenario 2 overview
- Scenario 3 overview
- Scenario 4 overview

All of these scenarios assume trusted devices connecting from trusted networks. They do not explicitly cover public internet or bring your own device (BYOD) scenarios.

6.1 Scenario 1 overview

Scenario 1 uses the MQTT hybrid mobile application developed for the Android platform. This scenario describes how authentication and authorization can be achieved from IBM MessageSight.

Truck drivers log in to the mobile application before starting for a delivery, and log out when back to the station after completing the delivery. Truck drivers submit their user name, password, and truck number for authentication and authorization. These credentials are used by the mobile application to establish secure connectivity with IBM MessageSight. The truck number is used as a client identifier.

This Mobile application establishes secure connectivity with the IBM MessageSight server. To establish this connectivity, the mobile application provides driver credentials (user name, password, and truck number) for authentication and authorization.

Figure 6-1 provides an overview of this scenario. For more details about this scenario, see Chapter 7, “Scenario 1: Secure messaging” on page 183.

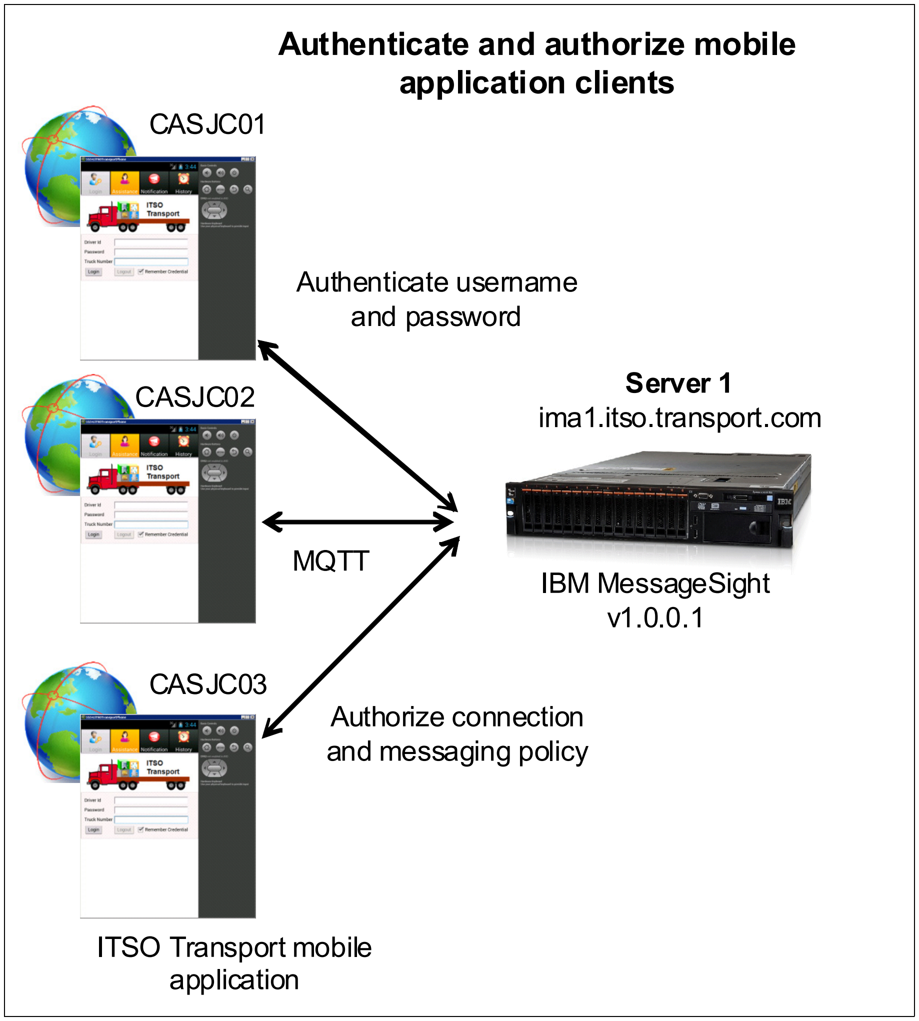

Authenticate and authorize mobile application clients
CASJC01
CASJC02
CASJC03
Authenticate username and password
Server 1
ima1.itso.transport.com
MQTT
IBM MessageSight v1.0.0.1
Authorize connection and messaging policy
ITSO Transport mobile application

Figure 6-1 Authentication and authorization for the MQTT mobile application

6.2 Scenario 2 overview

Scenario 2 uses the MQTT hybrid mobile application developed for the Android platform for drivers to send roadside assistance requests and responses through IBM MessageSight. The roadside assistance requests are sent with the type of request needed, and the accurate geographic location of truck.

The back-end application hosted on IBM Integration Bus performs the following functions:

- Receives these messages on WebSphere MQ Java Message Service (JMS)
- Maps the WebSphere MQ topics from the IBM MessageSight server to the WebSphere MQ queue manager
- Processes the assistance request and sends a corresponding response to the requester

Assistance requests and responses are sent with a highest quality of message delivery QoS2.

Figure 6-2 provides an overview of this scenario. For more details about this scenario, see Chapter 8, “Scenario 2: Request and response using MQTT” on page 201.

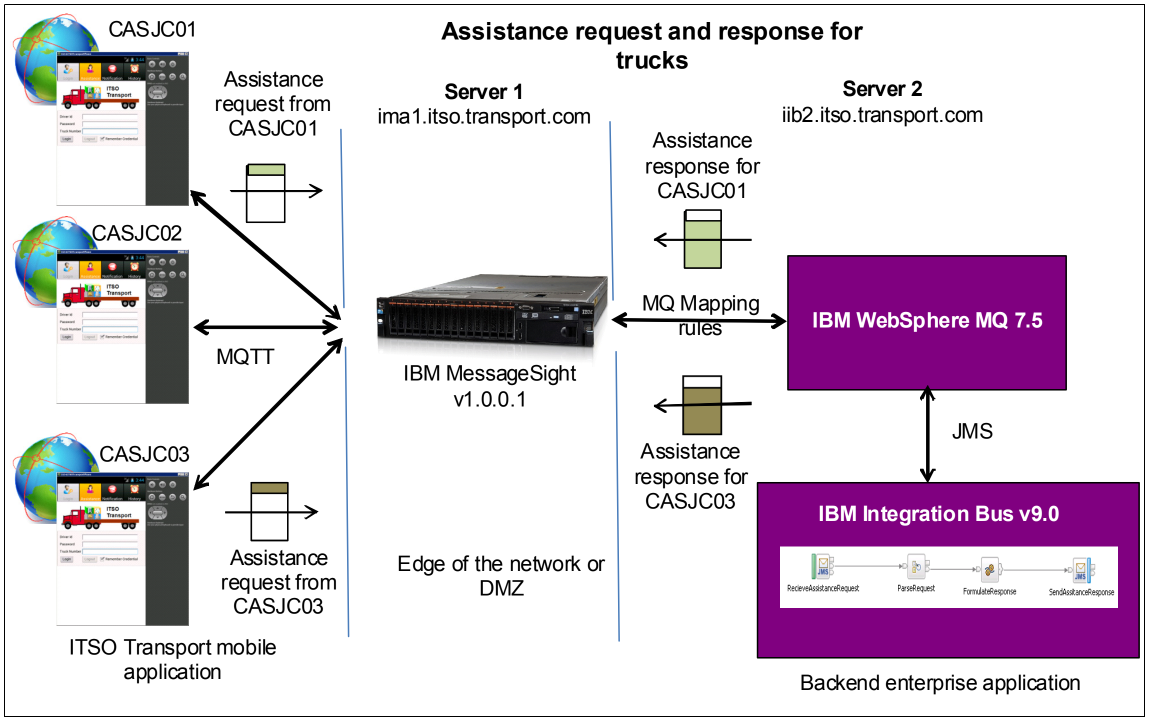

Assistance request and response for trucks
CASJC01
CASJC02
CASJC03
Assistance request from CASJC01
Assistance request from CASJC03
MQTT
Server 1
ima1.itso.transport.com
IBM MessageSight v1.0.0.1
Edge of the network or DMZ
Server 2
iib2.itso.transport.com
Assistance response for CASJC01
Assistance response for CASJC03
MQ Mapping rules
IBM WebSphere MQ 7.5
JMS
IBM Integration Bus v9.0
RecieveAssistanceRequest
ParseRequest
FormulateResponse
SendAssitanceResponse
ITSO Transport mobile application
Backend enterprise application

Figure 6-2 Roadside assistance requests for the MQTT mobile application

6.3 Scenario 3 overview

Scenario 3 uses the MQTT hybrid mobile application developed for the Android platform for truck drivers to be alerted with critical and important notifications. This scenario is divided into two sub-scenarios:

- Push notification for selected trucks based on their geographic location
- Push notification for all trucks in the fleet

6.3.1 Push notification for selected trucks based on their location

To help drivers save time and be safe on highways, the ITSO Transport company wants to develop a push notification system that analyzes trucks' current geographic location and notifies truck drivers of any problems predicted on their route to delivery. Any changes in the geographic location of the trucks are captured, based on the area that the trucks are currently in, and any critical notifications are sent to specific trucks in that area.

An example of such a notification is, "`Bay bridge is closed, plan alternative route`".

Figure 6-3 shows an example of push notification to selected trucks.

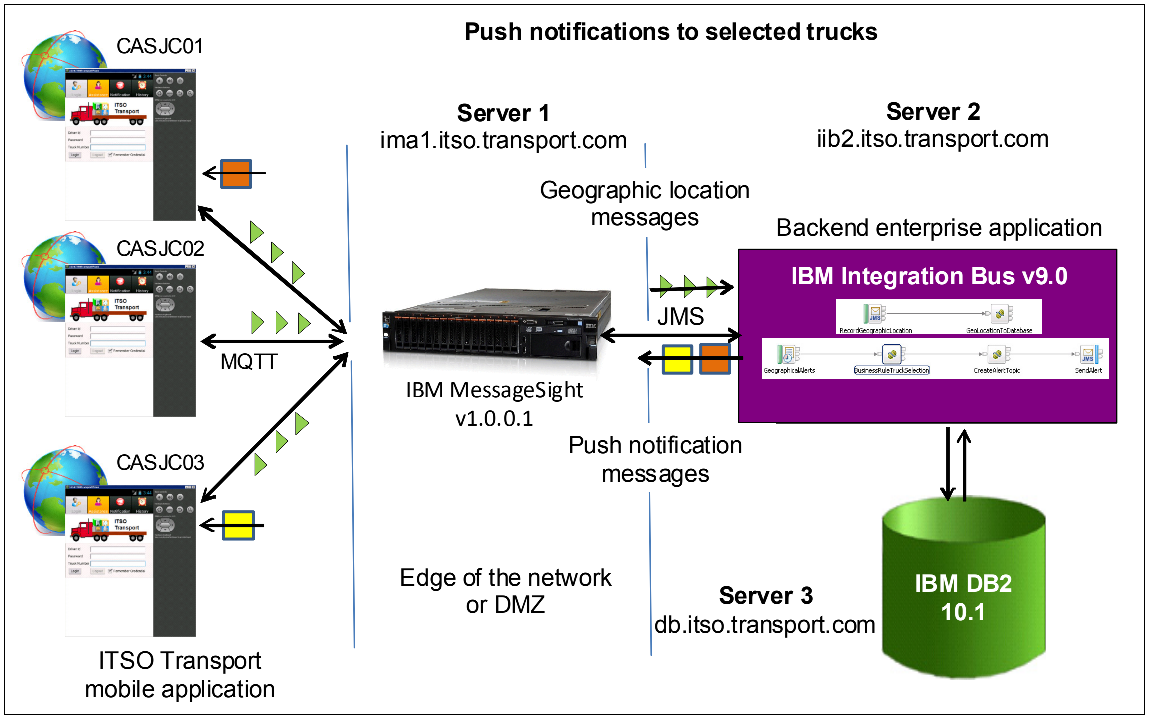


Figure 6-3 Push notification for selected trucks

6.3.2 Push notification for all trucks in the fleet

The ITSO Transport Corporation back-end office team and supervisors might sometime want to alert all of the trucks currently delivering groceries with a few important notifications.

IBM MessageSight receives publications from back-end enterprise applications or mobile applications, on specific topics, to send broadcast messages to all trucks currently delivering groceries. The broadcast message is later published to the mobile application in trucks that have subscribed on matching topic strings.

An example of this broadcast push notification is, “`Drive safely and obey state traffic laws`”.

Figure 6-4 provides an overview of a broadcast push notification.

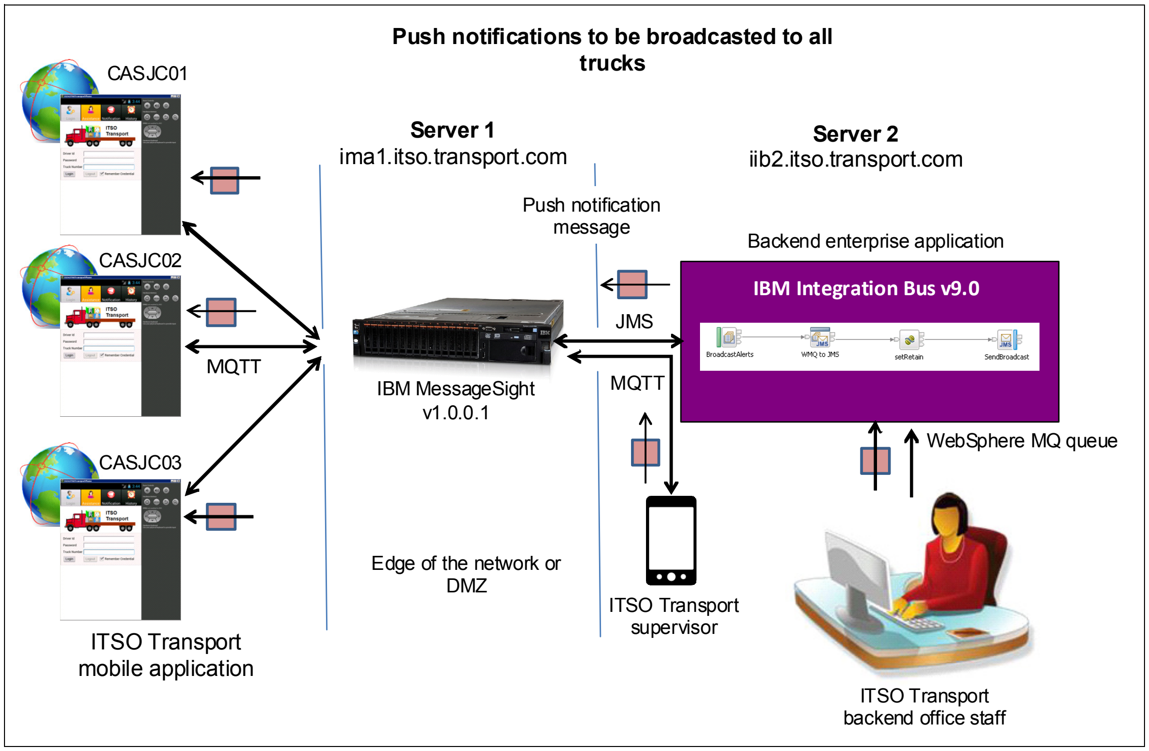

Push notifications to be broadcasted to all trucks
CASJC01
CASJC02
CASJC03
Server 1
ima1.itso.transport.com
Server 2
iib2.itso.transport.com
Push notification message
Backend enterprise application
IBM Integration Bus v9.0
BroadcastAlerts
WMQ to JMS
setRetain
SendBroadcast
JMS
MQTT
MQTT
IBM MessageSight v1.0.0.1
WebSphere MQ queue
Edge of the network or DMZ
ITSO Transport supervisor
ITSO Transport mobile application
ITSO Transport backend office staff

Figure 6-4 Push notification broadcast to all trucks

For more details about this scenario, see Chapter 9, “Scenario 3: Push notifications with quality of service” on page 225.

6.4 Scenario 4 overview

Scenario 4 uses the MQTT hybrid mobile application developed for the Android platform for drivers to send roadside assistance requests and responses through IBM MessageSight. The roadside assistance requests are sent with the type of request needed, and with the accurate geographic location of truck.

In this scenario, the back-end applications are coded as stand-alone Java implementations for MQTT and JMS, which communicate with IBM MessageSight directly. These applications process the assistance requests, and send corresponding responses to the requester. Assistance requests and responses are sent with a highest quality of message delivery QoS2.

Figure 6-5 provides an overview of this scenario.

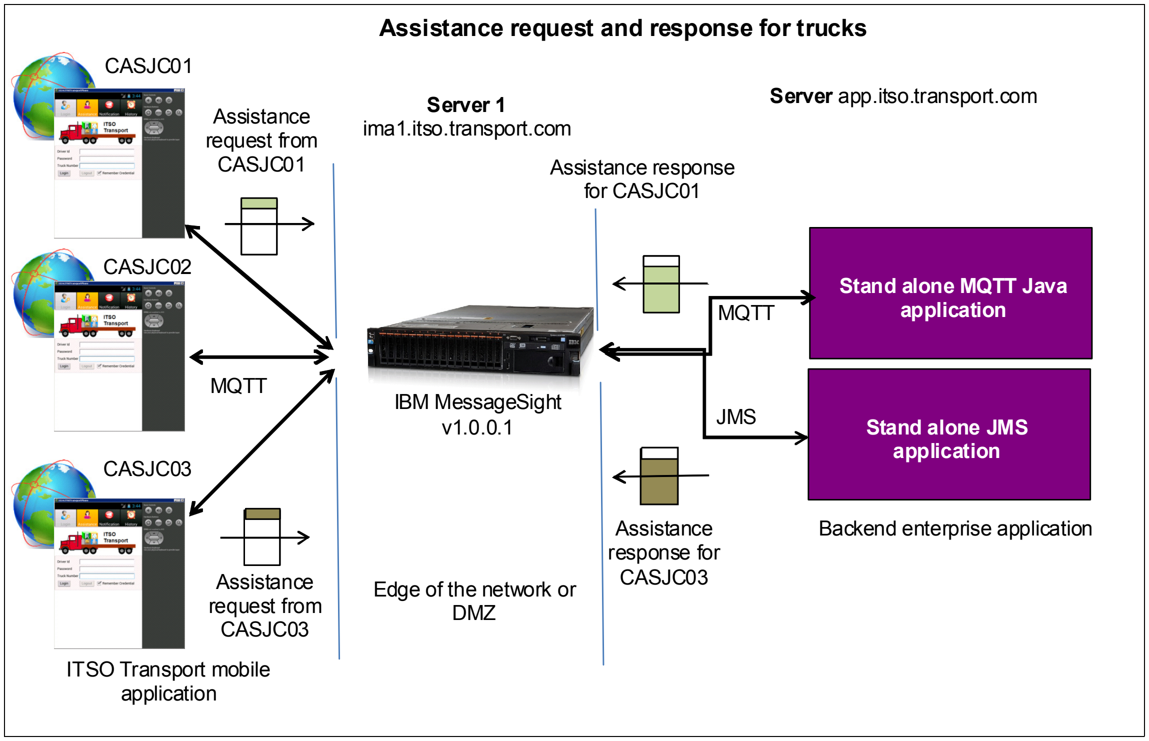

Assistance request and response for trucks
CASJC01
CASJC02
CASJC03
Assistance request from CASJC01
Assistance request from CASJC03
MQTT
Server 1
ima1.itso.transport.com
IBM MessageSight v1.0.0.1
Edge of the network or DMZ
Server app.itso.transport.com
Assistance response for CASJC01
Assistance response for CASJC03
MQTT
JMS
Stand alone MQTT Java application
Stand alone JMS application
Backend enterprise application
ITSO Transport mobile application

Figure 6-5 Roadside assistance and request for MQTT mobile application using stand-alone back-end system

For more details about this scenario, see Chapter 10, “Scenario 4: Stand-alone server applications” on page 281.

7

Scenario 1: Secure messaging

This chapter shows a scenario for secure messaging between MQTT clients and IBM MessageSight. The MQTT clients are the devices that are away from the network boundary of the enterprise. The IBM MessageSight appliance is the edge of the network server that connects to these devices.

This chapter covers the following topics:

- Scenario description
- Scenario setup
- Testing the security scenario

7.1 Scenario description

The example MQTT client application is an Android application that runs on a hand-held device. The essential functionality of this application is described in more detail in Chapter 6, “Scenarios overview” on page 175. This chapter describes the authentication and authorization of the application.

This chapter shows how only “known” devices (MQTT clients) are allowed to connect to the IBM MessageSight appliance. When connected, a given device is allowed to publish or subscribe only certain topic strings.

7.2 Scenario setup

Complete the following tasks to set up this scenario:

1. Create a group for users who can use the MQTT client.
2. Create users and passwords, and add them to the group created in step 1.
3. Create a connection policy to authenticate MQTT clients based on the client identifier, user ID, and password combination.
4. Create a messaging policy to allow publication and subscription on topic strings.

7.2.1 User group for MQTT clients

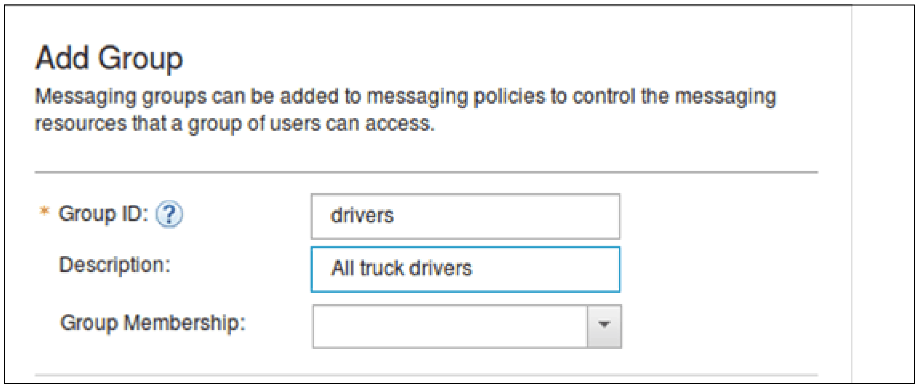
Create a user group driver for the connecting MQTT clients.

Adding a new group using the web user interface

To create users and groups using the web user interface (web UI), follow these steps:

1. Go to **Messaging** → **Users and groups**.
2. Start by creating a new group first. Under the Messaging Groups section, click the green plus (**+**) icon to add a new group.

3. In the resulting dialog box, enter details, as shown in Figure 7-1.


Add Group

Messaging groups can be added to messaging policies to control the messaging resources that a group of users can access.

* Group ID: drivers

Description: All truck drivers

Group Membership:

Figure 7-1 Adding a new group for all driver user IDs

Adding a new group using the command-line interface

Example 7-1 shows you how to create a new group using the command-line interface (CLI).

Example 7-1 Creating a new group on IBM MessageSight

```
Console> imaserver group add "GroupID=drivers" "Description=All truck
drivers"
The requested configuration change has completed successfully.
```

7.2.2 Users and passwords for MQTT clients

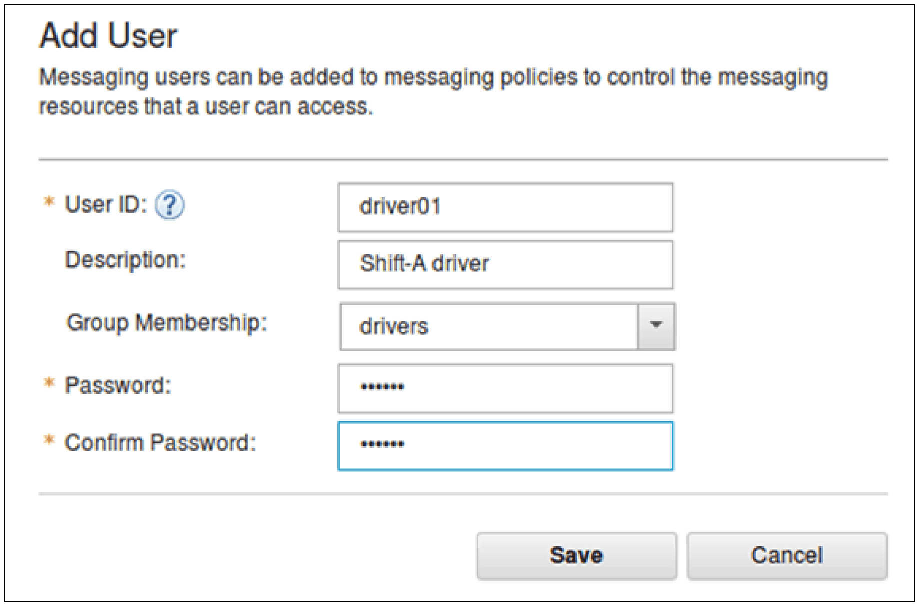
After the group is created, you are now ready to define and add users to this group.

Adding users with the web UI

To add users with the web UI, follow these steps:

1. On the same web UI page, go to the Messaging Users section and click the green plus (**+**) icon to add a new user. This user ID is a messaging user and not a web UI user.

2. In the resulting dialog box (see Figure 7-2), enter the required information to add a new user. Note that the **Group Membership** that was created earlier is set to **drivers**.



Add User

Messaging users can be added to messaging policies to control the messaging resources that a user can access.

* User ID:	driver01
Description:	Shift-A driver
Group Membership:	drivers
* Password:	••••••
* Confirm Password:	••••••

Figure 7-2 Add a new user to a group

Add users with the CLI

Example 7-2 shows how to add users using the CLI.

Example 7-2 Adding a new user on IBM MessageSight

```
Console> imaserver user add "UserID=driver01" "Type=Messaging"
"Password=new00pwd" "GroupMembership=drivers" "Description=An MQTT
client"
The requested configuration change has completed successfully.
```

Note that the type of user indicated is `Type=Messaging` to indicate that the user ID being created is a messaging user and not a web UI user.

Important: The passwords mentioned in this and subsequent examples are informational only. You ought to follow the password standards as applicable in your enterprise.

7.2.3 Connection policy for MQTT clients

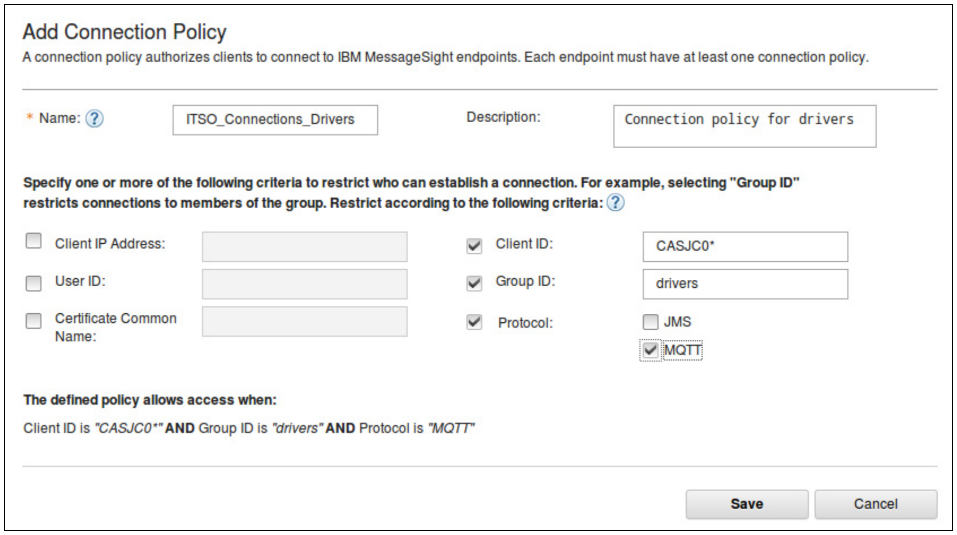
Create a connection policy based on the following rules. A given device can connect only if it meets the following requirements:

- It has a client ID among the list of client IDs considered valid for the overall system.
- It has passed a valid ID and password.

Implement these rules by setting up a criteria for client ID such that all client IDs beginning with `CASJC` are considered valid. Further, set the group ID in the connection policy to `drivers` so that the incoming connection carries a user ID and password as associated with the `drivers` group.

Note that these rules can be modified to apply for a particular combination of client ID and user ID too. This can be done by not providing the wildcard character for the client ID criterion, and providing a value for user ID rather than for the group ID.

Figure 7-3 shows how to add this connection policy using the web UI.


Add Connection Policy

A connection policy authorizes clients to connect to IBM MessageSight endpoints. Each endpoint must have at least one connection policy.

* Name: ITSO_Connections_Drivers

Description: Connection policy for drivers

Specify one or more of the following criteria to restrict who can establish a connection. For example, selecting "Group ID" restricts connections to members of the group. Restrict according to the following criteria:

☐ Client IP Address:		✓ Client ID:	CASJC0*
☐ User ID:		✓ Group ID:	drivers
☐ Certificate Common Name:		✓ Protocol:	☐ JMS ✓ MQTT

The defined policy allows access when:

Client ID is *"CASJC0*"* **AND** Group ID is *"drivers"* **AND** Protocol is *"MQTT"*

Figure 7-3 New connection policy to allow connections from trucks

Example 7-3 shows how to add the connection policy using the CLI.

Example 7-3 Create a new connection policy

```
Console> imaserver user add "UserID=driver01" "Type=Messaging"
"Password=new00pwd" "GroupMembership=drivers" "Description=An MQTT
client"
The requested configuration change has completed successfully.
```

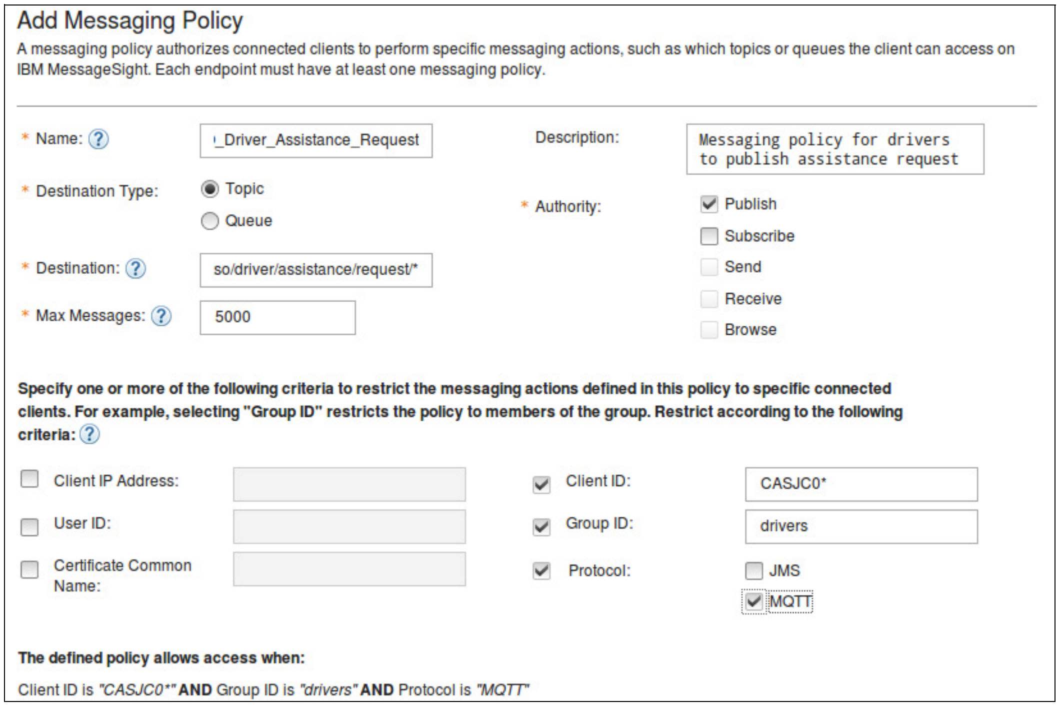
7.2.4 Messaging policy for MQTT clients

Create separate messaging policies for both the publish and subscribe actions. The aim is to limit the publications on a particular topic subtree and subscriptions as applicable to the client ID of the subscribing client.

For publishing clients, provide a topic tree that ends with a wildcard character so that the connecting client can match the type of assistance (AC malfunction, flat tire, or accident) and the level for the client ID.

For subscribing clients, use the variable substitution feature to limit subscriptions to topic strings ending with the client ID `${ClientID}`.

Figure 7-4 shows how to create a messaging policy for publishing using the web UI.



Add Messaging Policy

A messaging policy authorizes connected clients to perform specific messaging actions, such as which topics or queues the client can access on IBM MessageSight. Each endpoint must have at least one messaging policy.

* Name:	_Driver_Assistance_Request	Description:	Messaging policy for drivers to publish assistance request
* Destination Type:	[✓ Topic] Queue	* Authority:	[✓ Publish] Subscribe Send Receive Browse
* Destination:	so/driver/assistance/request/*		
* Max Messages:	5000		

Specify one or more of the following criteria to restrict the messaging actions defined in this policy to specific connected clients. For example, selecting "Group ID" restricts the policy to members of the group. Restrict according to the following criteria:

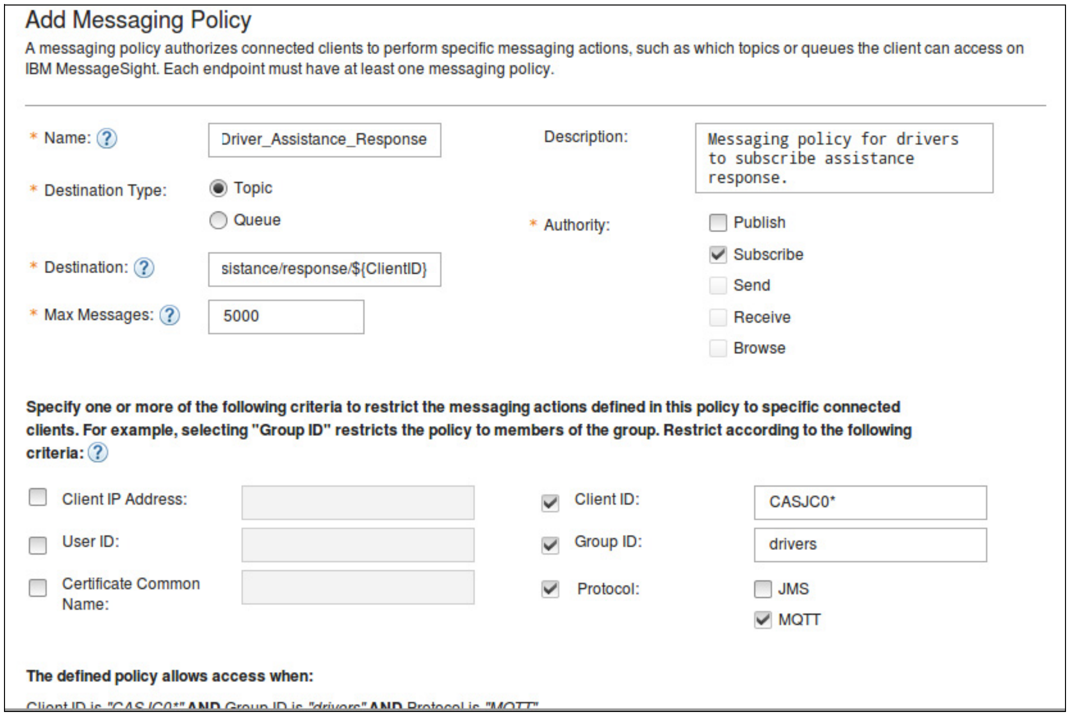
☐ Client IP Address:		✓ Client ID:	CASJC0*
☐ User ID:		✓ Group ID:	drivers
☐ Certificate Common Name:		✓ Protocol:	JMS [✓ MQTT]

The defined policy allows access when:

Client ID is *"CASJC0*"* **AND** Group ID is *"drivers"* **AND** Protocol is *"MQTT"*

Figure 7-4 Messaging policy to allow truck drivers to publish assistance requests

Figure 7-5 shows how to create a messaging policy for subscribing using the web UI.


Add Messaging Policy

A messaging policy authorizes connected clients to perform specific messaging actions, such as which topics or queues the client can access on IBM MessageSight. Each endpoint must have at least one messaging policy.

* Name:	Driver_Assistance_Response	Description:	Messaging policy for drivers to subscribe assistance response.
* Destination Type:	[✓ Topic] Queue	* Authority:	Publish [✓ Subscribe] Send Receive Browse
* Destination:	sistance/response/${ClientID}		
* Max Messages:	5000		

Specify one or more of the following criteria to restrict the messaging actions defined in this policy to specific connected clients. For example, selecting "Group ID" restricts the policy to members of the group. Restrict according to the following criteria:

Client IP Address:		✓ Client ID:	CASJC0*
User ID:		✓ Group ID:	drivers
Certificate Common Name:		✓ Protocol:	JMS [✓ MQTT]

The defined policy allows access when:

Client ID is "CASJC0*" AND Group ID is "drivers" AND Protocol is "MQTT"

Figure 7-5 Messaging policy to allow truck drivers to subscribe to assistance responses

Example 7-4 shows the CLI option to create the messaging policies for publish and subscribe, and how to list all of the messaging policies.

Example 7-4 Create new messaging policies

```
Console> imaserver create MessagingPolicy "Name=ITSO_Driver_Assistance_Request"
"Description=Messaging policy for drivers to publish assistance request"
"DestinationType=Topic" "Destination=/itso/driver/assistance/request/*"
"MaxMessages=5000" "ActionList=Publish" "ClientID=CASJC*" "GroupID=dirvers"
"Protocol=MQTT"

Console> imaserver create MessagingPolicy
"Name=ITSO_Driver_Assistance_Response" "Description=Messaging policy for
drivers to receive assistance response." "DestinationType=Topic"
"Destination=/itso/driver/assistance/response/${ClientID}" "MaxMessages=5000"
"ActionList=Subscribe" "ClientID=CASJC*" "GroupID=dirvers" "Protocol=MQTT"

Console> imaserver list MessagingPolicy
ITSO_Driver_Assistance_Request
```

```
ITSO_Driver_Assistance_Response

Console> imaserver show MessagingPolicy "Name=ITSO_Driver_Assistance_Request"
Name = ITSO_Driver_Assistance_Request
Description = Messaging policy for drivers to publish assistance request
ClientID = CASJCO*
ClientAddress =
UserID =
GroupID = drivers
CommonNames =
Protocol = MQTT
Destination = /itso/driver/assistance/request/*
DestinationType = Topic
ActionList = Publish
MaxMessages = 5000

Console> imaserver show MessagingPolicy "Name=ITSO_Driver_Assistance_Response"
Name = ITSO_Driver_Assistance_Response
Description = Messaging policy for drivers to receive assistance response.
ClientID = CASJCO*
ClientAddress =
UserID =
GroupID = drivers
CommonNames =
Protocol = MQTT
Destination = /itso/driver/assistance/response/${ClientID}
DestinationType = Topic
ActionList = Subscribe
MaxMessages = 5000
```

7.3 Testing the security scenario

To test the security scenario, first test against the connection and messaging policies. In this scenario, test if the connection policy allows connections from the known clients only. Then, test if clients can publish or subscribe on assigned topic strings.

We use the Android application for testing the constraints of the connection policy. Note that the application does not mandate the entering of credentials in the Login pane. This is done purposely to describe the constraints of the connection policy.

We use the Really Small Message Broker (RSMB) command-line MQTT client, because it enables us to demonstrate each constraint of the messaging policy.

Tip: The RSMB broker and client are available for download at the following website:

`https://www.ibm.com/developerworks/community/alphaworks/tech/rsmb`

When testing the application, you can check the log for detailed information by using the following command:

```
show log imaserver-connection.log
```

To view the log from the web UI, go to **Monitoring** → **Download logs** and click **imaserver-connection.log**. Scroll to the end to view the most recent log entry.

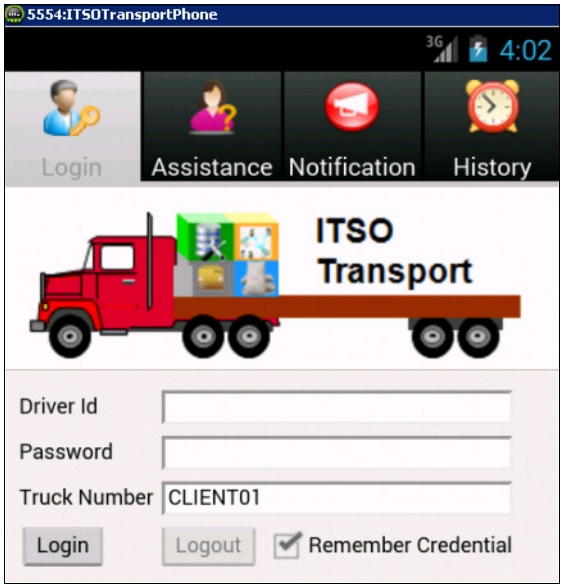
7.3.1 Testing the connection policy

Test the connection policy defined in 7.2.3, “Connection policy for MQTT clients” on page 187, where the connections are allowed only with the following conditions:

- MQTT protocol
- Client ID beginning with `CASJC`
- User names as defined in the `drivers` group
- Password for user names defined in 7.2.2, “Users and passwords for MQTT clients” on page 185

Connecting to a random client ID

Attempt a connection from the Android application with client ID (Truck Number) `CLIENT01`, as shown in Figure 7-6. This client ID does not match with the constraint defined for the client ID in the connection policy created in 7.2.3, "Connection policy for MQTT clients" on page 187).



Driver Id
Password
Truck Number CLIENT01
Login Logout ✓ Remember Credential

Figure 7-6 Connecting with an invalid client ID

The application responds with a message, such as "`The connection is not authorized`". Connect to the IBM MessageSight appliance and check the connection log, as shown in Example 7-5. Note that only the last few lines of the log are shown.

Example 7-5 Connection log when connecting with a random client ID

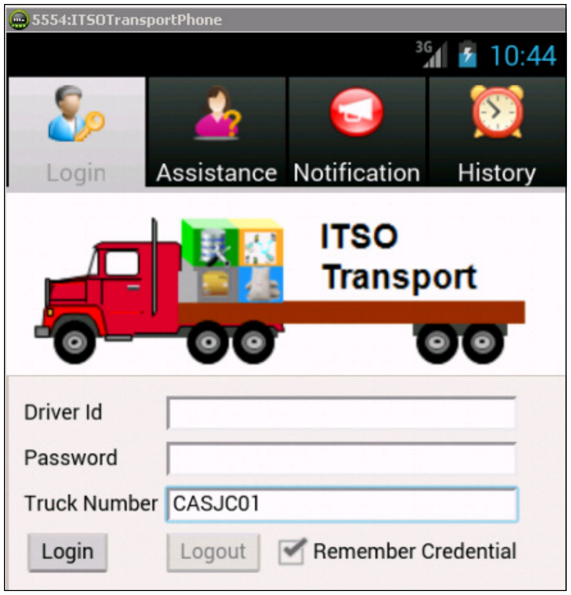
```
Console> show log imaserver-connection.log
2013-09-28T04:02:44.241+00:00 CWLNA1111 notice Connection imaserver
4475: Closing TCP connection: ConnectionID=6637 ClientID="CLIENT01"
Protocol=mqtt-tcp Endpoint="ITSO_EndPoint" UserID="" Uptime=0 RC=5
Clean=0 Reason="The connection is not authorized" ReadBytes=24
ReadMsg=0 Write-Bytes=4 WriteMsg=0 LostMsg=0.
```

As seen in the login Example 7-5, the reason clearly states, "`The connection is not authorized`". To further explore this error message, look at the connection policies in effect for the `ITSO_EndPoint` endpoint (also seen in the log).

Because the connection policy expects a client identifier beginning with `CASJC`, this connection was not authorized.

Connecting with a valid client ID but with no user name

Attempt a connection from the publishing client with a client ID (Truck Number) of `CASJC01`, as shown in Figure 7-7. This client ID matches with the constraint defined for the client ID in the connection policy. However, omit the user name (the Driver ID) and password fields, which violates the user name and password requirement in the connection policy.



Driver Id
Password
Truck Number CASJC01

Figure 7-7 Connecting with valid client ID but with no user name and password

The application responds with a message, such as "`The connection is not authorized`". Connect to the IBM MessageSight appliance and check the connection log, as shown in Example 7-6. Note that only the last few lines of the log are shown.

Example 7-6 Connection log for the IBM MessageSight appliance

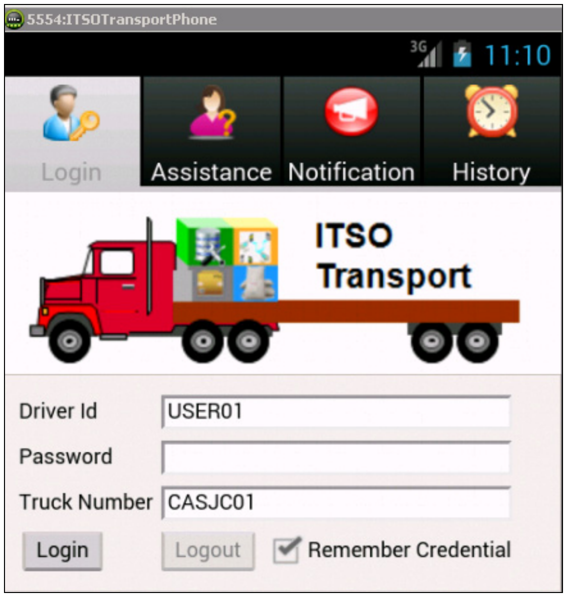
```
Console> show log imaserver-connection.log
2013-09-28T20:12:44.241+00:00 CWLNA1111 notice Connection imaserver
4475: Closing TCP connection: ConnectionID=6660 ClientID="CASJC01"
Protocol=mqtt-tcp Endpoint="ITSO_EndPoint" UserID="" Uptime=0 RC=5
Clean=0 Reason="The connection is not authorized" ReadBytes=23
ReadMsg=0 WriteBytes=4 WriteMsg=0 LostMsg=0.
```

As seen in Example 7-6 on page 194, the reason clearly states, "`The connection is not authorized`". To further explore this error message, look at the connection policies in effect for the `ITSO_EndPoint` endpoint (also seen in the log).

Because the connection policy has an additional constraint where the user name must belong to the `drivers` group, this connection was not authorized. Note that the `UserID` in the log is not set to any value.

Connecting with a valid client ID but with an invalid user name

Attempt a connection from the publishing client with client ID (Truck Number) of `CASJC01`, as shown in Figure 7-8. This client ID matches with the constraint defined for the client ID in the connection policy. Also provide a user name (Driver ID) that is *not* defined in the `drivers` group. This is a violation of the connection policy, because it requires a user name that is defined in the `drivers` group.



Driver Id: USER01
Password:
Truck Number: CASJC01
[✓] Remember Credential

Figure 7-8 Connecting with valid client ID with invalid user name

The application responds with a message, such as “`The connection is not authorized`”. Connect to the IBM MessageSight appliance and check the connection log, as shown in Example 7-7. Note that only the last few lines of the log are shown.

Example 7-7 Connection logon IBM MessageSight appliance

```
Console> show log imaserver-connection.log
2013-09-28T20:22:44.241+00:00 CWLNA1111 notice Connection imaserver
4475: Closing TCP connection: ConnectionID=6674 ClientID="CASJC01"
Protocol=mqtt-tcp Endpoint="ITSO_EndPoint" UserID="USER01" Uptime=0
RC=5 Clean=0 Reason="The connection is not authorized" ReadBytes=32
ReadMsg=0 WriteBytes=4 WriteMsg=0 LostMsg=0.
```

As seen in Example 7-7, the reason clearly states, “`The connection is not authorized`”. To further explore this error message, look at the connection policies in effect for the `ITSO_EndPoint` endpoint.

Because the connection policy expects a client identifier beginning with `CASJC` and a user name belonging to the `drivers` group, this connection was not authorized. Note that the `UserID` in the log is set to `USER01`.

Connecting with a valid client ID and user name but an invalid password

Attempt a connection from the publishing client using a client ID of `CASJC01` and user name of `driver01`, but with an invalid password, as shown in Figure 7-9 on page 197.

This client ID and user name match with the constraint defined for the client ID and user name (through Groups) in the connection policy. However, the password does not match. This causes a violation of the connection policy, because the user name is not associated with a password as defined for the user `driver01`.

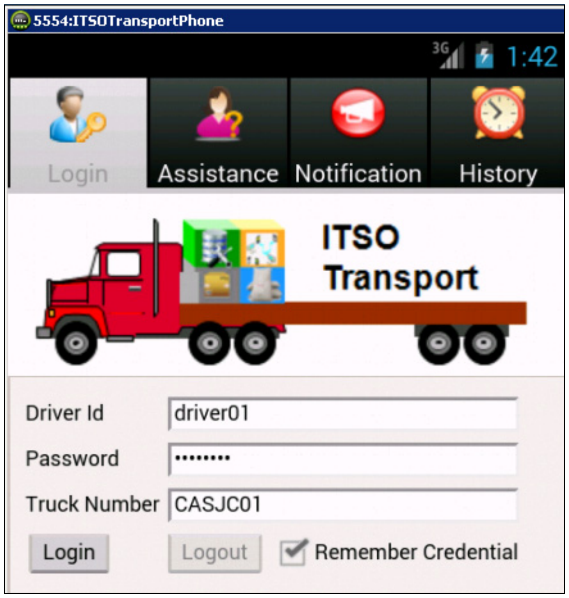


Figure 7-9 Connecting with a valid client ID and user name, but invalid password

The application responds with a message, such as “`The connection is not authorized`”. Connect to the IBM MessageSight appliance and check the connection log, as shown in Example 7-8. Note that only the last few lines of the log are shown.

Example 7-8 Connection logon IBM MessageSight appliance

```
2013-09-27T07:50:18.864+00:00 CWLNA1111 notice Connection imaserver
4475: Closing TCP connection: ConnectionID=6698 ClientID="CASJC01"
Protocol=mqtt-tcp Endpoint="ITSO_EndPoint" UserID="driver01" Uptime=0
RC=5 Clean=0 Reason="The connection is not authorized" ReadBytes=43
ReadMsg=0 WriteBytes=4 WriteMsg=0 LostMsg=0.
```

As seen in Example 7-8, the reason clearly states, “`The connection is not authorized`”. To further explore this error message, look at the connection policies in effect for the `ITSO_EndPoint` endpoint (also seen in the log).

Because the connection policy expects a client identifier beginning with `CASJC` and a user name with a valid password belonging to the `drivers` group, this connection was not authorized. Note that the `UserID` in the log is set to `driver01`.

Connecting with a valid client ID, user name, and password

To completely match with the constraints as defined in the connection policy, use the following inputs when connecting:

- Client ID of `CASJC01`
- User name of `driver01`
- Password of `dr!ver@1`

Attempt a connection with a publishing client with the previously mentioned values, and connect to the application successfully.

7.3.2 Testing messaging policy for publishing clients

In this scenario, the messaging policy for publishing is set so that the publications are allowed only with the following conditions:

- MQTT protocol
- Client ID beginning with `CASJC`
- User names as defined in the group drivers
- Topic string matches `itso/driver/assistance/request`

In the following sections, we assume that the connection is established successfully, because the connection policy was successfully applied.

Publishing to a random topic string

Attempt a publication to a topic string of your choice, as shown in Example 7-9. Note that the password is provided as `dr\!ver@1` (with a backslash preceding the exclamation mark (**!**) to "escape" it) because the exclamation mark refers to command history on a UNIX system.

Example 7-9 Publish to a random topic string

```
nsubrahm@nsubrahm:~/linux_ia64$ LD_LIBRARY_PATH=. ./stdinpub
/some/topic/string --host 9.12.5.191 --port 16105 --clientid CASJC01
 --username driver01 --password dr\!ver@1
Using topic /some/topic/string

Connecting

Hello! This is a test message
```

The **--host** parameter specifies the IP address of the IBM MessageSight appliance, and the **--port** parameter specifies the endpoint where the appliance is listening to MQTT protocol.

Remember: The client does not report a publication failure. This is because, according to the MQTT protocol specification, the client will not be informed of a publication failure due to authorization.

For more information about the PUBLISH message in MQTT, see the MQTT protocol specification for PUBLISH at the following website:

http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#publish

Connect to the IBM MessageSight appliance and look at the connection log, as shown in Example 7-10.

Example 7-10 Connection logon IBM MessageSight appliance

```
Console> show log imaserver-connection.log
…
2013-09-27T11:32:17.504+00:00 CWLNA2106 warning Connection imaserver
4475: Unable to send a message due to an authorization failure:
ConnectionID=6910 ClientID="CASJC01" Protocol="mqtt-tcp"
Endpoint="ITSO_EndPoint" UserID="driver01".
```

As seen in Example 7-10, the reason states, "`Unable to send a message due to an authorization failure`". To further explore this error message, look at the messaging policies in effect for the `ITSO_EndPoint` endpoint (also seen in the log). Because the messaging policy expects publication on topic strings that match `/itso/driver/assistance/request/*`, the publication has failed.

Subscribing to a random topic string

Attempt a subscription to a topic string, as shown in Example 7-11. Note that the password is provided as `dr\!ver@1` (with a backslash preceding the exclamation mark (!) to "escape" it) because the exclamation mark refers to command history on a UNIX system.

Example 7-11 Subscribe to a random topic string

```
nsubrahm@nsubrahm:~/linux_ia64$ LD_LIBRARY_PATH=. ./stdoutsub
/some/topic/string --host 9.12.5.191 --port 16105 --clientid CASJC01
 --username driver01 --password dr\!ver@1
Using topic /some/topic/string
```

The **--host** parameter specifies the IP address of the IBM MessageSight appliance and the **--port** parameter specifies the endpoint where the appliance is listening to MQTT protocol.

Remember: The client does not report a subscription failure. This is because, according to the MQTT protocol specification, the client will not be informed of a subscription failure due to authorization.

For more information about the SUBSCRIBE message in MQTT, see the MQTT protocol specification for SUBSCRIBE at the following website:

http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#subscribe

Connect to the IBM MessageSight appliance and look at the connection log, as shown in Example 7-12.

Example 7-12 Connection logon IBM MessageSight appliance

```
Console> show log imaserver-connection.log
…
2013-09-27T11:54:14.725+00:00 CWLNA2202 warning Connection imaserver
4475: Unable to create a consumer due to an authorization failure:
ConnectionID=6954 ClientID="CASJC01" Protocol="mqtt-tcp"
End-point="ITSO_EndPoint" UserID="driver01".
```

As seen in Example 7-12, the reason states, “`Unable to create a consumer due to an authorization failure`”. To further explore this error message, look at the messaging policies in effect for the `ITSO_EndPoint` endpoint (also seen in the log). Because the messaging policy expects subscription on topic strings that match `/itso/driver/assistance/response/${ClientID}`, the subscription has failed.

8

Scenario 2: Request and response using MQTT

This chapter shows a scenario of request and response between an Android-based device, IBM MessageSight, and IBM Integration Bus. The device sends out a request by publishing a request message. This message is sent out as an MQTT message, and a response message is delivered to the Android-based device as an MQTT message.

This chapter covers the following topics:

- Scenario description
- Scenario setup
- Scenario execution
- Scenario testing

8.1 Scenario description

Consider an Android application running on a hand-held device as the MQTT client application. This hand-held device is onboard the trucks in the fleet of the ITSO Transport Corporation company. This hand-held device is expected to be used by a driver (when the truck is stopped) to request assistance in case of difficulty.

Chapter 6, “Scenarios overview” on page 175 describes the scenario application used in this book in more detail. This chapter describes the scenario for the request and response portion of the application.

From a user’s perspective, the hand-held device has options to send out a request for assistance when the driver faces a difficult situation. The response is delivered from the back-end system to address the driver's difficulty as soon as possible.

For example, the truck might carry perishable goods that are kept at a constant temperature by the air-conditioning onboard the truck. At some point, the driver might detect that the air-conditioning is unable to maintain the required temperature.

The driver then sends out a request for assistance with the malfunctioning air conditioner. The back-end system can then dispatch an air-conditioner mechanic to a location nearest to the location from where the request originated.

For the specific request, the back-end system (the enterprise system of ITSO Transport Corporation) delivers the response to the truck that requested assistance. This is different from the scenario where the back-end system broadcasts general information about routes, weather, goods delivery, and so on to all of the trucks in the fleet.

8.2 Scenario setup

We assume that the Android application is already up and running, either on an emulator or a real device. We also assume that a connection between IBM MessageSight and WebSphere MQ already exists. For more information about how to set up the WebSphere MQ connection, see 3.1, “WebSphere MQ connectivity and destination mapping” on page 40.

Tip: For more information about running an Android application on a real device or Eclipse IDE (with Android plug-ins), see the *Running Your App* topic at the following website:

`http://developer.android.com/training/basics/firstapp/running-app.html`

Complete the following tasks to set up this scenario:

1. Define and save the Java Message Service (JMS) bindings file to a location to which the integration server has access.
2. Define the destination mapping rules on the IBM MessageSight appliance so that the publications on MQTT and JMS topic strings are exchanged automatically.
3. Deploy the IBM Integration Bus application to an integration node on an integration server.

8.2.1 JMS bindings file

The JMS bindings file can be developed using WebSphere MQ Explorer. Alternatively, you can use the bindings file that was developed for the application. In either option, the following conventions must be followed:

- Name of the connection factory

  This is the name to be used in the JMS nodes of the message flow.

- Destination type

  The type of destination will be the topic, because the destination mapping rule in the IBM MessageSight appliance maps MQTT and JMS topic subtrees.

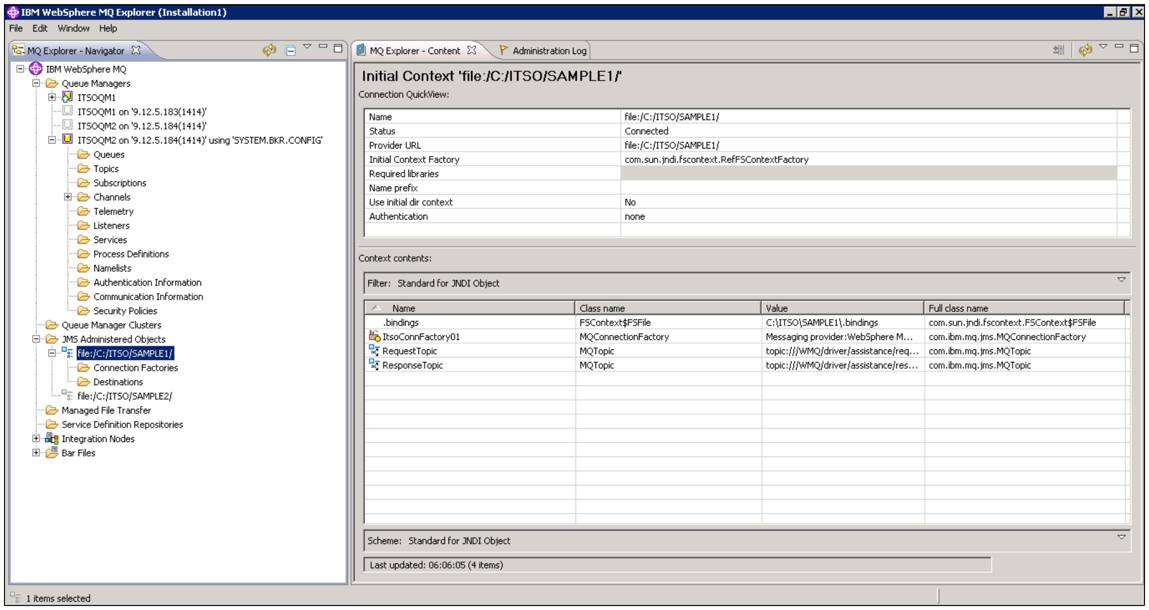
- Destination name

  The name of the destinations must match with the value used in the JMS nodes for input and output.

- Topic strings

  The topic strings must match with the ones used in the destination mapping rule on the IBM MessageSight appliance. The destination mapping rule maps an IBM MessageSight topic subtree and a WebSphere MQ topic subtree, which is then subscribed using JMS in WebSphere Message Broker.

In this application, the JMS bindings file was developed using WebSphere MQ Explorer. Figure 8-1 shows a JMS-administered object, and the bottom half shows the connection factory and topic strings.



```
IBM WebSphere MQ
  Queue Managers
    ITSOQM1
    ITSOQM1 on '9.12.5.183(1414)'
    ITSOQM2 on '9.12.5.184(1414)'
    ITSOQM2 on '9.12.5.184(1414)' using 'SYSTEM.BKR.CONFIG'
      Queues
      Topics
      Subscriptions
      Channels
      Telemetry
      Listeners
      Services
      Process Definitions
      Namelists
      Authentication Information
      Communication Information
      Security Policies
  Queue Manager Clusters
  JMS Administered Objects
    file:/C:/ITSO/SAMPLE1/
      Connection Factories
      Destinations
    file:/C:/ITSO/SAMPLE2/
  Managed File Transfer
  Service Definition Repositories
  Integration Nodes
  Bar Files
```

Initial Context 'file:/C:/ITSO/SAMPLE1/'

Connection QuickView:

Name	file:/C:/ITSO/SAMPLE1/
Status	Connected
Provider URL	file:/C:/ITSO/SAMPLE1/
Initial Context Factory	com.sun.jndi.fscontext.RefFSContextFactory
Required libraries	
Name prefix	
Use initial dir context	No
Authentication	none

Context contents:

Filter: Standard for JNDI Object

Name	Class name	Value	Full class name
.bindings	FSContext$FSFile	C:\ITSO\SAMPLE1\.bindings	com.sun.jndi.fscontext.FSContext$FSFile
ItsoConnFactory01	MQConnectionFactory	Messaging provider:WebSphere M...	com.ibm.mq.jms.MQConnectionFactory
RequestTopic	MQTopic	topic:///WMQ/driver/assistance/req...	com.ibm.mq.jms.MQTopic
ResponseTopic	MQTopic	topic:///WMQ/driver/assistance/res...	com.ibm.mq.jms.MQTopic

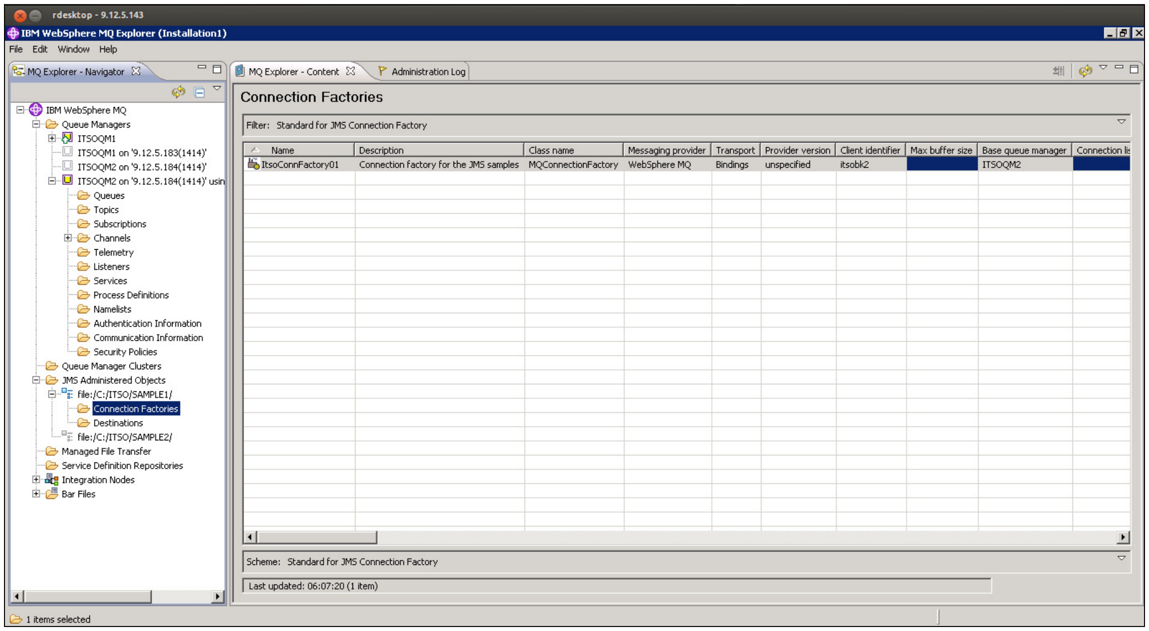
Scheme: Standard for JNDI Object

Last updated: 06:06:05 (4 items)

1 items selected

Figure 8-1 JMS-administered objects defined in WebSphere MQ Explorer

Figure 8-2 shows some of the details of the connection factory as defined in the JMS object. The name of the connection factory shown here must match with the name of the connection factory provided in the JMS nodes in the message flow.


Connection Factories

Filter: Standard for JMS Connection Factory

Name	Description	Class name	Messaging provider	Transport	Provider version	Client identifier	Max buffer size	Base queue manager	Connection lis
ItsoConnFactory01	Connection factory for the JMS samples	MQConnectionFactory	WebSphere MQ	Bindings	unspecified	itsobk2		ITSOQM2	

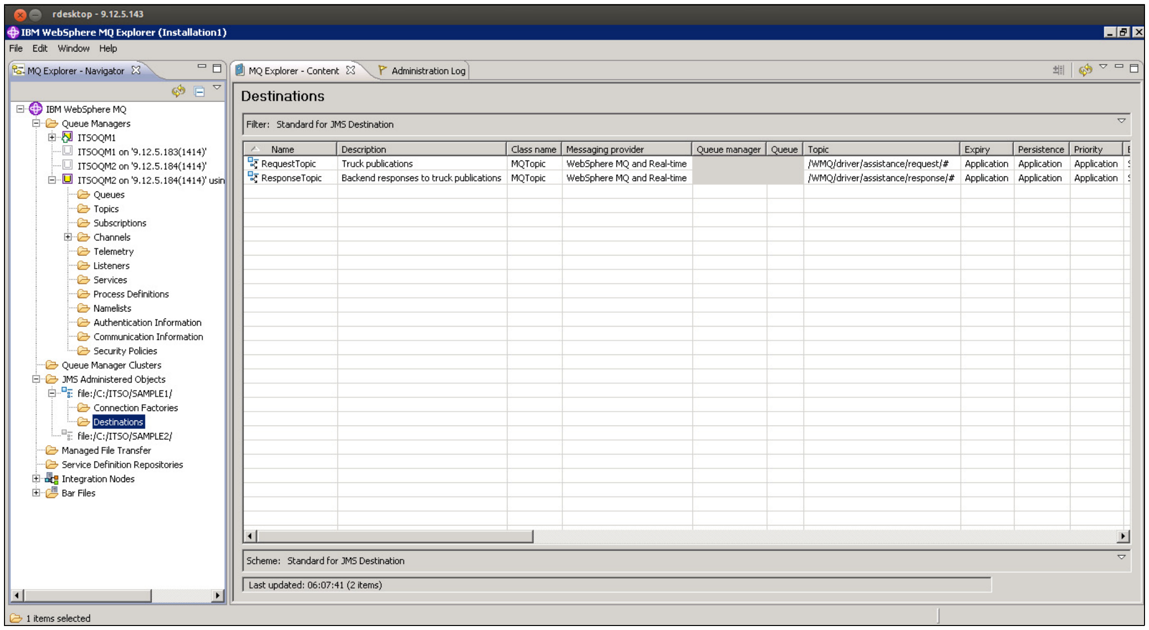
Scheme: Standard for JMS Connection Factory

Last updated: 06:07:20 (1 item)

1 items selected

Figure 8-2 Connection factory of the JMS-administered object

Figure 8-3 shows the topic definitions for the connection factory shown in Figure 8-2 on page 205. The name of the topics must match with the value entered in the JMS nodes. Similarly, the topic string shown in Figure 8-3 must match with the value of the topic string as provided in the IBM MessageSight destination mapping rule.



Destinations

Filter: Standard for JMS Destination

Name	Description	Class name	Messaging provider	Queue manager	Queue	Topic	Expiry	Persistence	Priority
RequestTopic	Truck publications	MQTopic	WebSphere MQ and Real-time			/WMQ/driver/assistance/request/#	Application	Application	Application
ResponseTopic	Backend responses to truck publications	MQTopic	WebSphere MQ and Real-time			/WMQ/driver/assistance/response/#	Application	Application	Application

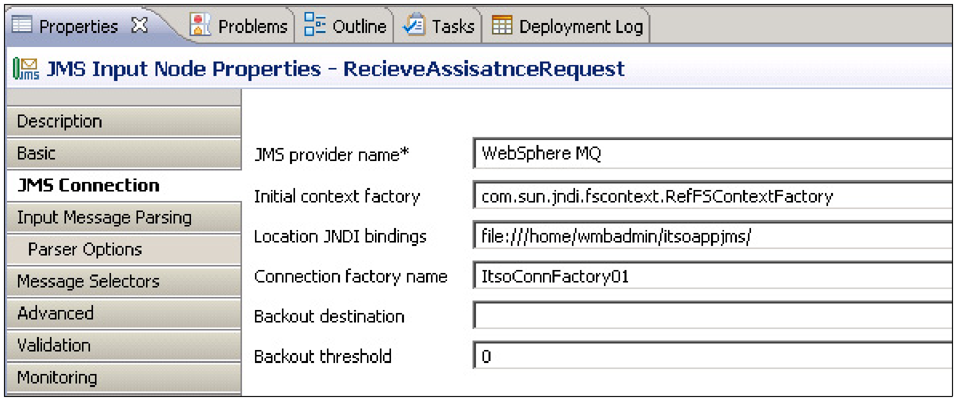
Scheme: Standard for JMS Destination

Last updated: 06:07:41 (2 items)

Figure 8-3 Definitions of topics name and strings

After all of the definitions are complete, this bindings file needs to be saved at a location that the integration server has access to. Note that this location must match with the value set for the JMS nodes in the IBM Integration Bus message flow.

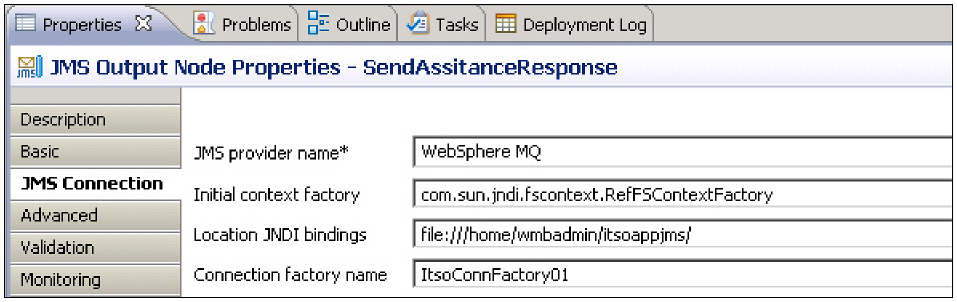
Figure 8-4 shows the values set for the JMS Connection tab for the `JMSInput` node. Note that the name of the connection factory matches with the one shown in Figure 8-2 on page 205. The location of the bindings file needs to be a location that the Integration server has access to.



JMS provider name*	WebSphere MQ
Initial context factory	com.sun.jndi.fscontext.RefFSContextFactory
Location JNDI bindings	file:///home/wmbadmin/itsoappjms/
Connection factory name	ItsoConnFactory01
Backout destination	
Backout threshold	0

Figure 8-4 Values set for the JMS Connection of the JMSInput node

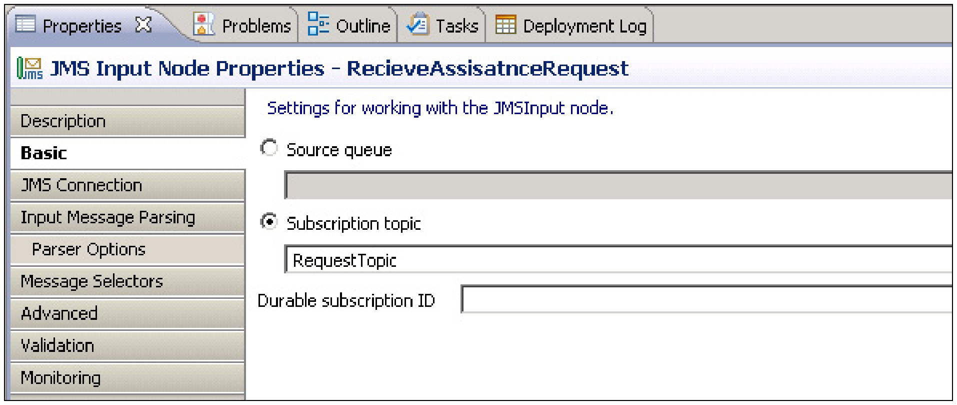
Figure 8-5 shows the values for JMS Connection as set for the `JMSOutput` node.



JMS provider name*	WebSphere MQ
Initial context factory	com.sun.jndi.fscontext.RefFSContextFactory
Location JNDI bindings	file:///home/wmbadmin/itsoappjms/
Connection factory name	ItsoConnFactory01

Figure 8-5 Values set for the JMS Connection of the JMSOutput node

Figure 8-6 shows the values set on the Basic tab for the `JMSInput` node. Note that the name of the request topic matches with the one shown in Figure 8-3 on page 206.


Properties | Problems | Outline | Tasks | Deployment Log

JMS Input Node Properties - RecieveAssisatnceRequest

Description
Basic
JMS Connection
Input Message Parsing
Parser Options
Message Selectors
Advanced
Validation
Monitoring

Settings for working with the JMSInput node.

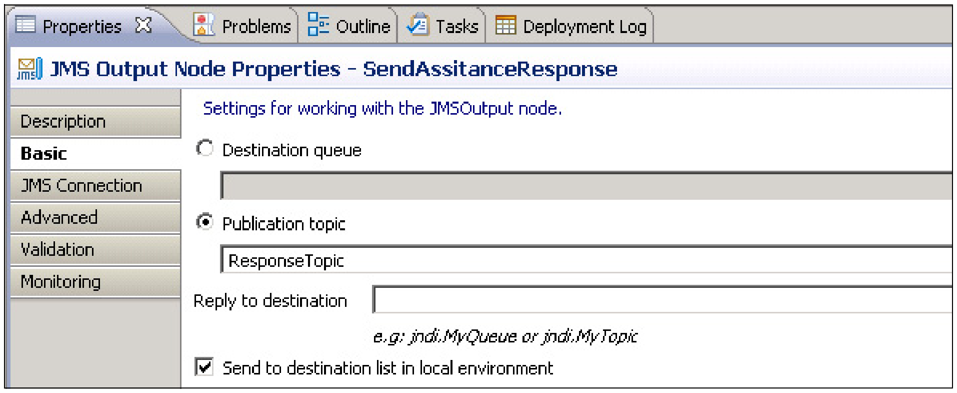
Source queue [ ]

[✓ Subscription topic] RequestTopic

Durable subscription ID [ ]

Figure 8-6 Values set for Basic tab of the JMSInput node

Figure 8-7 shows the values set on the Basic tab for the `JMSOutput` node.


Properties | Problems | Outline | Tasks | Deployment Log

JMS Output Node Properties - SendAssitanceResponse

Description
Basic
JMS Connection
Advanced
Validation
Monitoring

Settings for working with the JMSOutput node.

Destination queue [ ]

[✓ Publication topic] ResponseTopic

Reply to destination [ ]

e.g: jndi.MyQueue or jndi.MyTopic

☑ Send to destination list in local environment

Figure 8-7 Values set for Basic tab of the JMSOutput node

8.2.2 Destination mapping rules between IBM MessageSight and WebSphere MQ

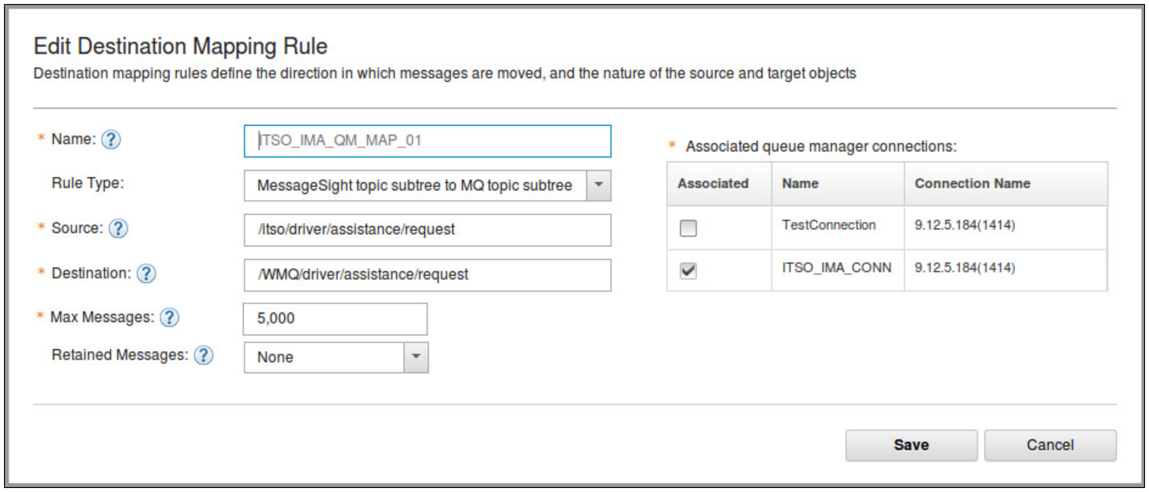
For this scenario, the destination mapping rule between IBM MessageSight and WebSphere MQ must be set with the following values, as shown in Table 8-1.

Table 8-1 Values for destination mapping rule to WebSphere MQ

Property name	Value	Remarks
Name	ITSO_IMA_QM_MAP_01	This is the name of the destination mapping rule.
Rule type	IBM MessageSight topic subtree to WebSphere MQ topic subtree	The source and destination are both topics because the MQTT client publishes onto topics on the IBM MessageSight appliance. Also, an application in the integration server subscribes to topics on WebSphere MQ. This rule type uses topic subtrees because the complete topic string is not known well in advance. Therefore, all topic publications for all drivers of all assistance types on IBM MessageSight are mapped to WebSphere MQ topics.
Source	`/itso/driver/assistance/request`	The topic subtree onto which the MQTT client (Android application on hand-held) publishes.
Destination	`/WMQ/driver/assistance/request`	The topic subtree subscribed by the `JMSInput` node in the message flow.
Max messages	`5000`	This is the maximum number of messages in the buffer. It must be set to a value based on the expected volume *and* performance expectations.
Associated queue manager connection	`ITSO_IMA_CONN`	An WebSphere MQ connection name to which the IBM MessageSight appliance connects. Multiple connection names can be selected to allow for higher throughput.

For the exact mechanism to create the destination mapping rule, see 3.1, “WebSphere MQ connectivity and destination mapping” on page 40 and 3.2.1, “JMS integration with WebSphere MQ as the JMS provider” on page 47.

Figure 8-8 shows the destination mapping rule between IBM MessageSight and WebSphere MQ for this scenario. See Example 3-4 on page 50 for more information about how to create this destination mapping rule from the command-line interface (CLI).


Edit Destination Mapping Rule

Destination mapping rules define the direction in which messages are moved, and the nature of the source and target objects

* Name:	ITSO_IMA_QM_MAP_01
Rule Type:	MessageSight topic subtree to MQ topic subtree
* Source:	/itso/driver/assistance/request
* Destination:	/WMQ/driver/assistance/request
* Max Messages:	5,000
Retained Messages:	None

* Associated queue manager connections:

Associated	Name	Connection Name
☐	TestConnection	9.12.5.184(1414)
✓	ITSO_IMA_CONN	9.12.5.184(1414)

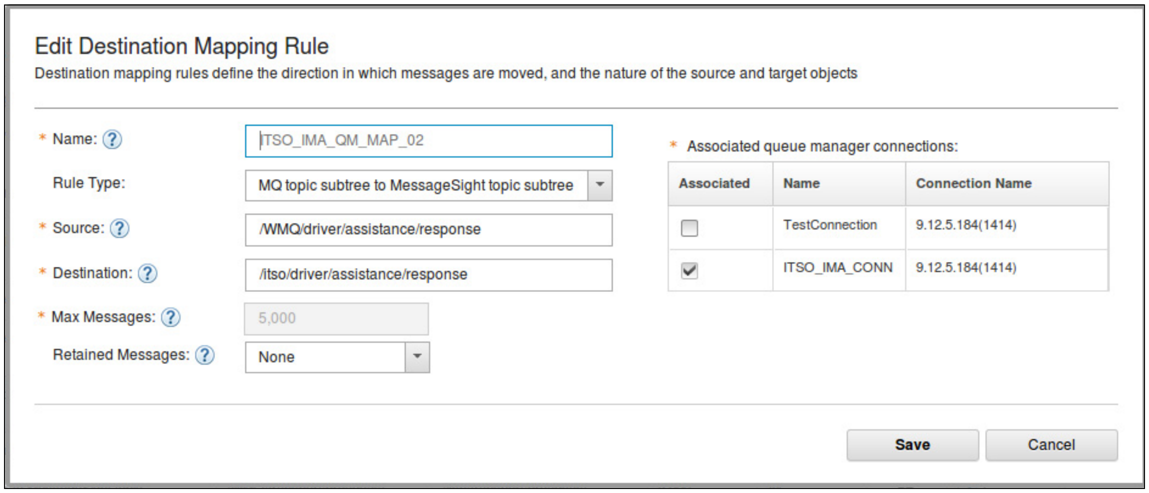
Figure 8-8 Destination mapping rule from IBM MessageSight to the WebSphere MQ topic subtrees

For this scenario, the destination mapping rule between WebSphere MQ and IBM MessageSight must be set with the following values, as shown in Table 8-2 on page 211.

Table 8-2 Values for the destination mapping rule to IBM MessageSight

Property name	Value	Remarks
Name	`ITSO_IMA_QM_MAP_02`	This is the name of the destination mapping rule.
Rule type	**WebSphere MQ topic subtree to IBM MessageSight topic subtree**	The source and destination are both topics because the application in the integration server publishes to topics on WebSphere MQ and MQTT client subscribes to topics on IBM MessageSight appliance. This rule type uses topic subtrees, because the complete topic string is not known well in advance. Therefore, all topic publications for responses to all drivers on WebSphere MQ topics will be mapped to IBM MessageSight topics.
Source	`/WMQ/driver/assistance/response`	The topic subtree onto which the application in the integration server publishes (through the `JMSOutput` node).
Destination	`/itso/driver/assistance/response`	The topic subtree subscribed by the MQTT client (the Android application on the hand-held device).
Max messages	`5000`	This is the maximum number of messages in the buffer. It must be set to a value based on the expected volume, and performance expectations. This value has no effect for a destination mapping rule from WebSphere MQ to MessageSight, only for a rule from MessageSight to WebSphere MQ. That is why it is disabled in Figure 8-9 on page 212.
Associated queue manager connection	`ITSO_IMA_CONN`	An WebSphere MQ connection name to which the IBM MessageSight appliance connects. Multiple connection names can be selected to allow for higher throughput.

For the exact mechanism to create the destination mapping rule, see 3.1, “WebSphere MQ connectivity and destination mapping” on page 40 and 3.2.1, “JMS integration with WebSphere MQ as the JMS provider” on page 47.
Figure 8-9 shows the destination mapping rule between WebSphere MQ and IBM MessageSight for this scenario. See Example 3-5 on page 52 for information about how to create this destination mapping rule from the CLI.


Edit Destination Mapping Rule

Destination mapping rules define the direction in which messages are moved, and the nature of the source and target objects

* Name:	ITSO_IMA_QM_MAP_02
Rule Type:	MQ topic subtree to MessageSight topic subtree
* Source:	/WMQ/driver/assistance/response
* Destination:	/itso/driver/assistance/response
* Max Messages:	5,000
Retained Messages:	None

* Associated queue manager connections:

Associated	Name	Connection Name
☐	TestConnection	9.12.5.184(1414)
✓	ITSO_IMA_CONN	9.12.5.184(1414)

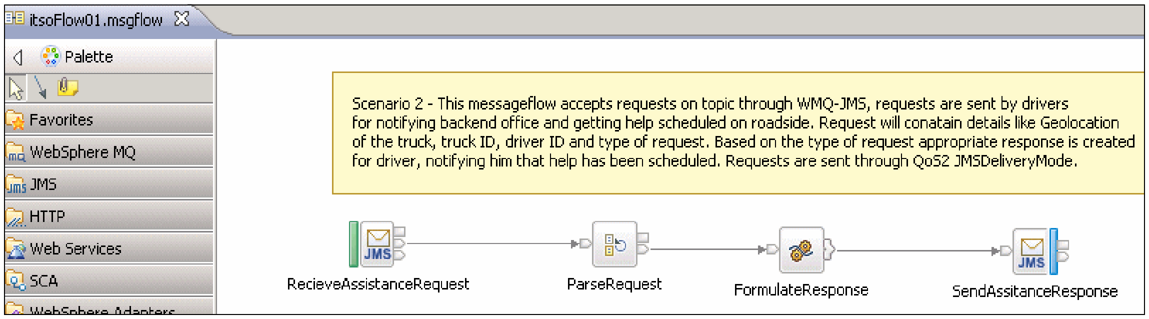
Figure 8-9 Destination mapping rule from WebSphere MQ to IBM MessageSight topic subtrees

8.2.3 Creating and deploying an IBM Integration Bus application

Deploy the IBM Integration Bus application to an integration server on the integration node instance. You can download this application from the IBM Redbooks website, as described in Appendix D, “Additional material” on page 341.

Creating an IBM Integration Bus application

This section describes the ITSOAssistanceRequestResponse application to be deployed in IBM Integration Bus for this scenario. This application has a message flow that consists of four nodes, as shown in Figure 8-10 on page 213.



Scenario 2 - This messageflow accepts requests on topic through WMQ-JMS, requests are sent by drivers for notifying backend office and getting help scheduled on roadside. Request will conatain details like Geolocation of the truck, truck ID, driver ID and type of request. Based on the type of request appropriate response is created for driver, notifying him that help has been scheduled. Requests are sent through QoS2 JMSDeliveryMode.

Figure 8-10 The ITSOAssistanceRequestResponse IBM Integration Bus application

Table 8-3 shows the nodes in the message flow of the ITSO Assistance Request Response application.

Table 8-3 Nodes in message flow of the ITSOAssistanceRequestResponse application

Node name	Node type	Description
ReceiveAssistanceRequest	`JMSInput` node	This node subscribes to a topic on WebSphere MQ, and receives roadside assistance request.
ParseRequest	`Reset Content Descriptor` node	This node converts the BLOB message into DFDL
FormulateResponse	`Compute` node	A response message is created for the roadside request received. Topic name for response is set.
SendAssistanceResponse	`JMSOutput` node	This node publishes the roadside assistance response on a topic on WebSphere MQ.

Table 8-4 shows the terminals used to connect the nodes listed in Table 8-3.

Table 8-4 Connection between nodes

Source node	Output terminal	Destination node	Input terminal
`ReceiveAssistance Request`	Out	`ParseRequest`	In
`ParseRequest`	Out	`FormulateResponse`	In
`FormulateResponse`	Out	`SendAssistance Response`	In

Table 8-5 shows the properties configured for the nodes in this message flow.

Table 8-5 Node properties

Node name	Property tab	Property name	Property value
ReceiveAssistance Request	Basic	Subscription Topic	RequestTopic
ReceiveAssistance Request	JMS Connection	Location JNDI bindings	file:///home/ wmbadmin/ itsoappjms/
ReceiveAssistance Request	JMS Connection	Connection factory name	ItsoConnFactory01
ParseRequest	Basic	Message Domain	DFDL (for binary or text messages with a Data Format Description Language (DFDL) schema model)
ParseRequest	Basic	Reset Message Domain	True
ParseRequest	Basic	Reset Message model	True
ParseRequest	Basic	Message	{urn:com.ibm.itso .samples}:itsoGeo LocationRequest
ParseRequest	Basic	Reset Message	True
FormulateResponse	Basic	Compute mode	LocalEnvironment and Message
SendAssistanceResp onse	Basic	Send to destination list in local environment	True
SendAssistanceResp onse	JMS Connection	Location Java Naming and Directory Interface (JNDI) bindings	file:///home/ wmbadmin/ itsoappjms/
SendAssistanceResp onse	JMS Connection	Connection factory name	ItsoConnFactory01

Example 8-1 shows the extended SQL (ESQL) code in the `FormulateResponse` compute node.

Example 8-1 ESQL in FormulateResponse compute node

```
BROKER SCHEMA com.ibm.itso.samples
DECLARE ns NAMESPACE 'urn:com.ibm.itso.samples';

CREATE COMPUTE MODULE itsoFlow01_CMP01
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
Declare requestTopicString Character ;
Declare responseTopicString Character ;
Declare assistanceText Character ;

-- Set topic string for response
Set requestTopicString =
InputRoot.JMSTransport.Transport_Folders.Header_Values.JMSDestination;
Set responseTopicString = getResponseTopicString(requestTopicString);
Set
OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData.Destinati
onName
= responseTopicString ;
Set
OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData.Destinati
onType
= 'Topic' ;

-- Set delivery mode appropriate for QoS=2 MQTT message from MessageSignt to
subscriber
Set
OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSDeliveryMode.(XML.At
tribute)dt = 'i4';
Set OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSDeliveryMode =
2;

-- Set body of response
Set OutputRoot.DFDL.ns:itsoGeoLocationResponse.responseMessage.driverId =
InputRoot.DFDL.ns:itsoGeoLocationRequest.requestMessage.driverId;
Set assistanceText = getAssistanceText(requestTopicString);
Set OutputRoot.DFDL.ns:itsoGeoLocationResponse.responseMessage.assistanceText =
assistanceText;

RETURN TRUE;
END;

Create Function getResponseTopicString(In requestTopicString Char) Returns Char
Begin
Declare responseTopicString Character;
```

```
-- Subject of assistance and Truck ID e.g. ACMalfunction/Truck1
Set responseTopicString = Substring(requestTopicString after
'topic:///WMQ/driver/assistance/request/');
-- Truck ID for final response string.
Set responseTopicString = Substring(responseTopicString after '/');
Set responseTopicString = 'topic:///WMQ/driver/assistance/response/' ||
responseTopicString ;

Return responseTopicString;
End ;

Create Function getAssistanceText(In requestTopicString Char) Returns Char
Begin
Declare assistanceType Character ;
Declare assistanceText Character ;

-- Subject of assistance and Truck ID e.g. ACMalfunction/Truck1
Set assistanceType = Substring(requestTopicString after
'topic:///WMQ/driver/assistance/request/');
-- Subject of assistance
Set assistanceType = Substring(assistanceType before '/');

Case

When assistanceType = 'ACMalfunction' Then
Set assistanceText = ' AC Engineer dispatched to your nearest location.';

When assistanceType = 'FlatTire' Then
Set assistanceText = ' Vehicle mechanic dispatched to your nearest location.';

When assistanceType = 'Accident' Then
Set assistanceText = ' Emergency staff notified and dispatched to your nearest
location.';

Else
Set assistanceText = 'Message not understood.';

End Case;

Return assistanceText;
End ;

END MODULE;
```

After the application is created, it is compiled and deployed on the integration server.

8.3 Scenario execution

We begin by introducing the components of the scenario. Then, we describe the connectivity between these components, both the flow of the messages and the protocols involved. Finally, we also describe the sample messages that are exchanged between the various components.

8.3.1 Components of the scenario

Figure 8-11 shows the components of a request/response scenario. The components include the Android application on the hand-held device that behaves as the MQTT client. The IBM MessageSight appliance has an endpoint that caters to the MQTT publications and subscriptions.

Through WebSphere MQ connection and destination mapping rules, the IBM MessageSight appliance communicates with an WebSphere MQ queue manager. Finally, the IBM Integration Bus communicates with the queue manager to process the request and deliver the response.

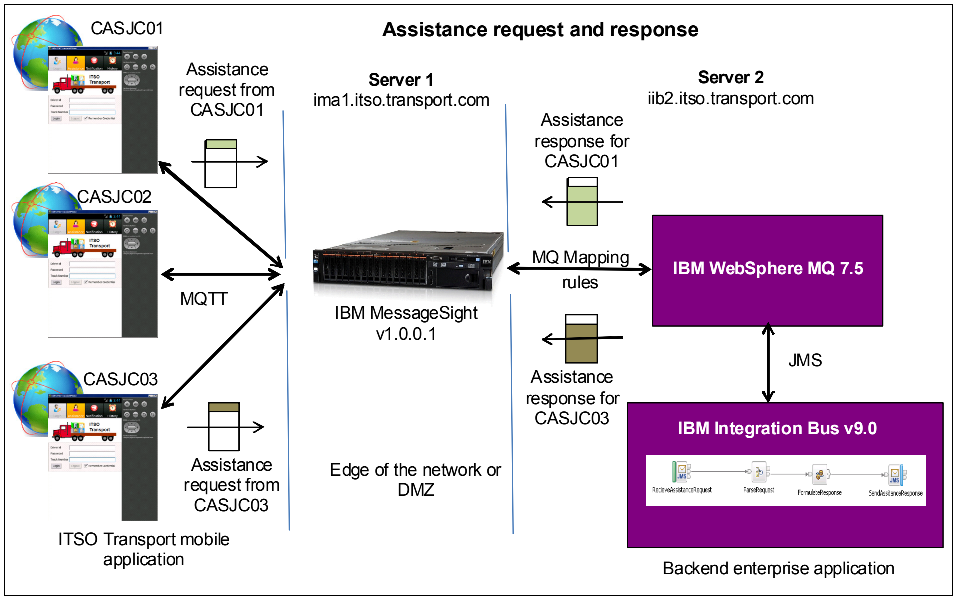

Assistance request and response
CASJC01
CASJC02
CASJC03
Assistance request from CASJC01
MQTT
Assistance request from CASJC03
ITSO Transport mobile application
Server 1
ima1.itso.transport.com
IBM MessageSight v1.0.0.1
Edge of the network or DMZ
Assistance response for CASJC01
MQ Mapping rules
Assistance response for CASJC03
Server 2
iib2.itso.transport.com
IBM WebSphere MQ 7.5
JMS
IBM Integration Bus v9.0
RecieveAssistanceRequest
ParseRequest
FormulateResponse
SendAssistanceResponse
Backend enterprise application

Figure 8-11 Components of the request/response scenario

The following steps describe the overall solution flow:

1. The Android application on the hand-held device sends out a request. This request is seeking assistance, such as for an air-conditioner malfunction, a flat tire, or an accident.
2. The request is published as an MQTT publication on a topic string on the IBM MessageSight appliance where one of the endpoints is configured to accept the MQTT publication. The IP address and the port number of this endpoint is configured in the application beforehand.
3. The IBM MessageSight appliance is configured with a WebSphere MQ connection so that IBM MessageSight can communicate with a WebSphere MQ queue manager instance. The appliance is also configured with a destination mapping rule so that the MQTT topic strings are mapped to JMS topic strings on WebSphere MQ.
4. The IBM Integration Bus has a message flow, with `JMSInput` and `JMSOutput` nodes, running. The `JMSInput` node subscribes to the JMS topic strings to which the destination mapping rule on the IBM MessageSight appliance mapped the MQTT publication.
5. This message flow parses the topic string and determines the truck number and the request type. With this parsed information, an appropriate business rule is started. In this scenario, the business rule is no more than returning with a favorable response corresponding to the request.
6. The message flow publishes the response through a `JMSOutput` node onto a topic string on the IBM MessageSight appliance.
7. The IBM MessageSight appliance maps this WebSphere MQ topic string to the IBM MessageSight topic string as defined in the destination mapping rule.
8. The Android application on the hand-held device subscribes to this topic string and receives the response.

8.3.2 Topic strings and protocols

Table 8-6 summarizes the applications, their actions, their topic strings, and the protocol involved. The Action column describes the action on the IBM MessageSight appliance.

Table 8-6 Topic strings and protocols on the IBM MessageSight appliance

From	Action	Topic string	Protocol
Android application	Publish	`/itso/driver/assistance/request`	MQTT
Android application	Subscribe	`/itso/driver/assistance/response`	MQTT
IBM Integration Bus message flow	Publish	`/WMQ/driver/assistance/response`	JMS
IBM Integration Bus message flow	Subscribe	`/WMQ/driver/assistance/request`	JMS

8.3.3 Sample messages

The messages, published as requests for assistance, conform to the following pattern:

```
yyyy-mm-dd hh:MM:ss,driverID,latitude,longitude
```

These request messages are published onto a topic string in the following pattern:

`/itso/driver/assistance/request/`*`requestType`*`/TruckID`

The *`requestType`* is one of `ACMalfunction`, `FlatTire`, or `Accident`, and the `TruckID` is the ID of the truck that also serves as the client ID of the MQTT client.

The response message received follows a pattern:

```
driverID,assistance message
```

This response message is published onto a topic string that follows a pattern:

```
/itso/driver/assistance/response/TruckID
```

For example, a sample request message (`2013-08-09 15-10-29,driver01,-142.08409333333,23.422005`) is published onto a topic string `/itso/driver/assistance/request/ACMalfunction/CASJC01`.

The corresponding response is received as (`driver01, AC Engineer despatched to your nearest location`) on the topic string `/itso/driver/assistance/response/CASJC01`.

Using this example, Table 8-6 on page 219 can be rewritten with complete topic strings, as shown in Table 8-7.

Table 8-7 Topic strings on the IBM MessageSight appliance

From	Action	Topic string
Android application	Publish	`/itso/driver/assistance/request/ACMalfunction/CASJC01`
Android application	Subscribe	`/itso/driver/assistance/response/CASJC01`
IBM Integration Bus message flow	Publish	`/WMQ/driver/assistance/response/CASJC01`
IBM Integration Bus message flow	Subscribe	`/WMQ/driver/assistance/request/ACMalfunction/CASJC01`

Note: The message flow in IBM Integration Bus parses the publication topic string to determine the nature of the request and the truck identifier. This enables the flow to formulate an appropriate response, and also the topic string to which the response must be delivered.

8.4 Scenario testing

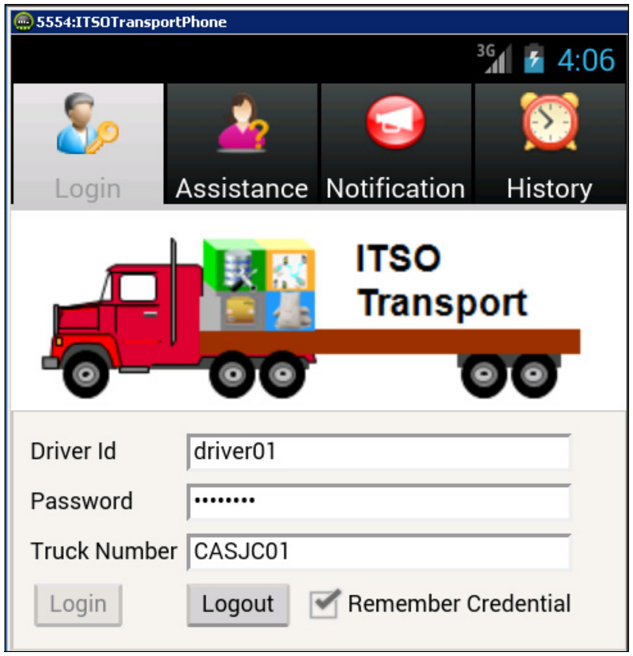
Test this scenario:

1. Start the Android application.
2. Connect to the IBM MessageSight appliance.
3. Publish a request message.

   The response is received on the Android application.

Authenticate with IBM MessageSight:

1. Enter the driver ID, password, and the Truck ID.
2. Click Login, as shown in Figure 8-12.



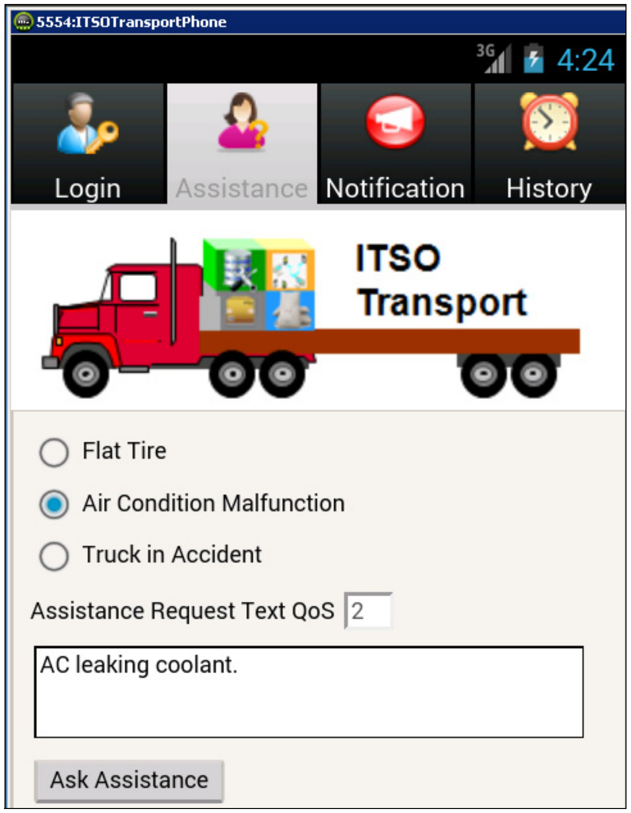
Driver Id: driver01
Password: ••••••••
Truck Number: CASJC01

Figure 8-12 Log in to IBM MessageSight appliance

To send an assistance request, follow these steps:

1. Click the **Assistance** tab.
2. Click **Air Condition Malfunction**.
3. Enter the QoS value as `2`.
4. Enter `AC leaking coolant` in the **Assistance Request Text** box.
5. Click **Ask Assistance**.

Figure 8-13 shows the Android application requesting assistance.


5554:ITSOTransportPhone

- ( ) Flat Tire
- [✓ Air Condition Malfunction]
- ( ) Truck in Accident

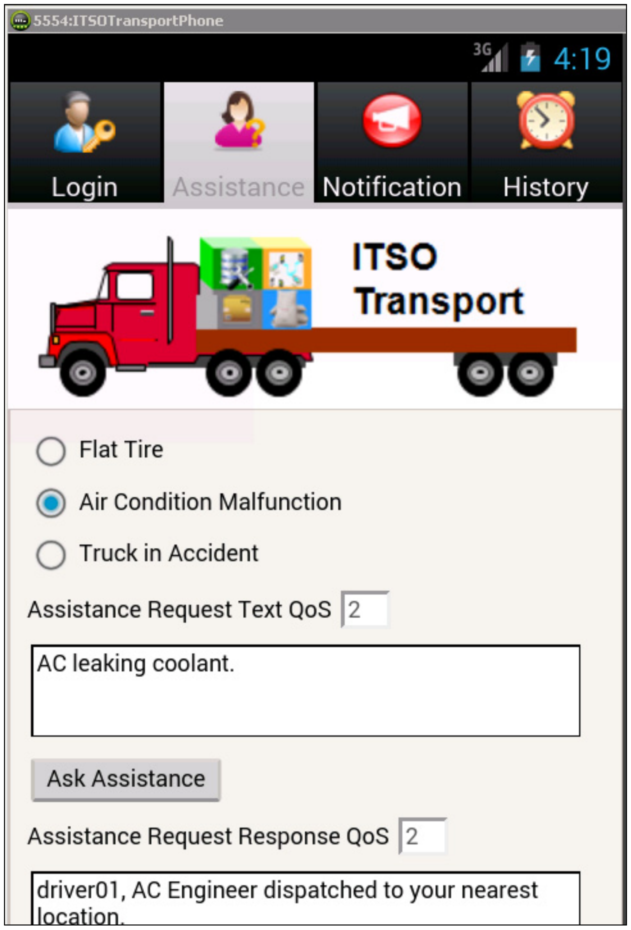
Assistance Request Text QoS: 2

AC leaking coolant.

Figure 8-13 Requesting assistance (publish request message)

6. As a result of step 5, the Android application publishes the request message to a topic string determined by the truck identifier and the request type, as described in Table 8-7 on page 220.

7. The back-end system responds back with the message (its QoS is also set to 2). Figure 8-14 shows the response as received on the Android application.



Flat Tire
[✓ Air Condition Malfunction]
Truck in Accident

Assistance Request Text QoS 2

AC leaking coolant.

Assistance Request Response QoS 2

driver01, AC Engineer dispatched to your nearest location.

Figure 8-14 Response received on Android application (subscription of response message)

Tip: This setup can be used with WebSphere Message Broker v8.0 to integrate with IBM MessageSight through WebSphere MQ as the JMS Service provider.

Scenario 3: Push notifications with quality of service

This chapter provides an example scenario that uses several different systems to send push notifications with different qualities of service (QoS). This scenario demonstrates how a back-end enterprise application can send push notifications to selected mobile applications, and can also send broadcast messages to all mobile applications connected to IBM MessageSight server.

Beyond the delivery of notification messages to Android applications, this scenario does not present the business decision-making logic for sending these notifications.

This chapter covers the following topics:

- Business value
- Prerequisites: Technical and infrastructure
- Scenario outline
- Scenario configuration and implementation
- Testing the scenarios

Tip: If you want to implement the scenario presented in this chapter in your own environment, you can download the Project Interchange file for the applications used in this scenario from the IBM Redbooks website. For download instructions, see Appendix D, “Additional material” on page 341.

9.1 Business value

Short Messaging Service (SMS) and Multimedia Messaging Service (MMS) messages are delivered by wireless carriers to specific devices using the phone numbers. Developers of server-side applications that implement SMS and MMS messaging must interact extensively with the existing, closed telecommunications infrastructure, including obtaining phone numbers.

The popularity of smartphones, such as iPhone and Android devices, has created a need for a modern mobile messaging infrastructure:

- Is more open and less expensive than SMS or MMS
- Can deliver rich media information directly to the installed application, rather than to phone numbers

A *push notification* is the ability of mobile devices to receive messages that are pushed from the back-end application and servers. Notifications are received regardless of whether the application is running, and take these forms:

- Alerts (text messages)
- Badges (marks on the application icon)
- Sounds (audio alerts)
- Notification center (iOS) and Notification bar (Android)

The following list includes a few of the benefits of using push notifications:

- Control. Users can choose whether to subscribe to the push service.
- Availability. Push notifications arrive at the user’s device regardless of whether the application is running.
- Efficiency. There is no need to issue constant queries from the applications, which translates to reduced development effort, better application performance, and savings on battery and communication fees.

Common push notification architecture supports a variety of scenarios:

- A single application notifying multiple devices of the same user
- Multiple event sources used by the same application
- Multiple applications using the same event source
- Multiple users logging in to the same application

9.1.1 Using push notifications with MQTT

In many situations, for mobile applications that must react to some data from a remote server, it is more efficient for changes to the data to be pushed to the mobile application, rather than for the mobile to repeatedly poll the server. Polling is generally an unsatisfactory compromise.

Polling infrequently results in a mobile application that can appear slow to respond to changes in the data. For example, polling every 10 minutes can potentially result in an application that takes 9 minutes before it notices the queried data has changed.

Frequently polling can mitigate long waits, but at the cost of increasing the consumption of battery and network resources in the device. It is better for the mobile application to do neither, and simply wait for the server to tell it when the data changes.

In MQTT, after the connection is established, events can be transmitted in both directions, from the client to the server or from the server to the client. Therefore, if the server has data available for the client, the server can push this data to the client rather than the client continuously polling for data on the server until the message is available.

Reliable delivery of messages across fragile networks with different qualities of service, even when the connection breaks, is one of the important features of the MQTT specification. MQTT provides three qualities of service for message delivery:

Qos0	*At most once*, where messages are delivered according to the best efforts of the operating environment. Message loss or duplication can occur.
QoS1	*At least once*, where messages are assured to arrive but duplicates can occur.
QoS2	*Exactly once*, where messages are assured to arrive exactly once.

Using pushing notifications with MQTT provides all of the advantages of MQTT, along with the ability to deliver notifications with various QoS.

9.2 Prerequisites: Technical and infrastructure

To fully understand the scenario, and to successfully implement it in your own infrastructure, both technical and infrastructure prerequisites are required.

9.2.1 Software prerequisites

To run this scenario, you must have the following components installed:

- IBM MessageSight V1.0.0.1 (Appliance)
- WebSphere MQ V7.5.0.2
- IBM Integration Bus V9.0.0.0
- IBM DB2 V10.1.0.2
- Android software development kit (SDK) emulator

Figure 9-1 illustrates the configuration of the environment used to implement this scenario, and represents the systems of the fictitious company, ITSO Transport Corporation.

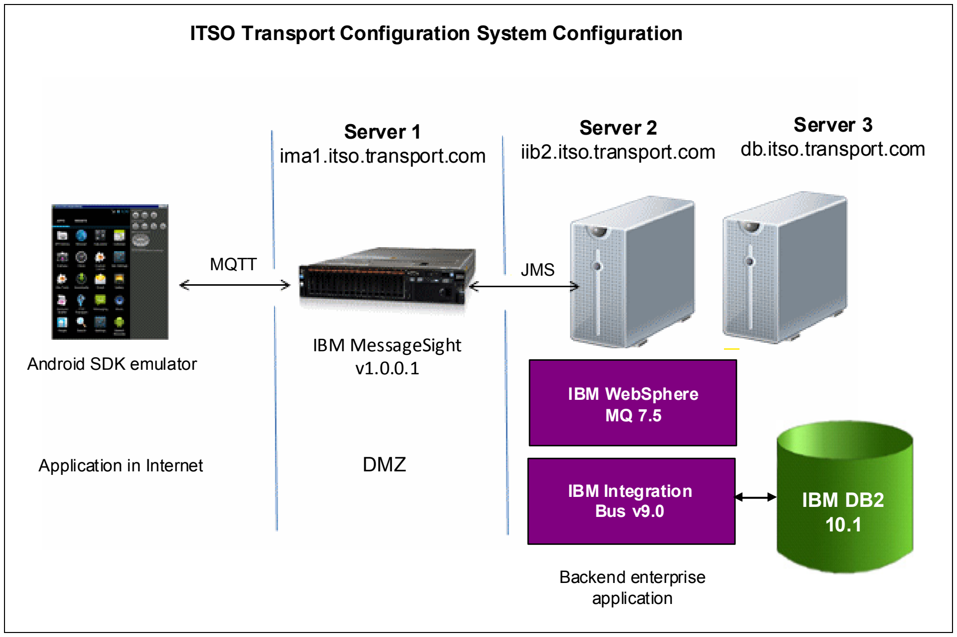


Figure 9-1 System configuration for ITSO Transport Corporation

For the purpose of this scenario, there are three servers:

- The first server is the IBM MessageSight appliance placed in the DMZ (layers of protection between the server and the device).
- The second server hosts the back-end enterprise service bus (ESB) integration server, and has IBM Integration Bus V9.0 and WebSphere MQ V7.5 installed on it.
- The third server hosts the back-end database, and has IBM DB2 V10.1 installed on it.

9.2.2 Skills prerequisites

To fully implement and understand this scenario, you must be familiar with these tasks:

- Configuring IBM MessageSight infrastructure
- Configuring WebSphere MQ infrastructure
- Configuring IBM Integration Bus infrastructure and developing message flows
- Configuring IBM DB2 infrastructure

9.3 Scenario outline

In this section, you look at an overview of how to configure this scenario. In later sections we go through a detailed description of this configuration.

The fictitious company ITSO Transport Corporation provides transport services to grocery retailers around the country. Trucks from this company are scheduled to pick up groceries from centralized cold storage and deliver groceries to the retail stores using trucks enabled with cold storage containers.

Timely delivery of groceries is a key success attribute for their business, and it helps build customer satisfaction. Every truck scheduled for delivery has two crew members, a truck driver and an assistant.

Timely delivery of groceries is critical to this business, so that fresh produce can be sold on the counters. Any problems during the delivery can cause delivery delays and groceries to get stale. It has been observed that deliveries are obstructed by a variety of problems on the highways. Truck drivers who are not aware of traffic congestion and weather problems on their delivery route might run into unexpected problems and cause delivery delays.

9.3.1 Push notification to selected mobile applications

To help drivers save time and be safe on the highways, ITSO Transport company wants to develop a push notification system that analyzes drivers' current locations and notifies them with any problems predicted on their delivery route.

Any changes to the geographic location of trucks are published from the mobile applications through MQTT using the MQTT topic string illustrated in Example 9-1. <*Truck Number*> in the topic string is also the client ID, and is replaced with the actual truck number.

Example 9-1 Geographic location change, publish topic string

```
/itso/driver/geolocation/<Truck Number>
```

Example 9-2 illustrates the message payload format for the geographic location changes.

Example 9-2 Geographic location change, message format

```
TimeStamp,driver username,longitude,latitude
```

Table 9-1 summarizes the publisher and subscriber for the geographic location topic.

Table 9-1 Publisher and subscriber for the geographic location topic

Application	Action	Protocol	QoS	Retained
ITSO mobile application	PUBLISH	MQTT	0	N/A
ITSO back-end enterprise application	SUBSCRIBE	Java Message Service (JMS)	N/A	N/A

Geographic location changes are published from the ITSO Transport mobile application through MQTT to the IBM MessageSight server. In the background, there is an enterprise application that subscribes to the same topic, as shown in Example 9-1, and receives truck location changes through JMS connection.

This back-end enterprise application is implemented through an IBM Integration Bus message flow that has a `JMSInput` node. This message flow captures the location changes of trucks and inserts that data into the database.

There is another message flow on IBM Integration Bus:

1. Runs at a scheduled time
2. Checks for all trucks whose geographic location is available in the database
3. Sends them a message as a push notification that sends the drivers critical alert messages

Note: The algorithm to select trucks that is provided in this scenario is basic. Better implementations can be performed through integration with rule management systems, such as IBM WebSphere Operational Decision Management. However, to keep it simple for the purposes of this example, push notification is sent to unique trucks whose geographic location is available in the database.

The second message flow is based on timer nodes and is run at specific intervals of time. It identifies unique truck numbers that updated their geographic location in the ITSO database. The message flow selects appropriate notification messages for the identified trucks.

This message flow uses a JMS topic, as shown in Example 9-3, to send the notification alerts. <*Truck Number*> in this topic string is replaced with the actual truck number for which the notification is intended.

Example 9-3 Push notification topic for selected trucks

```
/itso/driver/notification/<TruckNumber>
```

The MQTT mobile application subscribes on the same topic in Example 9-3, but the <*Truck Number*> is replaced with the actual truck number to receive notifications from the back-end system. An alert message sent as a push notification can be in ASCII string format.

Table 9-2 summarizes the publisher and subscriber for push notification topics for selected trucks.

Table 9-2 Publisher and subscriber for push notification topics for selected trucks

Application	Action	Protocol	QoS	Retained
ITSO mobile application	SUBSCRIBE	MQTT	2	N/A
ITSO back-end enterprise application	PUBLISH	JMS	N/A	N/A

Figure 9-2 illustrates the scenario:

1. Geographic location changes are captured from mobile applications in trucks.
2. The back-end system uses that data to send critical alert notifications to selected trucks.

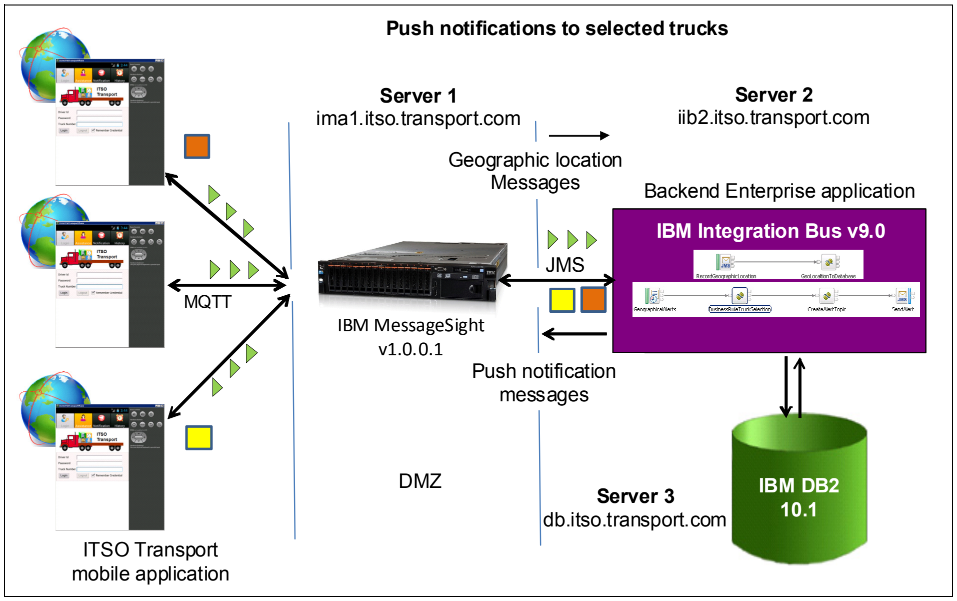

Push notifications to selected trucks
Server 1
ima1.itso.transport.com
Server 2
iib2.itso.transport.com
Geographic location Messages
Backend Enterprise application
IBM Integration Bus v9.0
JMS
MQTT
IBM MessageSight v1.0.0.1
Push notification messages
DMZ
Server 3
db.itso.transport.com
IBM DB2 10.1
ITSO Transport mobile application

Figure 9-2 Push notification to selected mobile application

9.3.2 Push notification broadcast to all mobile applications

The ITSO Transport Corporation back-end office team might want to notify all of the trucks currently delivering groceries with a few important alerts. Messages to be broadcast are accepted through a WebSphere MQ queue, and later the IBM Integration Bus V9 message flow gets those messages and transforms them in JMS format. The message is marked as retained, and published on a JMS topic, as shown in Example 9-4.

Example 9-4 Push notification topic for all trucks

```
/itso/driver/notification/AllTrucks
```

The same topic string is used by the ITSO Transport mobile application to subscribe for broadcast notification messages through MQTT. Alert messages sent as push notifications can be in ASCII string format.

Table 9-3 summarizes the publisher and subscriber for push notification topics for all trucks currently delivering groceries.

Table 9-3 Publisher and subscriber for push notification topics for all trucks

Application	Action	Protocol	QoS	Retained
ITSO mobile application	SUBSCRIBE	MQTT	1	N/A
ITSO back-end enterprise application	PUBLISH	JMS	N/A	True

Figure 9-3 illustrates the scenario described in this section:

1. The back-end office staff can send important notifications for trucks through WebSphere MQ.
2. Later, the message is published to all trucks.

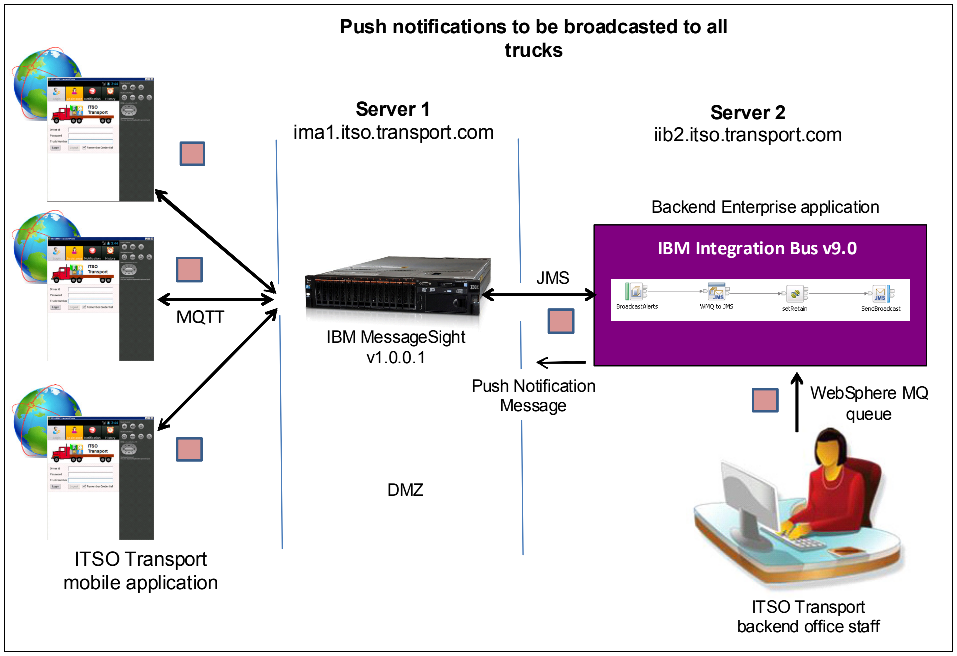

Push notifications to be broadcasted to all trucks
Server 1
ima1.itso.transport.com
Server 2
iib2.itso.transport.com
Backend Enterprise application
IBM Integration Bus v9.0
BroadcastAlerts
WMQ to JMS
setRetain
SendBroadcast
JMS
MQTT
IBM MessageSight v1.0.0.1
Push Notification Message
WebSphere MQ queue
DMZ
ITSO Transport mobile application
ITSO Transport backend office staff

Figure 9-3 Push notification to all mobile applications

Note: In the broadcast scenario, the JMS publisher sends retained publications to the MQTT subscribers.

9.4 Scenario configuration and implementation

In all of the following steps, we assume that the IBM MessageSight, IBM Worklight, and IBM Integration Bus components are installed, configured, and tested to ensure that they function properly, and that a nominal performance configuration is put in place for each component of the integration to accommodate the anticipated processing load.

For detailed information about installing the applications, see the installation guide and the performance tuning guide (if applicable) for each of these products.

9.4.1 Configuring IBM MessageSight

First, log in to the appliance using the console.

Creating a group for ITSO Transport truck drivers

This is a functional group for truck driver user names. The users of this group are allowed through the connection and messaging policy in a message hub.

You can create the IBM MessageSight groups using the **`imaserver group`** command. Example 9-5 shows the syntax of this command.

Example 9-5 The imaserver create group command

```
imaserver group add "GroupID=GroupID" "GroupMembership=group"
"Description=description"
```

For more information about the parameters for the **`imaserver group`** command, see the following website:

http://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00317_.html

The command in Example 9-6 creates a group called `drivers` on IBM MessageSight `ima1.itso.transport.com`.

Example 9-6 Creating group drivers on IBM MessageSight

```
Console> imaserver group add "GroupID=drivers" "Description=ITSO Truck driver
group"
The requested configuration change has completed successfully.
```

Creating users for ITSO Transport

Create four messaging users as example truck drivers in ITSO Transport Corporation. These users are members of the `drivers` group created previously. Table 9-4 illustrates the messaging users for the ITSO Transport mobile application.

Table 9-4 Messaging users for ITSO Transport mobile application

Driver user name	Driver password	Group membership
driver01	dr!ver@1	drivers
driver02	dr!ver@2	drivers
driver03	dr!ver@3	drivers
driver04	dr!ver@4	drivers

Important: The password mentioned here is for illustration purposes only. You ought to follow the password standards as applicable in your organization.

You can create the IBM MessageSight groups by using the **`imaserver user`** command. Example 9-7 shows the syntax of this command.

Example 9-7 The imaserver create user command

```
imaserver user add "UserID=UserID" "Type=Messaging" "Password=password"
"GroupMembership=GroupID" "Description=description"
```

For more information about the parameters for the **`imaserver user`** command, see the following website:

http://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00313_.html

The **`imaserver user add`** commands in Example 9-8 create messaging users `driver01`, `driver02`, `driver03`, and `driver04` on IBM MessageSight `ima1.itso.transport.com`.

Example 9-8 Creating messaging user drivers on IBM MessageSight

```
Console> imaserver user add "UserID=driver01" "Type=Messaging"
"Password=dr!ver@1" "GroupMembership=drivers" "Description=ITSO Transport
Driver 01"
The requested configuration change has completed successfully.
```

```
Console> imaserver user add "UserID=driver02" "Type=Messaging"
"Password=dr!ver@2" "GroupMembership=drivers" "Description=ITSO Transport
Driver 02"
The requested configuration change has completed successfully.

Console> imaserver user add "UserID=driver03" "Type=Messaging"
"Password=dr!ver@3" "GroupMembership=drivers" "Description=ITSO Transport
Driver 03"
The requested configuration change has completed successfully.
Console> imaserver user add "UserID=driver04" "Type=Messaging"
"Password=dr!ver@4" "GroupMembership=drivers" "Description=ITSO Transport
Driver 04"
The requested configuration change has completed successfully.
```

The **`imaserver user list`** command in Example 9-9 lists the messaging users created on IBM MessageSight.

Example 9-9 List of messaging users

```
Console> imaserver user list "Type=Messaging"

UserID: driver01
Description: ITSO Transport Driver 01
Group Membership: drivers

UserID: driver02
Description: ITSO Transport Driver 02
Group Membership: drivers

UserID: driver03
Description: ITSO Transport Driver 03
Group Membership: drivers

UserID: driver04
Description: ITSO Transport Driver 04
Group Membership: drivers
```

Creating a message hub

Create a message hub for ITSO Transport Corporation with the name ITSO_HUB. Table 9-5 lists the message hub to be created.

Table 9-5 ITSO message hub in MessageSight

Message hub	Description
ITSO_HUB	Message hub for ITSO Transport application

You can create the MessageSight message hubs by using the **`imaserver create MessageHub`** command. Example 9-10 shows the syntax of this command.

Example 9-10 The imaserver create hub command

```
imaserver create MessageHub "Name=hubName" "Description=description"
```

For more information about all of the parameters for the **`imaserver create MessageHub`** command, see the following website:

`http://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00323_.html`

The command in Example 9-11 creates a message hub called `ITSO_HUB` on IBM MessageSight ima1.itso.transport.com.

Example 9-11 Creating a message hub on IBM MessageSight

```
Console> imaserver create MessageHub "Name=ITSO_HUB" "Description=Message Hub
for ITSO Transport Application"
The requested configuration change has completed successfully.
```

Creating connection policies

Two connection policies are created for the ITSO_HUB message hub. Table 9-6 summarizes these connection policies.

Table 9-6 Summary of connection policies

Connection policy name	Protocol	Description
ITSO_Connections_Driver	MQTT	Connection policy for the ITSO Transport mobile application on trucks
ITSO_Connections_ESB	JMS	Connection policy for the ITSO Transport back-end enterprise application using the JMS protocol

You can create the MessageSight connection policies by using the **`imaserver create ConnectionPolicy`** command. Example 9-12 shows the syntax for this command.

Example 9-12 The imaserver create connection policy command

```
imaserver create ConnectionPolicy "Name=connPolicyName"
"Description=description" "ClientID=ClientID" "ClientAddress=IP_Address"
"UserID=UserID" "GroupID=groupName" "CommonNames=certificateCommonNames"
"Protocol=protocols"
```

For more information about all of the parameters for the **imaserver create ConnectionPolicy** command, see the following website:

http://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00327_.html

The command in Example 9-13 creates the ITSO_Connections_Drivers connection policy for mobile applications to connect to IBM MessageSight ima1.itso.transport.com.

Example 9-13 Creating the ITSO_Connections_Drivers connection policy

```
Console> imaserver create ConnectionPolicy "Name=ITSO_Connections_Drivers"
"Description=Connection policy for ITSO Transport mobile application on trucks"
"ClientID=CASJC*" "GroupID=drivers" "Protocol=MQTT"
The requested configuration change has completed successfully.
```

The command in Example 9-14 creates the ITSO_Connections_ESB connection policy for back-end connections established from IBM Integration Bus message flows using the JMS protocol. The ClientAddress referred to in the command is the IP address for the IBM Integration Bus server iib1.itso.transport.com.

Example 9-14 Creating the ITSO_Connections_ESB connection policy

```
Console> imaserver create ConnectionPolicy "Name=ITSO_Connections_ESB"
"Description=Connection policy for ITSO Transport back-end enterprise
application through JMS protocol" "ClientAddress=9.12.5.184" "Protocol=JMS"
The requested configuration change has completed successfully.
```

The ClientID referred to in the connection policy for the mobile application in Example 9-13 is the truck number driven by an ITSO Transport Corporation driver. Table 9-7 illustrates the trucks in the ITSO Transport Corporation fleet.

Table 9-7 ITSO Transport trucks

Truck Number	**Description**
CASJC01	Truck driven by driver01
CASJC02	Truck driven by driver02
CASJC03	Truck driven by driver03
CASJC04	Truck driven by driver04

The command in Example 9-15 shows the list of connection policies just created.

Example 9-15 List connection policies

```
Console> imaserver list ConnectionPolicy
ITSO_Connections_Drivers
ITSO_Connections_ESB
```

Creating messaging policies

Four messaging policies are created for this scenario. Table 9-8 summarizes these messaging policies.

Table 9-8 Summary of messaging policies

Messaging policy name	Protocol	Description
ITSO_Driver_Geolocation_ Publish	MQTT	Messaging policy for publishing the truck longitudes and latitudes
ITSO_ESB_Geolocation_ Subscribe	JMS	Messaging policy for back-end ESB to subscribe for geographic location changes
ITSO_Driver_Notification_ Subscribe	MQTT	Messaging policy for trucks to receive notifications and broadcasts
ITSO_ESB_Notification_ Publish	JMS	Messaging policy for back-end policy to send notifications

You can create the MessageSight messaging policies by using the **`imaserver create MessagingPolicy`** command. Example 9-16 shows the syntax for this command.

Example 9-16 The imaserver create messaging policy command

```
imaserver create MessagingPolicy "Name=msgPolicyName" "Description=description"
"DestinationType=type" "Destination=destinationName" "MaxMessages=count"
"ActionList=actions" "ClientID=ClientID" "UserID=UserID" "GroupID=groupName"
"CommonNames=certificateCommonNames" "ClientAddress=IP_Address"
"Protocol=protocols"
```

For more information about all of the parameters for the **`imaserver create MessagingPolicy`** command, see the following website:

`http://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00368_.html`

The command in Example 9-17 creates the `ITSO_Driver_Geolocation_Publish` messaging policy. This policy is for mobile applications to connect to IBM MessageSight `ima1.itso.transport.com` through MQTT, and to enable the publish action on topics matching the pattern `/itso/driver/geolocation/*`.

Example 9-17 Creating the ITSO_Driver_Geolocation_Publish messaging policy

```
Console> imaserver create MessagingPolicy
"Name=ITSO_Driver_Geolocation_Publish" "Description=Messaging policy for
publishing the truck longitude and latitudes" "DestinationType=Topic"
"Destination=/itso/driver/geolocation/*" "MaxMessages=5000"
"ActionList=Publish" "ClientID=CASJC*" "GroupID=drivers" "Protocol=MQTT"
The requested configuration change has completed successfully.
```

The command in Example 9-18 creates the `ITSO_ESB_Geolocation_Subscribe` connection messaging policy. This policy is for back-end connections established from IBM Integration Bus message flows using the JMS protocol. This messaging policy enables message flows to subscribe on topics with pattern `*`. The `ClientAddress` referred to in the command is the IP address for the IBM Integration Bus server `iib1.itso.transport.com`.

Example 9-18 Creating the ITSO_ESB_Geolocation_Subscribe messaging policy

```
Console> imaserver create MessagingPolicy "Name=ITSO_ESB_Geolocation_Subscribe"
"Description=Messaging policy for back-end ESB to subscribe for GeoLocation"
"DestinationType=Topic" "Destination=*" "MaxMessages=5000"
"ActionList=Subscribe" "ClientAddress=9.12.5.184" "Protocol=JMS"
The requested configuration change has completed successfully.
```

The command in Example 9-19 creates the `ITSO_Driver_Notification_Subscribe` messaging policy. This policy is for mobile applications to connect to IBM MessageSight `ima1.itso.transport.com` using the MQTT protocol, and to enable the subscribe action on topics matching the pattern `*`.

Example 9-19 creating messaging policy ITSO_Driver_Notification_Subscribe

```
Console> imaserver create MessagingPolicy
"Name=ITSO_Driver_Notification_Subscribe" "Description=Messaging policy for
trucks to receive notifications and broadcast." "DestinationType=Topic"
"Destination=/itso/driver/notification/*" "MaxMessages=5000"
"ActionList=Subscribe" "ClientID=CASJC*" "GroupID=drivers" "Protocol=MQTT"
The requested configuration change has completed successfully.
```

The command in Example 9-20 creates the `ITSO_ESB_Notification_Publish`, for back-end connection connections established from IBM Integration Bus message flows through JMS protocol. This messaging policy enables message flows to publish on topics with pattern *. The clientAddress referred in the command is the IP address for IBM Integration Bus server iib1.itso.transport.com.

Example 9-20 Creating messaging policy ITSO_ESB_Notification_Publish

```
Console> imaserver create MessagingPolicy "Name=ITSO_ESB_Notification_Publish"
"Description=Messaging policy for back-end policy to send notifications"
"DestinationType=Topic" "Destination=*" "MaxMessages=5000" "ActionList=Publish"
"ClientAddress=9.12.5.184" "Protocol=JMS"
The requested configuration change has completed successfully.
```

The command in Example 9-21 on page 241 shows the list of messaging policies just created.

Example 9-21 List messaging policies

```
Console> imaserver list MessagingPolicy
ITSO_Driver_Geolocation_Publish
ITSO_ESB_Geolocation_Subscribe
ITSO_Driver_Notification_Subscribe
ITSO_ESB_Notification_Publish
```

Creating an endpoint

One endpoint is created for this scenario, as listed in Table 9-9.

Table 9-9 ITSO endpoint in IBM MessageSight

Endpoint	Description
ITSO_EndPoint	Connection endpoint for MQTT and JMS connections

You can create the IBM MessageSight endpoint by using the **`imaserver create Endpoint`** command. Example 9-22 shows the syntax of this command.

Example 9-22 The imaserver create endpoint command

```
imaserver create Endpoint "Name=endpointName" "Description=description"
"Port=portNumber" "Interface=IP_Address" "Protocol=protocols"
"ConnectionPolicies=connPolicies" "MessagingPolicies=msgPolicies"
"MaxMessageSize=size" "MessageHub=hubName" "SecurityProfile=secProfile"
"Enabled=True|False"
```

For more information about all of the parameters for the **imaserver create Endpoint** command, see the following website:

http://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Administering/ad00369_.html

The command in Example 9-23 creates an endpoint called `ITSO_Endpoint` for the ITSO_HUB messaging hub. This endpoint uses all of the connection and messaging policies that were just created.

Example 9-23 Creating endpoint ITSO_Endpoint

```
Console> imaserver create Endpoint "Name=ITSO_Endpoint" "Description=Connection
end-point for MQTT and JMS connection" "Port=16105" "Interface=All"
"Protocol=JMS,MQTT"
"ConnectionPolicies=ITSO_Connections_Drivers,ITSO_Connections_ESB"
"MessagingPolicies=ITSO_Driver_Geolocation_Publish,ITSO_ESB_Geolocation_Subscri
be,ITSO_Driver_Notification_Subscribe,ITSO_ESB_Notification_Publish"
"MaxMessageSize=1024KB" "MessageHub=ITSO_HUB" "Enabled=True"
The requested configuration change has completed successfully.
```

After the endpoint is created and enabled, it is ready to be used by JMS and MQTT applications.

9.4.2 Configuring IBM DB2

Database for capturing the geographic locations published from the mobile application in the trucks is created on the `db.itso.transport.com` server. The database is created with the name `ITSODB`. Example 9-24 shows the command to create DB2 database `ITSODB`.

Example 9-24 Creating the ITSODB database

```
$ db2 CREATE DB ITSODB
DB20000I  The CREATE DATABASE command completed successfully.
```

After the database is successfully created, issue a command to connect to the database. Example 9-25 shows the command to connect to the `ITSODB` database.

Example 9-25 Connect to the ITSODB database

```
$ db2 CONNECT TO ITSODB

   Database Connection Information

 Database server        = DB2/LINUXX8664 10.1.2
 SQL authorization ID   = DB2INST1
 Local database alias   = ITSODB
```

Open the DB2 console, and create a table called TRUCKGEOLOCATION, as shown in Example 9-26.

Example 9-26 Creating the TRUCKGEOLOCATION table

```
db2 => CREATE TABLE TRUCKGEOLOCATION (TIMSTAMP TIMESTAMP, DRIVER_NAME
VARCHAR(30), TRUCK_NUMBER VARCHAR(30), LONGITUDE VARCHAR(30), LATITUDE
VARCHAR(30))
DB20000I  The SQL command completed successfully.
```

Note: Ensure that the database is configured for a DB2 connection service port. For ITSODB, this service is configured to bind port 50001.

9.4.3 Configuring WebSphere MQ

In this scenario, use WebSphere MQ queue manager to create an IBM Integration Bus integration node. Also, a queue on this queue manager is used for receiving broadcast messages from the ITSO Transport Corporation back-end office team.

Creating the ITSOQM2 queue manager

You can create the queue managers by using the **crtmqm** command.

For more information about all of the parameters for the **crtmqm** command, see the following website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.ref.adm.doc/q083120_.htm

Example 9-27 shows the command to create a queue manager called ITSOQM2.

Example 9-27 Command to create the ITSOQM2 queue manager

```
$ crtmqm ITSOQM2
WebSphere MQ queue manager created.
Directory '/var/mqm/qmgrs/ITSOQM2' created.
The queue manager is associated with installation 'Installation1'.
Creating or replacing default objects for queue manager 'ITSOQM2'.
Default objects statistics : 74 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

Now that all of the queue managers are created, you must start them using the **strmqm** command.

For more information about all of the parameters for the **strmqm** command, see the following website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.ref.adm.doc/q083650_.htm

Example 9-28 shows the command used to start the various queue managers.

Example 9-28 Command to start the ITSOQM2 queue manager

```
strmqm ITSOQM2
```

Creating a listener object on the queue manager

In this section, you create listener objects on various queue managers by issuing commands on the WebSphere MQ runmqsc console. Also, the listeners' control is set to be the queue manager (`QMGR`), so that listeners start and stop based on the control of the queue manager.

For more information about defining listeners, see the following website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.ref.adm.doc/q085630_.htm

Example 9-29 illustrates commands to create and start a listener on the `ITSOQM2` queue manager.

Example 9-29 Creating and start listener on the ITSOQM2 queue manager

```
DEFINE LISTENER(ITSOQM2.LISTENER) TRPTYPE(TCP) CONTROL(QMGR) PORT(1414)
IPADDR('iib2.itso.transport.com') REPLACE

START LISTENER(ITSOQM2.LISTENER)
```

Creating a server connection channel

Message Queue Interface (MQI) channels are used by applications in client mode (rather than bindings mode) to connect to queue managers. MQI channels are bidirectional:

- They carry WebSphere MQ API calls, (for example, `GET a message from queue XYZ`) from the application to the queue manager.
- They also carry responses to those calls from the queue manager back to the application.

For more information about defining channels, see the following website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.ref.adm.doc/q085520_.htm

Example 9-30 shows the command to be used from the runmqsc console to create the SYSTEM.BKR.CONFIG server connection channel on the ITSOQM2 queue manager.

Example 9-30 Creating a server connection channel

```
DEFINE CHANNEL('SYSTEM.BKR.CONFIG') CHLTYPE(SVRCONN) REPLACE
```

Creating a local queue for the application

Example 9-31 shows the command to be ran from the runmqsc console of the ITSOQM2 queue manager.

Example 9-31 Creating a local queue for the ITSO Transport back-end application

```
DEFINE QLOCAL(ITSO.BROADCAST.IN)
```

For more information about defining local queues, see the following website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.ref.adm.doc/q085710_.htm

The ITSO.BROADCAST.IN queue on the ITSOQM2 queue manager is used for receiving broadcast messages from the ITSO Transport Corporation back-end office team. Messages from this queue are broadcast to the mobile application in trucks using IBM Integration Bus and the IBM MessageSight appliance.

Exception: To simplify this scenario, the channel authentication is disabled on the ITSOQM2 queue manager by using the following command in the runmqsc console:

```
ALTER QMGR CHLAUTH(DISABLED)
```

9.4.4 Configuring IBM Integration Bus

In this scenario, you use IBM Integration Bus to capture geographic location change messages from IBM MessageSight. All communication from IBM Integration Bus and IBM MessageSight is performed using the JMS protocol. IBM MessageSight acts as a JMS service provider, and IBM Integration Bus acts as a JMS client. The message flow created in this scenario serves the following purposes:

- Capture geographic location changes and submit them in a database.
- Run a timer-based flow, then identify the trucks that must be notified with notifications based on their geographic location. Notify all of the selected trucks.
- Send a broadcast notification to all of the trucks.

Use the following steps to configure IBM Integration Bus for this scenario:

1. Create an integration node.
2. Create an integration server.
3. Configure a JMS provider for IBM MessageSight.
4. Configure a Java Database Connectivity (JDBC) provider for IBM DB2.
5. Create an IBM Integration Bus application.
6. Deploy an IBM Integration Bus application.

Creating an integration node

You can create an integration node using the **`mqsicreatebroker`** command. Example 9-32 shows the syntax of this command.

Example 9-32 The mqsicreatebroker command syntax

```
mqsicreatebroker brokerName -q queueManagerName [-i generalDefaultUserId -a
generalDefaultPassword] [-g configurationChangeTimeout] [-k
internalConfigurationTimeout] [-w workPath] [-e sharedWorkPath] [-l
userLilPath] [-t] [-m] [-v statisticsMajorInterval] [-P httpListenerPort] [-c
icuConverterPath] [-y ldapPrincipal -z ldapCredentials] [-x userExitPath] [-o
operationMode] [-s adminSecurity] [-d MQService] [-b cachePolicy] [-r
cachePortRange] [-4 integrationRegistryHostname]
```

To learn more about all of the parameters for the **`mqsicreatebroker`** command, see the following website:

http://pic.dhe.ibm.com/infocenter/wmbhelp/v9r0m0/topic/com.ibm.etools.mft.doc/an28135_.htm

ITSOBK2 integration node on iib2.itso.transport.com

Example 9-33 shows the **mqsicreatebroker** command run on the `iib2.itso.transport.com` server to create an integration node called `ITSOBK2`.

Example 9-33 Creating the ITSOBK2 integration node

```
mqsicreatebroker ITSOBK2 -i <username> -a <password> -q ITSOQM2
```

Now that an integration node is created, you can start the node using the **mqsistart** command, as shown in Example 9-34.

Example 9-34 The syntax for the mqsistart command

```
mqsistart <integration node name>
```

To learn more about the **mqsistart** command, go to the following website:

http://pic.dhe.ibm.com/infocenter/wmbhelp/v9r0m0/topic/com.ibm.etools.mft.doc/an28135_.htm

The command used to start the `ITSOBK2` integration node is shown in Example 9-35.

Example 9-35 Start the ITSOBK2 integration node

```
mqsistart ITSOBK2
```

Creating an integration server

In this section, you create an integration server on the `ITSOBK2` integration node by issuing the **mqsicreateexecutiogroup** command. An *execution group* is a named grouping of message flows that is assigned to an integration node. The integration node enforces a degree of isolation between message flows in distinct integration servers by ensuring that they run in separate address spaces, or as unique processes.

Example 9-36 shows the syntax of the **mqsicreateexecutiongroup** command.

Example 9-36 The syntax of the mqsicreateexecutiongroup command

```
mqsicreateexecutiongroup brokerSpec -e egName [-w timeoutSecs] [-v
traceFileName]
```

To learn more about the **mqsicreateexecutiongroup** command, see the following website:

http://pic.dhe.ibm.com/infocenter/wmbhelp/v9r0m0/topic/com.ibm.etools.mft.doc/an26000_.htm

Create the ITSOEG integration server on the ITSOBK2 integration node. Example 9-37 shows the **mqsicreateexecutiongroup** command run on ITSOBK2.

Example 9-37 Creating the ITSOEG integration server on ITSOBK2

```
mqsicreateexecutiongroup ITSOBK2 -e ITSOEG
```

Figure 9-4 illustrates the ITSOEG integration server on the ITSOBK2 integration node.

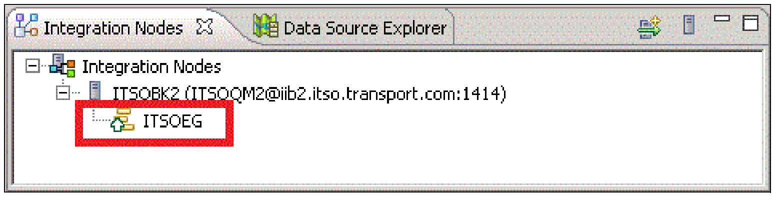


Figure 9-4 The ITSOEG integration node on ITSOBK2

Configuring a JMS provider for IBM MessageSight

An IBM Integration Bus node must be configured to use IBM MessageSight JMS client libraries. You can download IBM MessageSight client libraries from the following website:

http://tinyurl.com/q5cz7ck

Download the JMS client library for IBM MessageSight, and extract it into a file system accessible by IBM Integration Bus. In this scenario, the JMS libraries are placed in the /home/wmbadmin/ImaClient directory of the iib2.itso.transport.com server. Example 9-38 shows the JMS client libraries available in this location.

Example 9-38 IBM MessageSight JMS client libraries

```
$ pwd
/home/wmbadmin/ImaClient/jms/lib
$ ls -lrt
total 368
-rwxrwxrwx 1 wmbadmin mqbrkrs  77116 Jul 10 15:30 providerutil.jar
-rwxrwxrwx 1 wmbadmin mqbrkrs  25998 Jul 10 15:30 jms.jar
-rwxrwxrwx 1 wmbadmin mqbrkrs  22769 Jul 10 15:30 fscontext.jar
-rwxrwxrwx 1 wmbadmin mqbrkrs 209139 Jul 28 05:02 imaclientjms.jar
-rwxrwxrwx 1 wmbadmin mqbrkrs  29120 Jul 28 05:02 jmssamples.jar
```

After the ITSOBK2 integration node is created, it has a predefined, configurable service for the IBM MessageSight JMS provider named IBM_MessageSight.

Example 9-39 shows the predefined JMS provider that exists for IBM MessageSight in the `ITSOBK2` integration node.

Example 9-39 The IBM_MessageSight JMS provider in the ITSOBK2 integration node

```
$ mqsireportproperties ITSOBK2 -c JMSProviders  -o IBM_MessageSight -r
JMSProviders
  IBM_MessageSight
    clientAckBatchSize='0'
    clientAckBatchTime='0'
    connectionFactoryName=''
    initialContextFactory=''
    jarsURL='default_path'
    jmsAsyncExceptionHandling='false'
    jmsProviderXASupport='true'
    jndiBindingsLocation=''
    jndiEnvironmentParms='default_none'
    nativeLibs='default_Path'
    proprietaryAPIAttr1='default_none'
    proprietaryAPIAttr2='default_none'
    proprietaryAPIAttr3='default_none'
    proprietaryAPIAttr4='default_none'
    proprietaryAPIAttr5='default_none'
    proprietaryAPIHandler='default_none'
```

The **`mqsichangeproperties`** command is used to update the `IBM_MessageSight` JMS provider with the location of the IBM MessageSight JMS client libraries.

Example 9-40 shows the syntax of the **`mqsichangeproperties`** command.

Example 9-40 The syntax of the mqsichangeproperties command

```
mqsichangeproperties BrokerName ( -b httplistener | -b securitycache | -b
servicefederation | -b webadmin | -b cachemanager | -e ExecutionGroupLabel | -c
ConfigurableService ) -o ObjectName -n PropertyName ( -v PropertyValue | -d |
-p Path/Filename ) [-f]
```

To learn more about the **`mqsichangeproperties`** command, see the following website:

http://pic.dhe.ibm.com/infocenter/wmbhelp/v9r0m0/topic/com.ibm.etools.mft.doc/an09140_.htm

Example 9-41 shows the **mqsichangeproperties** command to configure the IBM_MessageSight JMS provider with the location of the IBM MessageSight JMS client libraries.

Example 9-41 Configuring JMS provider for IBM MessageSight

```
mqsichangeproperties ITSOBK2 -c JMSProviders -o IBM_MessageSight -n jarsURL -v
/home/wmbadmin/ImaClient/jms/lib
```

Example 9-42 shows the JMS provider properties after the JMS client location is set.

Example 9-42 The IBM MessageSight JMS provider set to use JMS client libraries

```
$ mqsireportproperties ITSOBK2 -c JMSProviders  -o IBM_MessageSight -r
JMSProviders
  IBM_MessageSight
    clientAckBatchSize='0'
    clientAckBatchTime='0'
    connectionFactoryName=''
    initialContextFactory=''
    jarsURL='/home/wmbadmin/ImaClient/jms/lib'
    jmsAsyncExceptionHandling='false'
    jmsProviderXASupport='true'
    jndiBindingsLocation=''
    jndiEnvironmentParms='default_none'
    nativeLibs='default_Path'
    proprietaryAPIAttr1='default_none'
    proprietaryAPIAttr2='default_none'
    proprietaryAPIAttr3='default_none'
    proprietaryAPIAttr4='default_none'
    proprietaryAPIAttr5='default_none'
    proprietaryAPIHandler='default_none'
```

Configuring a JDBC provider for IBM DB2

In this section, you create a new configurable service to configure a JDBC provider for IBM DB2. However, before a new configurable service can be created, a security identity must be created to access the database. This can be done by issuing the **mqsisetdbparms** command.

Example 9-43 shows the syntax of the **mqsisetdbparms** command.

Example 9-43 The syntax of the mqsisetdbparms command

```
mqsisetdbparms <brokerName> -n <resource> [-u <userId>] [-p <password>] [-f]
```

To learn more about the **mqsisetdbparms** command, see the following website:

http://pic.dhe.ibm.com/infocenter/wmbhelp/v9r0m0/topic/com.ibm.etools.mft.doc/an09155_.htm

Example 9-44 shows the ITSODBId security identity created to access the database from the ITSOBK2 integration node.

Example 9-44 Creating the ITSODBId security identity

```
$ mqsisetdbparms ITSOBK2 -n jdbc::ITSODBId -u db2inst1 -p ********
BIP8071I: Successful command completion.
```

To create a new JDBC provider configurable service, use the **mqsicreateconfigurableservice** command. Example 9-45 shows the syntax of this command.

Example 9-45 The syntax of the mqsicreateconfigurableservice command

```
mqsicreateconfigurableservice brokerName -c configurableService -o object ( -n
name -v value )
```

To learn more about the **mqsicreateconfigurableservice** command, see the following website:

http://pic.dhe.ibm.com/infocenter/wmbhelp/v9r0m0/topic/com.ibm.etools.mft.doc/an37200_.htm

Example 9-46 shows the command to create a new IBM DB2 JDBC provider configurable service named ITSODB.

Example 9-46 Creating the ITSODB DB2 JDBC provider

```
$ mqsicreateconfigurableservice ITSOBK2 -c JDBCProviders -o ITSODB -n
connectionUrlFormat,databaseName,databaseVersion,databaseType,jarsURL,portNumbe
r,securityIdentity,serverName,type4DatasourceClassName,type4DriverClassName -v
'jdbc:db2://db.itso.transport.com:50001/ITSODB:user=db2inst1;password=*******;'
,'ITSODB','10.1','DB2 Universal
Database','/opt/ibm/db2/V10.1/java','50001','ITSODBId','db.itso.transport.com',
'com.ibm.db2.jcc.DB2XADataSource','com.ibm.db2.jcc.DB2Driver'
BIP8071I: Successful command completion.
```

Example 9-47 shows the properties of the `ITSODB` JDBC provider configurable service.

Example 9-47 The ITSODB JDBC provider properties

```
$ mqsireportproperties ITSOBK2 -c JDBCProviders -o ITSODB -r
JDBCProviders
  ITSODB
connectionUrlFormat='jdbc:db2://db.itso.transport.com:50001/ITSODB:user=db2inst
1;password=*******;'
    connectionUrlFormatAttr1=''
    connectionUrlFormatAttr2=''
    connectionUrlFormatAttr3=''
    connectionUrlFormatAttr4=''
    connectionUrlFormatAttr5=''
    databaseName='ITSODB'
    databaseSchemaNames='useProvidedSchemaNames'
    databaseType='DB2 Universal Database'
    databaseVersion='10.1'
    description='default_Description'
    environmentParms='default_none'
    jarsURL='/opt/ibm/db2/V10.1/java'
    jdbcProviderXASupport='jdbcProviderXASupport'
    maxConnectionPoolSize='0'
    portNumber='50001'
    securityIdentity='ITSODBId'
    serverName='db.itso.transport.com'
    type4DatasourceClassName='com.ibm.db2.jcc.DB2XADataSource'
    type4DriverClassName='com.ibm.db2.jcc.DB2Driver'
BIP8071I: Successful command completion.
```

Creating an IBM Integration Bus application

In this section, you create a new application named `ITSOGeoNotificationBroadcast` for IBM Integration Bus. Before that, you start the IBM Integration Toolkit 9.0.0.0, as described in the following procedure:

1. On the development workstation, select **Start** → **All Programs** → **IBM Integration Toolkit** → **IBM Integration Toolkit 9.0.0.0**.
2. Click **Integration Toolkit 9.0.0.0** to start the Eclipse GUI.
3. In the workspace wizard, enter the workspace location and click **OK.**
4. After the workspace is launched, close the **Welcome** tab and go to the Integration Development perspective.

5. Figure 9-5 illustrates the Application Development view in the upper left corner of the Integration Development perspective. Click **New Application** to open the New Application wizard.

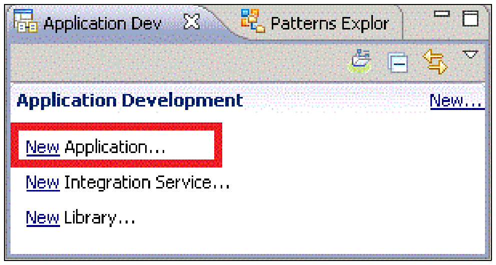
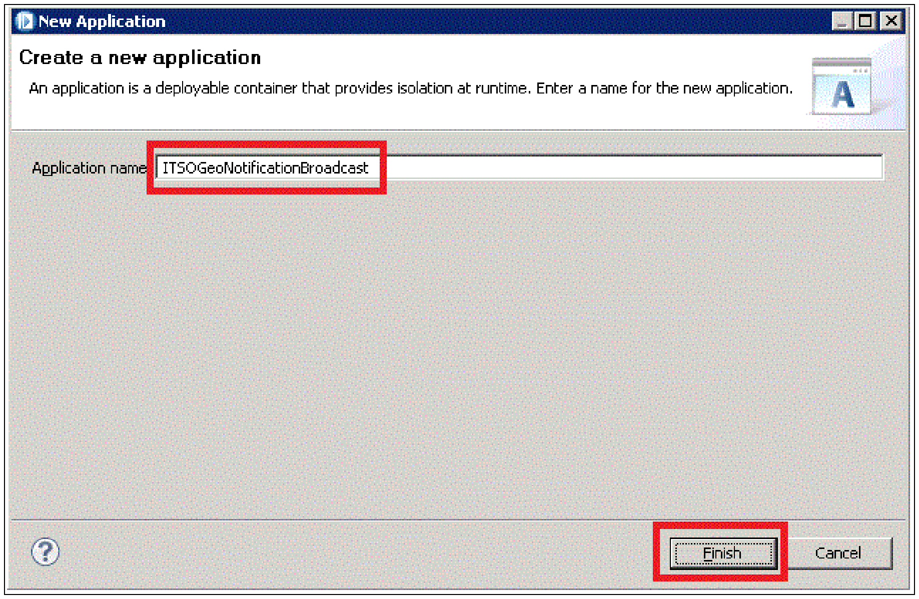


Figure 9-5   Creating a new integration application project

6. Figure 9-6 shows the New Application wizard. Enter `ITSOGeoNotificationBroadcast` as the Application name, and then click **Finish**.



Create a new application

An application is a deployable container that provides isolation at runtime. Enter a name for the new application.

Application name: ITSOGeoNotificationBroadcast

Figure 9-6   Creating the ITSOGeoNotificationBroadcast application

7. Figure 9-7 shows the new IBM Integration Toolkit application project, ITSOGeoNotificationBroadcast, created in the Application Development view. Click **New** → **Message Flow.**

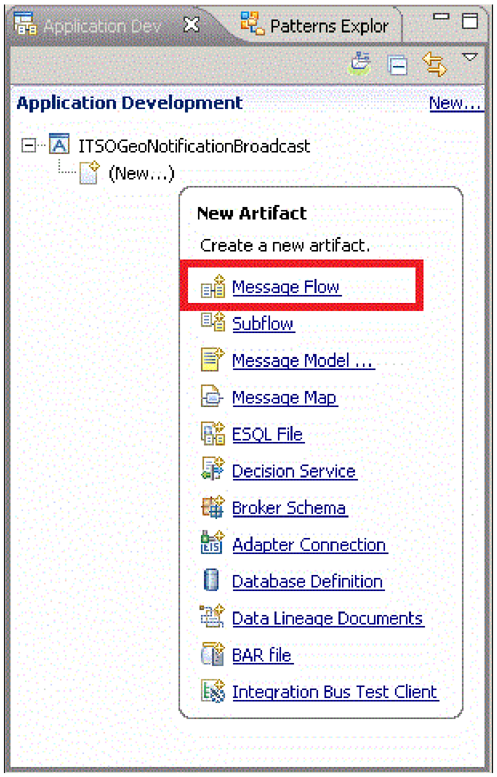


Figure 9-7 Creating a new message flow

8. Figure 9-8 shows the New Message Flow creation wizard. Enter the Message flow name `GeoLocationNotificationMessageFlow`, and then click **Finish**.

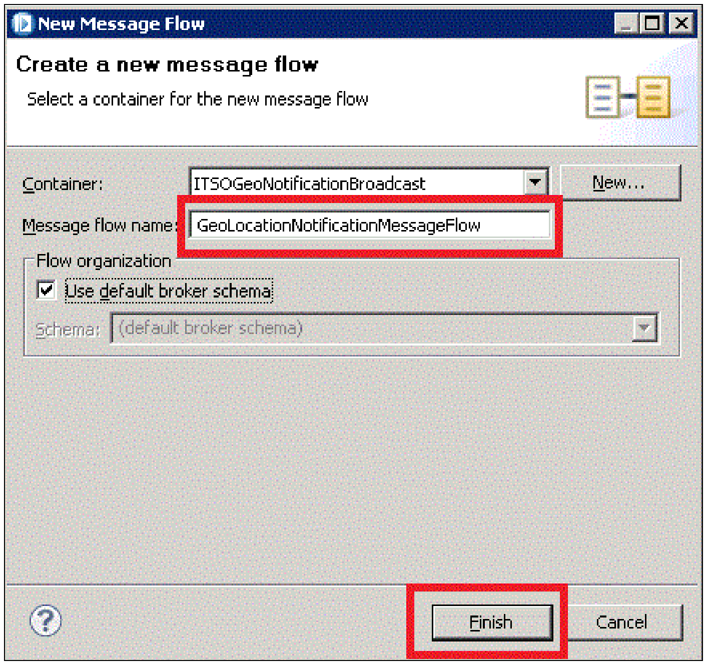

New Message Flow
Create a new message flow
Select a container for the new message flow
Container:
ITSOGeoNotificationBroadcast
New...
Message flow name:
GeoLocationNotificationMessageFlow
Flow organization
Use default broker schema
Schema:
(default broker schema)
Finish
Cancel

Figure 9-8 Creating the GeoLocationNotificationMessageFlow message flow

9. Figure 9-9 shows the message flow editor for the GeoLocationNotificationMessageFlow message flow. Proceed by dragging the nodes listed in Table 9-10 on page 256.

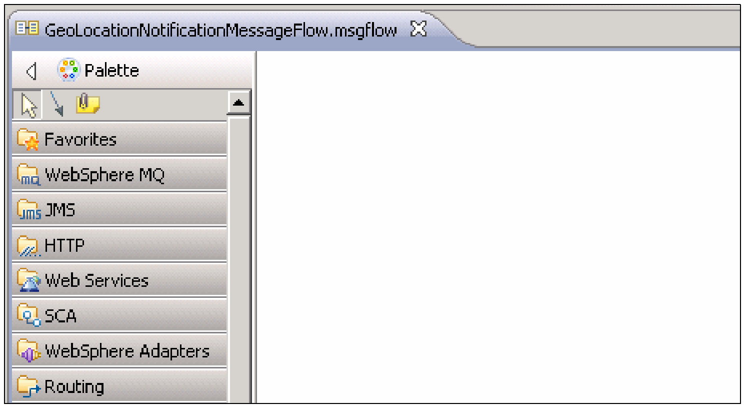


Figure 9-9 Creating GeoLocationNotificationMessageFlow message flow functionality

Table 9-10 lists the node name, node type, and a brief description about that node. Drag these nodes from the palette and rename them to the corresponding node name in the message flow editor area. This message flow has three different parts, each providing a different functionality.

Table 9-10 Nodes in the GeoLocationNotificationMessageFlow message flow

Node name	Node type	Description
Message flow - part 1: This part of the message flow receives GeoLocation updates on the `/itso/driver/geolocation/+` topic, and then inserts details into the database.		
`RecordGeographicLocation`	JMSInput node	This node subscribes to a topic on IBM MessageSight, and receives geographic location change messages.
`GeoLocationToDatabase`	Java Compute node	This Java compute node inserts the changes in geographic location into a database.
Message flow - part 2: This part of the message flow runs periodically, extracts unique truck numbers from the database, and sends notifications to selected trucks. The business rule illustrated here is simple, and is for scenario purposes only.		
`Geographic Alerts`	Timeout notification node	This node triggers the part of the flow after every 600 seconds.
`BusinessRuleTruckSelection`	Java compute node	This Java compute node selects unique truck numbers from the database for trucks that are updating changes in the geographic location.
`CreateAlertTopic`	Java compute node	Sends a notification to trucks that were selected by the previous node.
`SendAlert`	JMS output node	This JMS output node publishes notifications to specific trucks.

Node name	Node type	Description
Message flow - part 3: A broadcast is sent to all of the trucks that are in the field and have connectivity with IBM MessageSight. The topic on which the notifications are published is `/itso/driver/notification/AllTrucks`.		
`BroadcastAlerts`	WebSphere MQ Input node	This WebSphere MQ input node receives messages on WebSphere MQ queue, which need to be broadcast.
`WebSphere MQ to JMS`	WebSphere MQ JMS transform node	WebSphere MQ to JMS transformation.
`setRetain`	Java compute node	Set retain flag on JMS message.
`setBroadcast`	JMS output node	This JMS output node publishes notifications to all trucks.

Table 9-11 shows the terminals used to connect the nodes listed in Table 9-10 on page 256.

Table 9-11 Connection between nodes

Source node	Output terminal	Destination node	Input terminal
Message flow - part 1			
`RecordGeographicLocation`	Out	`GeoLocationToDatabase`	In
Message flow - part 2			
`Geographic Alerts`	Out	`BusinessRuleTruckSelection`	In
`BusinessRuleTruckSelection`	Out	`CreateAlertTopic`	In
`CreateAlertTopic`	Out	`SendAlert`	In
Message flow - part 3			
`BroadcastAlerts`	Out	`WebSphere MQ to JMS`	In
`WebSphere MQ to JMS`	Out	`setRetain`	In
`setRetain`	Out	`setBroadcast`	In

10. After the nodes are renamed and connected using the details mentioned in Table 9-10 on page 256 and Table 9-11 on page 257, the connected nodes in the message flow looks similar to those shown in Figure 9-10.

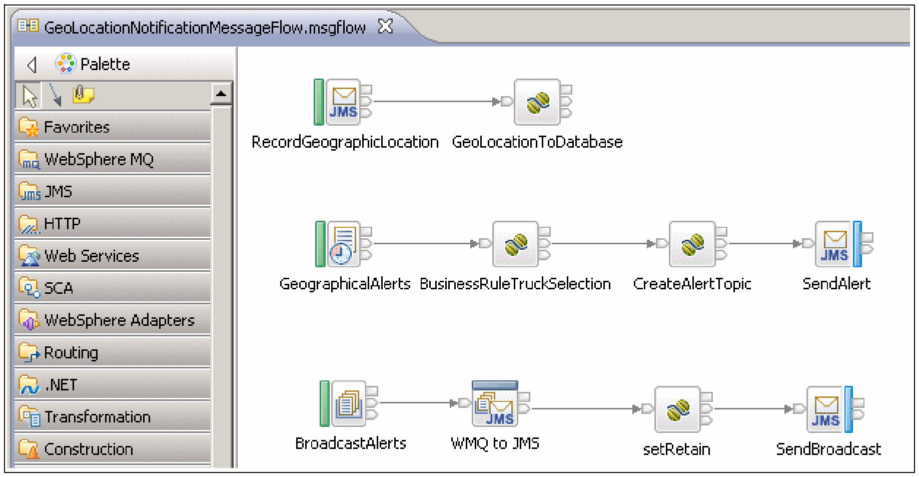


Figure 9-10 Connected nodes in the GeoLocationNotificationMessageFlow message flow

11. Configure the properties of the nodes as shown in Table 9-12.

Table 9-12 Node properties

Node name	Property tab	Property name	Property value
Message flow - part 1			
`RecordGeographicLocation`	Basic	Subscription Topic	`GeoTopic`
`RecordGeographicLocation`	Basic	Durable subscription ID	`itsoesb`
`RecordGeographicLocation`	JMS Connection	Location JNDI bindings	`file:///home/wmbadmin`
`RecordGeographicLocation`	JMS Connection	Connection factory name	`ITSOConnectFactory`
`RecordGeographicLocation`	Input Message parsing	Message domain	`BLOB`
`GeoLocationToDatabase`	Basic	Java class	`GeoLocationPersistence`

Node name	Property tab	Property name	Property value
Message flow - part 2			
Geographic Alerts	Basic	Unique identifier	GeoAlert
Geographic Alerts	Basic	Timeout interval in seconds	600
BusinessRule TruckSelection	Basic	Java class	BusinessRule TruckSelection
CreateAlertTopic	Basic	Java class	CreateTopic
SendAlert	Basic	Publication topic	Notification
SendAlert	Basic	Send to destination list in local environment	true
SendAlert	JMS Connection	Location JNDI bindings	file:///home /wmbadmin
SendAlert	JMS Connection	Connection factory name	ITSOConnect Factory
Message flow - part 3			
BroadcastAlerts	Basic	Queue name	ITSO.BROADCAST. IN
WebSphere MQ to JMS	N/A	N/A	N/A
setRetain	Basic	Java class	setRetainFlag
SendBroadcast	Basic	Publication topic	Broadcast
SendBroadcast	JMS Connection	Location JNDI bindings	file:///home /wmbadmin
SendBroadcast	JMS Connection	Connection factory name	ITSOConnect Factory

12. Create a `.bindings` file in the `/home/wmbadmin` directory on the `iib2.itso.transport.com` server for JMS Topic connection factories and topic details. Example 9-48 shows the contents of the `.bindings` file.

Example 9-48 The .bindings file for JMS nodes in the message flow

```
ITSOConnectFactory/ClassName=com.ibm.ima.jms.impl.ImaConnectionFactory
ITSOConnectFactory/FactoryName=com.ibm.ima.jms.impl.ImaConnectionFactory
ITSOConnectFactory/RefAddr/0/Encoding=String
ITSOConnectFactory/RefAddr/0/Type=Port
ITSOConnectFactory/RefAddr/0/Content=16105
ITSOConnectFactory/RefAddr/1/Encoding=String
ITSOConnectFactory/RefAddr/1/Type=Server
ITSOConnectFactory/RefAddr/1/Content=ima1.itso.transport.com
ITSOConnectFactory/RefAddr/2/Encoding=String
ITSOConnectFactory/RefAddr/2/Content=common
ITSOConnectFactory/RefAddr/2/Type=ObjectType
GeoTopic/ClassName=com.ibm.ima.jms.impl.ImaTopic
GeoTopic/FactoryName=com.ibm.ima.jms.impl.ImaTopic
GeoTopic/RefAddr/0/Encoding=String
GeoTopic/RefAddr/0/Type=Name
GeoTopic/RefAddr/0/Content=/itso/driver/geolocation/\+
Notification/ClassName=com.ibm.ima.jms.impl.ImaTopic
Notification/FactoryName=com.ibm.ima.jms.impl.ImaTopic
Notification/RefAddr/0/Encoding=String
Notification/RefAddr/0/Type=Name
Notification/RefAddr/0/Content=/itso/driver/notification
Broadcast/ClassName=com.ibm.ima.jms.impl.ImaTopic
Broadcast/FactoryName=com.ibm.ima.jms.impl.ImaTopic
Broadcast/RefAddr/0/Encoding=String
Broadcast/RefAddr/0/Type=Name
Broadcast/RefAddr/0/Content=/itso/driver/notification/AllTrucks
```

13. Create a Java class for the `GeoLocationToDatabase` Java compute node. Double-click this Java compute node and create a Java class named `GeoLocationPersistence`. Also enter `ITSOGeoNotificationandBroadcastJava` as the source folder name. Example 9-49 shows the Java code for this class.

Example 9-49 Java code for the GeoLocationToDatabase Java compute node

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.StringTokenizer;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.broker.plugin.MbUserException;

public class GeoLocationPersistence extends MbJavaComputeNode {

public void evaluate(MbMessageAssembly inAssembly) throws MbException {
MbOutputTerminal out = getOutputTerminal("out");

String geoMessageElement[] =  new String[4];

MbMessage inMessage = inAssembly.getMessage();
MbMessageAssembly outAssembly = null;
try {
  // create new message as a copy of the input
  MbMessage outMessage = new MbMessage(inMessage);
  outAssembly = new MbMessageAssembly(inAssembly, outMessage);

  MbElement inputRoot   = inMessage.getRootElement();
  byte[] MessageBodyByteArray  = (byte[])
(inputRoot.getFirstElementByPath("/BLOB/BLOB").getValue());
  String MessageBodyString     = new String(MessageBodyByteArray);
  String JMSTopic              =
inputRoot.getFirstElementByPath("/JMSTransport/Transport_Folders/Header_Val
ues/JMSDestination").getLastChild().getValueAsString();
  String truckNumber[]         = JMSTopic.split("/");
  StringTokenizer st = new StringTokenizer(MessageBodyString,",");
  int index=0;
  while (st.hasMoreTokens()) {
   geoMessageElement[index++]=st.nextToken();
  }

Connection conn =
getJDBCType4Connection("ITSODB",JDBC_TransactionType.MB_TRANSACTION_AUTO);
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
stmt.execute("INSERT INTO DB2INST1.TRUCKGEOLOCATION (TIMSTAMP, DRIVER_NAME,
TRUCK_NUMBER , LONGITUDE, LATITUDE) VALUES('"+
      geoMessageElement[0]+"','"+
      geoMessageElement[1]+"','"+
      truckNumber[4] +"','"+
      geoMessageElement[2]+"','"+
      geoMessageElement[3]+"')");

  } catch (MbException e) {
   throw e;
  } catch (RuntimeException e) {
   throw e;
  } catch (Exception e) {
        throw new MbUserException(this, "evaluate()", "", "",
e.toString(),null);
  }
 out.propagate(outAssembly);
}
}
```

14. Create a Java class for the `BusinessRuleTruckSelection` Java compute node. Double-click this Java compute node and create a Java class named `BusinessRuleTruckSelection`. Also enter `ITSOGeoNotificationandBroadcastJava` as the source folder name. Example 9-50 shows the Java code for this class.

Example 9-50 Java code for the BusinessRuleTruckSelection Java compute node

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ResourceBundle;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.broker.plugin.MbUserException;


public class BusinessRuleTruckSelection extends MbJavaComputeNode {

public void evaluate(MbMessageAssembly inAssembly) throws MbException {
MbOutputTerminal out = getOutputTerminal("out");

MbMessage inMessage = inAssembly.getMessage();
MbMessageAssembly outAssembly = null;
try {
   // create new message as a copy of the input
  MbMessage outMessage = new MbMessage(inMessage);
  outAssembly = new MbMessageAssembly(inAssembly, outMessage);

  String alertMessage = AlertSelection();
  Connection conn =
getJDBCType4Connection("ITSODB",JDBC_TransactionType.MB_TRANSACTION_AUTO);
  Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
  ResultSet rs   = stmt.executeQuery("SELECT UNIQUE TRUCK_NUMBER FROM
TRUCKGEOLOCATION");
```

```
  while(rs.next())
  {
  MbElement localEnv     =
outAssembly.getLocalEnvironment().getRootElement();
  MbElement TruckNumber  = localEnv.getFirstElementByPath("TruckNumber");
  MbElement AlertMessage = localEnv.getFirstElementByPath("AlertMessage");

  if(TruckNumber!=null)
  {
    TruckNumber.detach();
  }
  if(AlertMessage!=null)
  {
    AlertMessage.detach();
  }

localEnv.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,"TruckNumber",r
s.getString("TRUCK_NUMBER"));

localEnv.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,"AlertMessage",
alertMessage);
  out.propagate(outAssembly,true);
  }
  } catch (MbException e) {
   e.printStackTrace();
   throw e;
  } catch (RuntimeException e) {
   e.printStackTrace();
   throw e;
  } catch (Exception e) {
   e.printStackTrace();
   throw new MbUserException(this, "evaluate()", "", "",
e.toString(),null);
   }
 }

private String AlertSelection()
{
  String message = new String();
  int randomNumber = (int)(Math.random() * (10 - 1) + 1);
  ResourceBundle rb = ResourceBundle.getBundle("Messages");
  message = rb.getString(Integer.toString(randomNumber));
  return message;
}
}
```

Note: For a Java compute node to build random messages, a Messages file is created and is referred to from the Java ResourceBundle API. The following list shows the contents of the `Messages_en.properties` file:

1 = Inclement weather in your area, drive safely and inform main office if you expect delivery to be delayed.

2 = Highway construction on I-75, use alternative route if possible.

3 = Tornado Warnings issued in your area, take cover immediately.

4 = Truck malfunction in your area, inform front office if you are available to pick up load for delivery.

5 = Traffic Jam reported on Highway 5, use alternative route.

6 = Accident on Interstate 10, take alternative route.

7 = Heavy snow - use precaution and engage inclement weather driving gear.

8 = Bay bridge is closed, plan alternative route.

9 = Highway 92 closed, take alternative route.

10 = Return to station, additional cargo is ready for delivery.

Random messages are returned from the `AlertSelection()` method.

15. Create a Java class for the `CreateAlertTopic` Java compute node. Double-click this Java compute node and create a Java class named `CreateTopic`. Also enter `ITSOGeoNotificationandBroadcastJava` as the source folder name. Example 9-51 shows the Java code for this class.

Example 9-51 Java code for Java compute node CreateAlertTopic

```
import com.ibm.broker.plugin.MbBLOB;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.broker.plugin.MbUserException;

public class CreateTopic extends MbJavaComputeNode {

public void evaluate(MbMessageAssembly inAssembly) throws MbException {
MbOutputTerminal out = getOutputTerminal("out");
MbMessage inMessage = inAssembly.getMessage();
MbMessageAssembly outAssembly = null;
try {
```

```
MbMessage outMessage = new MbMessage(inMessage);
outAssembly = new MbMessageAssembly(inAssembly, outMessage);
MbElement localEnv = outAssembly.getLocalEnvironment().getRootElement();
MbElement dest = localEnv.createElementAsFirstChild(MbElement.TYPE_NAME,
"Destination", null);
MbElement jmsDestList = dest.createElementAsFirstChild(MbElement.TYPE_NAME,
"JMSDestinationList", null);
MbElement destData =
jmsDestList.createElementAsFirstChild(MbElement.TYPE_NAME,
"DestinationData", null);
destData.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
"DestinationName",
"/itso/driver/notification/"+localEnv.getFirstElementByPath("TruckNumber").
getValueAsString());
destData.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
"DestinationType", "Topic");
MbElement root = outMessage.getRootElement();
MbElement msgElement = root.getLastChild();
msgElement.detach();
MbElement BLOB = root.createElementAsLastChild(MbBLOB.PARSER_NAME);
BLOB.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB",
localEnv.getFirstElementByPath("AlertMessage").
        getValueAsString().getBytes());
outMessage.finalizeMessage(MbMessage.FINALIZE_VALIDATE);
} catch (MbException e) {
   throw e;
} catch (RuntimeException e) {
        throw e;
} catch (Exception e) {
        throw new MbUserException(this, "evaluate()", "", "",
e.toString(),null);
}
out.propagate(outAssembly);
}
}
```

16. Create a Java class for the `setRetain` Java compute node. Double-click this Java compute node and create a Java class named `setRetainFlag`. Also enter `ITSOGeoNotificationandBroadcastJava` as the source folder name.

Note: The purpose of this class is to set the JMS Retain flag on JMS publications sent through the JMS output node on the Broadcast topic. IBM MessageSight considers the JMS Retain flag while distributing messages using the MQTT protocol.

Example 9-52 shows the Java code for the `setRetainFlag` class.

Example 9-52 Java code for Java compute node setRetain

```
import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.broker.plugin.MbUserException;
import com.ibm.broker.plugin.MbXML;

public class setRetainFlag extends MbJavaComputeNode {

public void evaluate(MbMessageAssembly inAssembly) throws MbException {
MbOutputTerminal out = getOutputTerminal("out");
MbMessage inMessage = inAssembly.getMessage();
MbMessageAssembly outAssembly = null;
try {
MbMessage outMessage = new MbMessage(inMessage);
outAssembly = new MbMessageAssembly(inAssembly, outMessage);
MbElement outputRoot = outMessage.getRootElement();
MbElement applicationFolder =
outputRoot.getFirstElementByPath("/JMSTransport/Transport_Folders/Applicati
on_Properties");
MbElement JMS_IBM_Retain    =
applicationFolder.createElementAsLastChild(MbXML.ELEMENT,"JMS_IBM_Retain",1
);
JMS_IBM_Retain.createElementAsFirstChild(MbXML.ATTRIBUTE, "dt", "i4");
} catch (MbException e) {
  throw e;
} catch (RuntimeException e) {
  throw e;
} catch (Exception e) {
  throw new MbUserException(this, "evaluate()", "", "", e.toString(),null);
}
  out.propagate(outAssembly);
}
}
```

Deploying an IBM Integration Bus application

Before deployment, a broker archive (BAR) file must be created to compile the message flow. Use the following procedure:

1. Create a new BAR file, as shown in Figure 9-11.

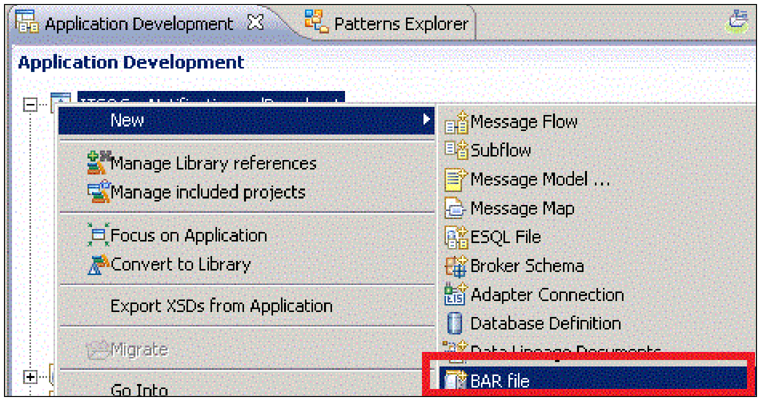
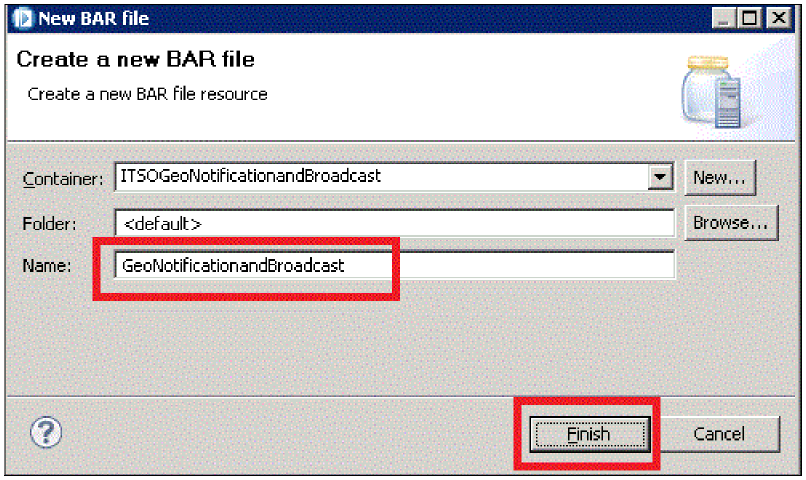


Figure 9-11 Creating a new BAR file

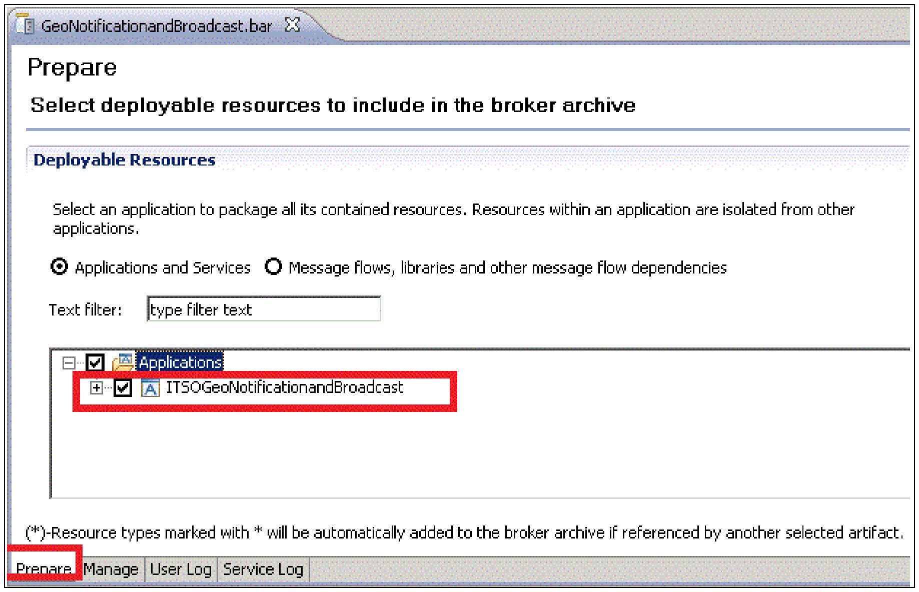
2. The New BAR file wizard opens. Enter `GeoNotificationandBroadcast` as the BAR file name and click **Finish**. Figure 9-12 shows the New BAR file wizard.



Container: ITSOGeoNotificationandBroadcast
Folder: <default>
Name: GeoNotificationandBroadcast

Figure 9-12 The New BAR file wizard

3. After the BAR file is created, the wizard opens in the BAR file editor, as shown in Figure 9-13. Click the **Prepare** tab in this editor and select the **ITSOGeoNotificationBroadcast** application to be compiled in this BAR file.



Prepare

Select deployable resources to include in the broker archive

Deployable Resources

Select an application to package all its contained resources. Resources within an application are isolated from other applications.

[✓ Applications and Services] Message flows, libraries and other message flow dependencies

Text filter: type filter text

[✓] Applications
- [✓] ITSOGeoNotificationandBroadcast

(*)-Resource types marked with * will be automatically added to the broker archive if referenced by another selected artifact.

Figure 9-13 Select an application in the BAR file editor

4. Click the **Manage** tab and then click **Rebuild**, as shown in Figure 9-14. If there are no compilation errors, the BAR file is built. Save the BAR file before deployment.

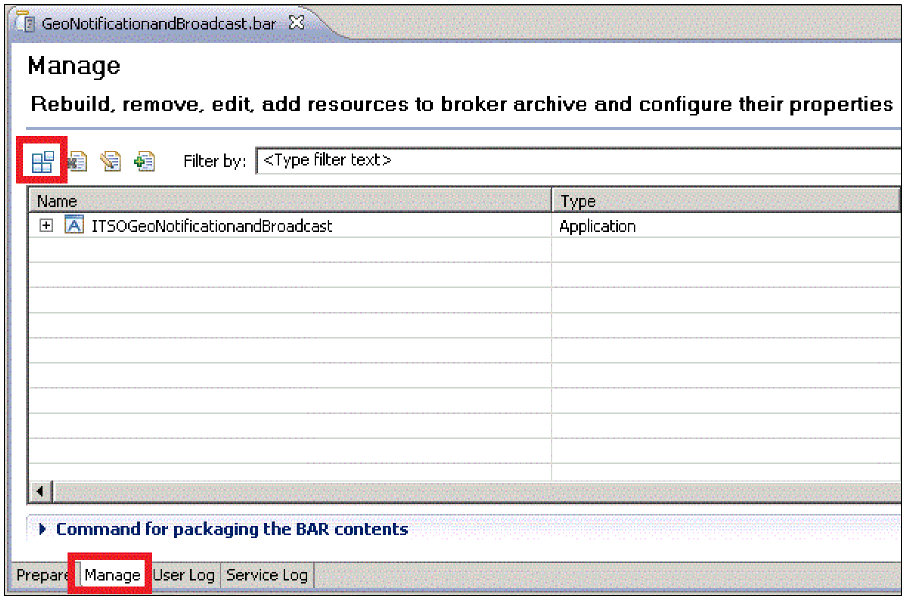

GeoNotificationandBroadcast.bar
Manage
Rebuild, remove, edit, add resources to broker archive and configure their properties
Filter by: <Type filter text>
Name
Type
ITSOGeoNotificationandBroadcast
Application
Command for packaging the BAR contents
Prepare
Manage
User Log
Service Log

Figure 9-14 Building the BAR file

5. After the BAR file is created, deploy it on the `ITSOEG` integration server on the `ITSOBK2` integration node. To deploy it from the Integration toolkit, drag the BAR file on `ITSOEG`. Figure 9-15 shows the ITSOGeoNotificationBroadcast application deployed on `ITSOEG`.

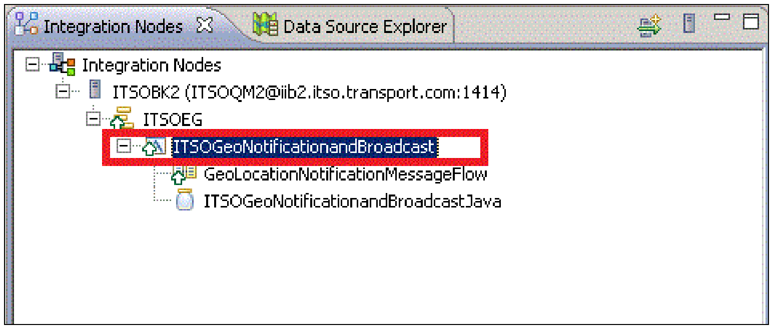

Integration Nodes
Data Source Explorer
Integration Nodes
ITSOBK2 (ITSOQM2@iib2.itso.transport.com:1414)
ITSOEG
ITSOGeoNotificationandBroadcast
GeoLocationNotificationMessageFlow
ITSOGeoNotificationandBroadcastJava

Figure 9-15 An application deployed on the ITSOEG integration server

9.5 Testing the scenarios

To test this scenario, you take three trucks and three truck drivers. Table 9-13 shows the current status of their assignments.

Table 9-13 Current delivery assignments for different trucks

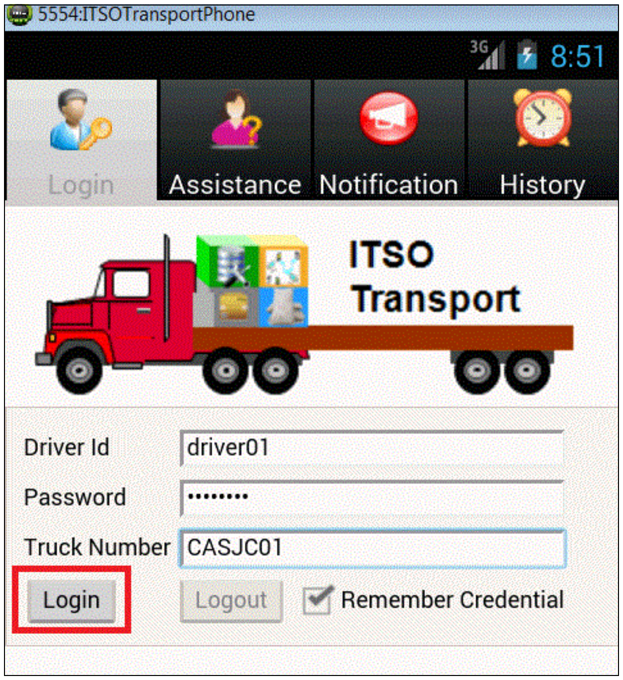
Truck number	Driver user name	Current assignment
CASJC01	driver01	Delivering groceries from San Jose, California (CA) to Santa Cruz, CA. The geographic location of this truck is always changing.
CASJC02	driver02	This truck has not yet started for the assignment, but the driver has logged in to the application and will start for delivery soon. The geographic location for this truck is not changing for now.
CASJC03	driver03	This truck is scheduled for delivery in the next few hours, and the driver has yet to check in.

Use the MQTT hybrid Android application developed using IBM Worklight to test this scenario.

Important: This scenario uses the following assumptions:

- All of the trucks are fitted with an Android mobile device.
- The mobile application is installed on all of the trucks in the fleet.

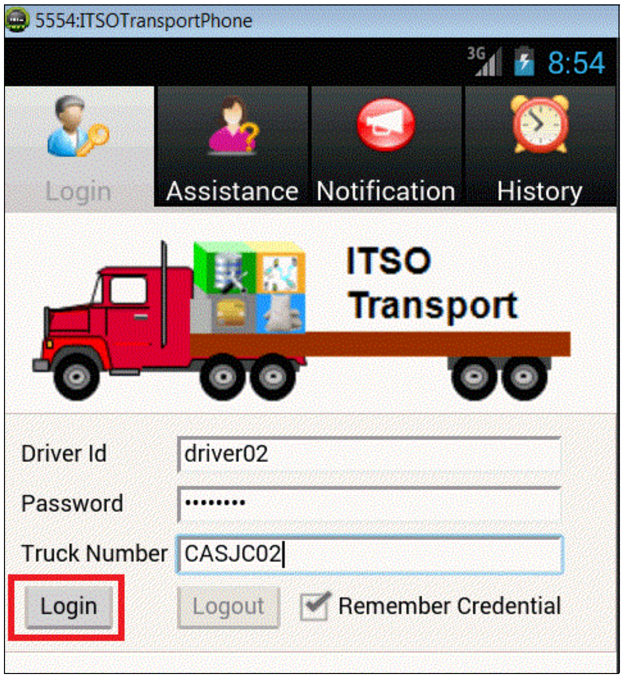
Figure 9-16 illustrates that driver01 is logged in to the mobile application on the CASJC01 truck. This Truck is currently on its way to delivery from San Jose, CA to Santa Cruz, CA.



Driver Id: driver01
Password: ••••••••
Truck Number: CASJC01
Remember Credential ✓

Figure 9-16 Driver01 logged in to the mobile application on CASJC01

Figure 9-17 illustrates that driver02 is logged in to the mobile application on the CASJC02 truck. This truck has still not left the ITSO Transport facility, and no geographic location changes are emitted from the mobile application.



Driver Id: driver02
Password: ••••••••
Truck Number: CASJC02
☑ Remember Credential

Figure 9-17 Driver02 logged in to the mobile application on CASJC02

9.5.1 Capturing and publishing geographic location changes

Detailed explanations about the MQTT hybrid mobile application are provided in Chapter 5, “MQTT with mobile platforms” on page 99. What we demonstrate here is that constant geographic location changes for the trucks are published by mobile application on a topic to IBM MessageSight.

IBM Integration Bus message flow has already subscribed to receive these publications. After a new publication is received, IBM Integration Bus submits the longitude and latitude location of the truck, along with the truck number and a time stamp into the TRUCKGEOLOCATION table in the ITSODB database.

Figure 9-18 on page 273 shows the part of the message flow on the IBM Integration Bus application that captures the geographic location change publications and insert details into the database.

More details about this scenario are included in section 9.3.1, “Push notification to selected mobile applications” on page 229.

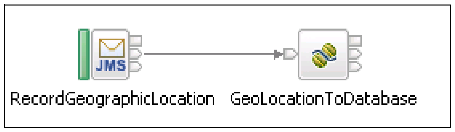
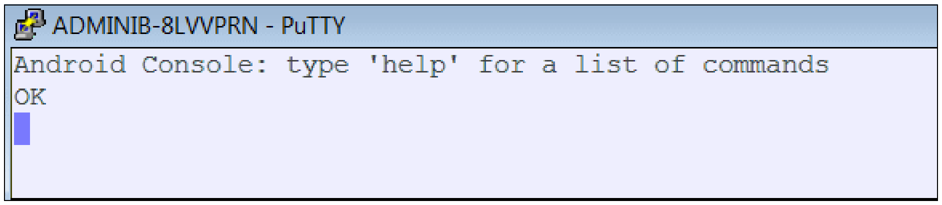


Figure 9-18 Geographic location changes captured in IBM Integration Bus

Because the application is running on an Android emulator for testing purposes, you use Android commands to simulate geographic location changes. The current emulator from where the CASJC01 truck is simulated has opened telnet port 5554. Figure 9-19 illustrates the telnet session for the Android emulator on localhost port 5554.



```
Android Console: type 'help' for a list of commands
OK
```

Figure 9-19 Android emulator telnet session

Example 9-53 shows the geographic location changes simulated on the Android emulator, where the driver has logged in with the user name driver01 and the truck number CASJC01.

Example 9-53 Geographic location changes for the CASJC01 truck

```
geo fix 123.34 -34.23
OK
geo fix 123.34 -34.53
OK
geo fix 123.34 -34.75
OK
```

Example 9-54 shows that the location changes simulated in Example 9-53 are now inserted into the TRUCKGEOLOCATION table on ITSODB.

Example 9-54 Geographic location changes inserted into database

```
db2 => select * from TRUCKGEOLOCATION

TIMSTAMP                   DRIVER_NAME TRUCK_NUMBER LONGITUDE
LATITUDE
-------------------------- ----------- ------------ ---------------------------
```

```
2013-09-17-09.29.21.912000 driver01    CASJC01      123.33999999999999   -34.22
2013-09-17-09.29.53.892000 driver01    CASJC01      123.33999999999999   -34.53
2013-09-17-09.30.00.389000 driver01    CASJC01      123.33999999999999   -34.75
```

This shows that any geographic location changes for a truck are captured by the mobile application and recorded in the back-end database.

9.5.2 Push notification for selected trucks

As seen in 9.5.1, "Capturing and publishing geographic location changes" on page 272, only the CASJC01 truck is sending geographic location changes. Figure 9-20 shows the part of the message flow in the IBM Integration Bus application that runs every 600 seconds. It scans for unique truck numbers in the TRUCKGEOLOCATION table, and then sends random alert messages to the mobile application in the trucks.

Note: The business logic to select trucks for alert notification is simple, and for illustration purpose only. Currently, all trucks that have recorded their geographic location will be sent a notification.

The mobile applications in the trucks have subscribed to receive this notification with quality of service QoS2. More details about this scenario are included in 9.3.1, "Push notification to selected mobile applications" on page 229.

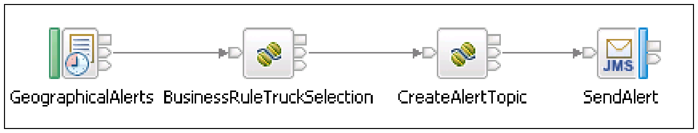
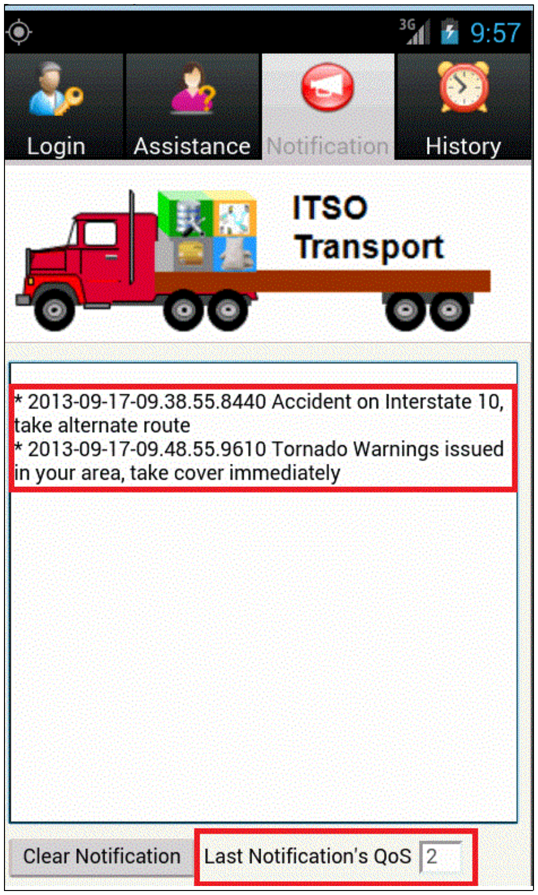


Figure 9-20 A push notification created and published to selected trucks

Figure 9-21 shows the mobile application on the CASJC01 truck, and the notifications that it received from the back-end system with QoS2.



* 2013-09-17-09.38.55.8440 Accident on Interstate 10, take alternate route
* 2013-09-17-09.48.55.9610 Tornado Warnings issued in your area, take cover immediately

Last Notification's QoS 2

Figure 9-21 Push notification messages for the CASJC01 truck with QoS2

9.5.3 Push notification broadcast to all trucks

ITSO Transport Corporation has a requirement that their back-end office team must be able to send important notifications to truck staff who are currently connected through the mobile application, or the staff who get connected when their delivery is scheduled. More details about this scenario are included in 9.3.2, “Push notification broadcast to all mobile applications” on page 232.

If the back-end office team sends a message to be broadcast now, because the CASJC01 and CASJC02 trucks are connected through the mobile application, they will receive these notifications. When driver03 connects the mobile application on the CASJC03 truck to IBM MessageSight, the mobile application will receive the previously broadcast notification, because the original messages were published with a retained property from the IBM Integration Bus message flow.

Figure 9-22 shows the part of the message flow in the IBM Integration Bus application that accepts important messages to be broadcast to all of the trucks in the ITSO Transport Corporation fleet.

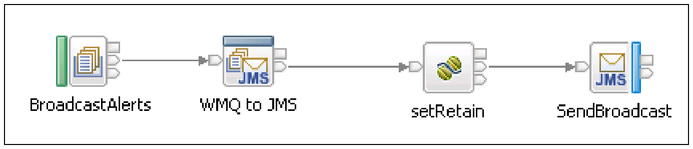


Figure 9-22 Push notification accepted from the back-end office team and sent to all of the trucks

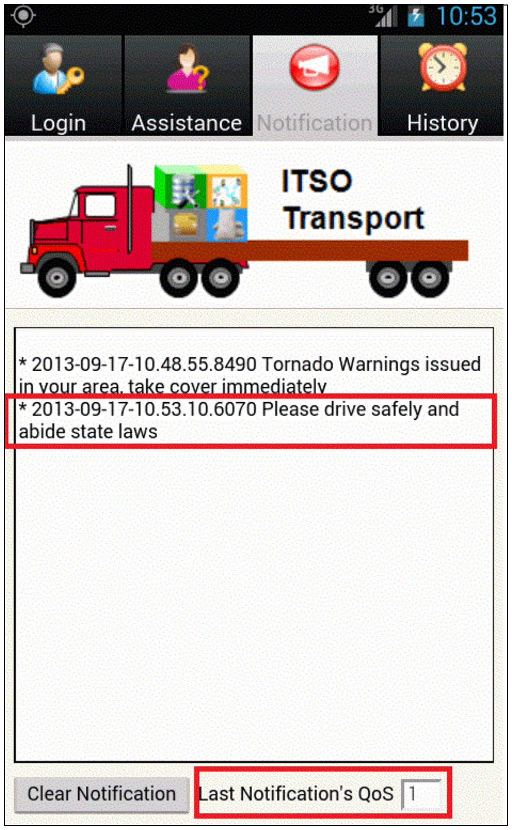
To run this scenario, simulate the back-end office team by directly putting the message in the ITSO.BROADCAST.IN queue on the ITSOQM2 queue manager. Example 9-55 shows a message to be broadcast from the ITSO Transport back-end team to all of the trucks. The message is for illustration purposes, and is put in the queue using the **amqsput** sample WebSphere MQ utility.

Example 9-55 Push notification to be broadcast to all trucks

```
$ ./amqsput ITSO.BROADCAST.IN ITSOQM2
Sample AMQSPUT0 start
target queue is ITSO.BROADCAST.IN
Please drive safely and abide state laws

Sample AMQSPUT0 end
```

Note: This message is picked up from the queue by the message flow, and a JMS retain flag is set to make this notification a retained publication.

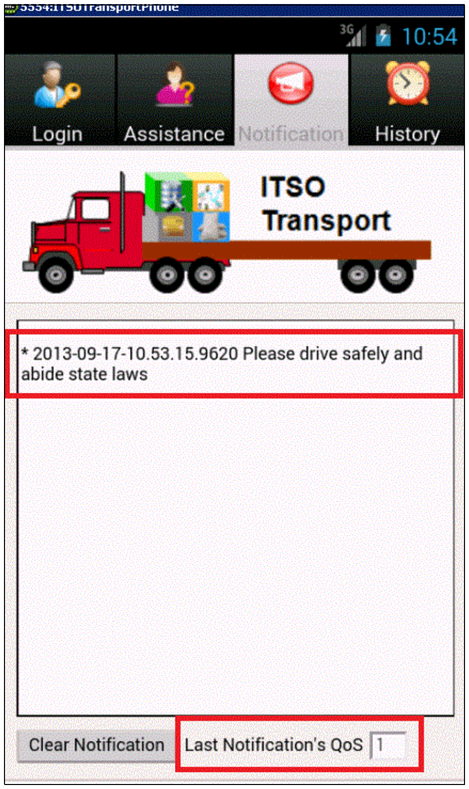
After the notification is sent, the CASJC01 and CASJC02 trucks receive this broadcast message. Figure 9-23 shows the notification received on the CASJC01 truck.



* 2013-09-17-10.48.55.8490 Tornado Warnings issued in your area, take cover immediately
* 2013-09-17-10.53.10.6070 Please drive safely and abide state laws

Figure 9-23 Broadcast notification received on the CASJC01 truck with QoS1

Figure 9-24 shows the notification received on the CASJC02 truck.



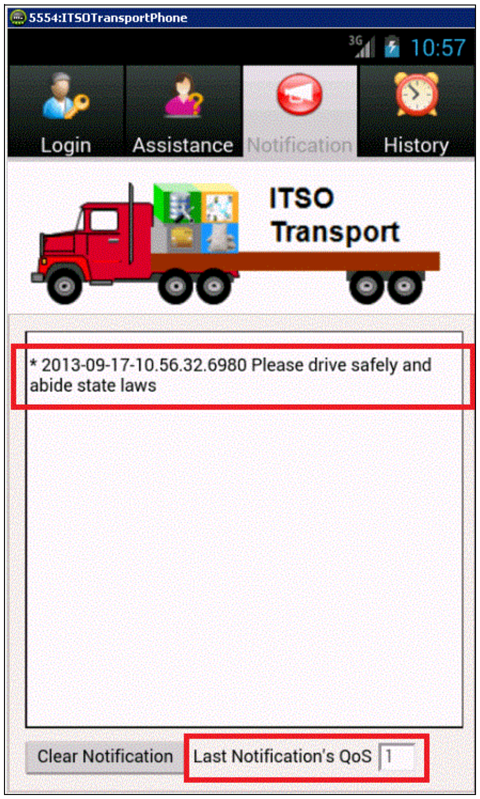
* 2013-09-17-10.53.15.9620 Please drive safely and abide state laws

Last Notification's QoS 1

Figure 9-24   Broadcast notification received on the CASJC02 truck with QoS2

Figure 9-24 and Figure 9-23 on page 277 show that the broadcast notification with QoS1 was received at the same time, because the truck drivers on the CASJC01 and CASJC02 trucks were connected at that time.

After some time, the driver03 truck driver connects the mobile application on the CASJC03 truck to IBM MessageSight by logging in to the application. CASJC03 received this broadcast because this message was published with a retain flag. As soon as the mobile application from CASJC03 got connected, it received the broadcast message that was published earlier. Figure 9-25 shows the notification received on CASJC03, but the receive time stamp is different from the receive time stamp of CASJC01 and CASJC02.



* 2013-09-17-10.56.32.6980 Please drive safely and abide state laws

1

Figure 9-25 Broadcast notification received on CASJC03 with QoS1

10

Scenario 4: Stand-alone server applications

This chapter describes the development of two sample stand-alone server applications that respond to requests published by the mobile application described in Chapter 6, “Scenarios overview” on page 175. These server applications demonstrate direct communications between mobile applications and a back-end server application, where both types of applications interface to the IBM MessageSight appliance.

The two server applications have exactly the same functionality:

- They respond to requests issued by the mobile application.
- They both use asynchronous reception of request messages.

The two server applications are both written in Java, but use different types of messaging application programming interfaces (APIs):

- One application is implemented using the Java API for MQTT. This set of Java classes is distributed with the standard WebSphere MQ distribution.
- The second application is implemented using the Java Message Service (JMS) classes implemented for the IBM MessageSight appliance.

10.1 A stand-alone server application implemented with MQTT Java

This section explains the development of a stand-alone server application that is implemented using the MQTT Java classes. The example illustrates how applications can communicate directly with the IBM MessageSight appliance without using any intermediate middleware, such as WebSphere MQ.

This can be necessary in situations where there is a high messaging load, or relatively low latency is required. It also alleviates the need to implement, administer, manage, and maintain intermediate middleware components.

The MQTT Java classes are provided in the MQTT software development kit (SDK) with WebSphere MQ. Note that the classes used in this sample server application are the org.eclipse.paho.client.mqttv3 Java classes, which are also provided in a WebSphere MQ distribution.

This sample server application uses a properties file for configuration information. The server application demonstrates an alternative approach, in place of the use of WebSphere Message Broker or IBM Integration Bus, to processing requests from mobile applications.

This server application receives requests from, and sends responses to, the mobile application described in Chapter 6, “Scenarios overview” on page 175.

The sample server application carries out the following steps:

1. Parse the command-line interface (CLI) arguments and process the properties file.
2. Create an MQTT client. This demonstrates how additional security can be used through the use of a user name and password. The application also shows how to set a *Last Will and Testament*, which is an MQTT feature that generates an alert if there is an unexpected break in the connection between the MQTT server (appliance) and the MQTT client (application).
3. This server application uses callbacks for the following functions:
   - Arrival of a request message
   - Indication of complete delivery when a response message has been published to the required response Topic
   - When a connection to the MQTT server (appliance) is lost
4. Create a connection to the MQTT server (the IBM MessageSight appliance).

5. Create a subscription for the required request Topic. This request Topic uses a wildcard, because requests can be received from multiple different requesting mobile applications.
6. When a request message is received, identify the source of the message, the message type and send the required response to the requesting application.

The application continues running perpetually, and must be stopped with the Ctrl+C keys. This is for demonstration purposes only. A conventional, production-quality application needs to stop as dictated by the business usage and associated functional requirements.

Example 10-1 shows the configuration file (properties file) for this sample server application.

Example 10-1 Sample configuration file

```
#
# IBM MessageSight sample MQTT server application
# Configuration file
#
ima.servers=192.168.121.135
ima.port=16102
client.id=ITSOSVR01
req.topic=itso/driver/assistance/#
resp.topic=itso/driver/response/
#
# Last Will and Testament
lwt.topic=ITSO/LWT/SVR/ITSOSVR01
lwt.retain=true
#
# The following time is in milliseconds
sleep.time=60
#
clean.session=false
qos=2
#
# Username and Password are used for additional security
#
#user.name=testuser
#user.passwd=testpassword
```

Note a few of the items in the configuration file:

- The MQTT client identifier used by this server application. As specified by the MQTT V3.1 specification, the MQTT Client Identifier has these attributes:
  - Must be between 1 and 23 characters
  - Must uniquely identify the client to the MQTT server
  - Must be unique across all clients connecting to the same MQTT server
- The use of a wildcard, number sign (#), in the request topic.
- The topic to be used for the Last Will and Testament for this application.
- The MQTT messaging quality of service (QoS) that is used for both request and response messages.

Example 10-2 shows the required Java imports for this sample server application.

Example 10-2 The server application Java imports

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;
import java.util.Date;
import java.text.SimpleDateFormat;

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttCallback
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.MqttTopic;
import org.eclipse.paho.client.mqttv3.persist.MqttDefaultFilePersistence;
```

Example 10-3 depicts the creation of an MQTT client. It also shows the optional use of additional security credentials, such as the `username` and `password`, and the configuration of the “Last Will and Testament” feature. The `Server` and `Port`, required identifiers and necessary topic names are passed into the application using the properties file.

In this code snippet, the application subscribes to the request message topic using a wildcard. Note the configuration of the asynchronous callbacks.

Example 10-3 Initial MQTT setup of the server application

```
try {
client = new MqttClient(URL, clientID, null);
CallBack callback = new CallBack(clientID);
client.setCallback(callback);
```

```
MqttConnectOptions conOptions = new MqttConnectOptions();

// NOTE: cleansession is being set to 'true'
// We should use a parameter to control this
conOptions.setCleanSession(true);
// Other connectOptions that can be set are:
// username and password
// Last Will and Testament (LWT)
if (userPass != null)
{
   conOptions.setUserName(userName);
   conOptions.setPassword(userPass.toCharArray());
}
if (lwtTopicName != null) {
   MqttTopic lwtTopic = client.getTopic(lwtTopicName);
   String lwtMsg = "MY LWT";
   conOptions.setWill(lwtTopic, lwtMsg.getBytes(), LWTQOS, lwtRetain);
}

// Connect to the server
client.connect(conOptions);

// Subscribe to Request Topic
System.out.println("Subscribing to topic \"" + reqTopicName
    + "\" for client instance \"" + client.getClientId()
    + "\" using QoS " + QoS + ". Clean session is true");
client.subscribe(reqTopicName, QoS);
```

This server application uses `cleansession=true`, because the application does not want the MQTT server to retain any information about it when the application is not connected. When the server application connects, its session is “clean”, meaning that the MQTT server has not retained any information from any previous sessions with this client.

This can occur if publications that matched the application’s subscriptions were received by the MQTT server while the application was no longer connected to the MQTT server. If `cleansession=false`, the MQTT server retains the published messages awaiting the application to reconnect to the MQTT server.

The Last Will and Testament message is published using the topic specified by the application when a connection between the application and MQTT server is unexpectedly broken (an unplanned loss of connection). For example, a monitoring application can subscribe to these Last Will and Testament messages and report on, or take any other required actions for, the detection of broken connections.

Example 10-4 depicts the implementation of the asynchronous callback methods. These are implemented in a separate Java class.

Example 10-4 Callback methods

```
/*
 *  Callback.java
 *
 *  This file contains the callback classes for asynchronous
 *  behaviour of the MQTT application.
 *
 *  The required methods are:
 * - connectionLost
 * - messageArrived
 * - deliveryComplete
 *
 */
package com.ibm.itso;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

public class CallBack implements MqttCallback
{
   public int deliveryConf = 0;
   public int respRcvd = 0;

  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
}

@Override
public void connectionLost(Throwable cause)
{
   System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
   cause.printStackTrace();
}

@Override
public void deliveryComplete(IMqttDeliveryToken arg0)
{
   try
   {
      //System.out.println("Delivery token \"" + token.hashCode()
      //    + "\" received by instance \"" + instanceData + "\"");
      deliveryConf++;
      } catch (Exception e) {
         e.printStackTrace();
      }
}

/*
 * (non-Javadoc)
 * @see
org.eclipse.paho.client.mqttv3.MqttCallback#messageArrived(java.lang.String,
org.eclipse.paho.client.mqttv3.MqttMessage)
 *
 * This method receives the request, calculates the Response Topic, and calls
 * the publishResp method to process and send the response.
 *
 * For the ITSO Transport sample, the expected request topic structure is:
 * /itso/driver/assistance/ACMalfunction/<TruckNumber>
 * /itso/driver/assistance/FlatTire/<TruckNumber>
 * /itso/driver/assistance/Accident/<TruckNumber>
 *
 * Request payload is
"DriverID+TruckNumber+Longitude+Latitude+hh:mm:ss+Date+<ACMalfunction|FlatTire|
Accident>+Text"
 *
 * Request Topic Elements:
 * Element[3] - Incident tyoe
 * Element[4] - Truck Number
 *
 * Payload Elements:
 * Element[0] - Driver ID
 * Element[1] - Truck ID
 * Element[6] - Incident Type
 *
 */
@Override
public void messageArrived(String topicName, MqttMessage msg) throws Exception
{
   try
   {
      if (topicName.contains("/"))
      {
         // Split it.
         String[] topElements = topicName.split("/");
         String reqPayload = msg.toString();
         String myRspTopic = ITSOMQTTServer.rspTopicName + topElements[4];
         // Add the Truck Number to the Response Topic
```

```
            System.out.println("MsgArr: Req Topic: "+topicName+" payload
            "+reqPayload+" Resp Topic "+myRspTopic);
            ITSOMQTTServer.publishResp(myRspTopic, reqPayload);
         } else {
               throw new IllegalArgumentException("Received topic name " +
               topicName + " does not contain /");
         }
         respRcvd++;
      } catch (Exception e) {
            e.printStackTrace();
        }
   }
}
```

The messageArrived method is where the request message is received and parsed, and the request type and origins of the request are identified. The required response Topic is constructed, and the response processing method is started.

In this sample, no special processing is done in the connectionLost method to reestablish the connection to the MQTT server. For an example of reestablishing this connection, see 3.3.4, “A sample JMS publisher” on page 63, and 3.3.5, “A sample JMS subscriber” on page 71.

The deliveryComplete method keeps a running total of the number of responses that have been published by this server application.

Example 10-5 shows the response message processing, and the publication of an appropriate response message.

Example 10-5 Response message processing

```
//  Send a response to the request
//
public static void publishResp(String rspTopic, String inMsg)
{
   String respPayload = " ";
   Date now;
   SimpleDateFormat dtFormat = new SimpleDateFormat("E yyyy.MM.dd 'at' HH:mm:ss
   zzz");
   now = new Date();
   String dtOut = dtFormat.format(now);

   MqttTopic myTopic = client.getTopic(rspTopic);
   MqttMessage rspMsg = new MqttMessage();

   // If the request payload was "QUIT"
   // then terminate this server app.
```

```
    if (inMsg.compareToIgnoreCase("quit") == 0)
    {
       quitSvr = true;
       return;
    }

    try {
       // Parse the request payload
       // Payload Elements:
       // Element[0] - Driver ID
       // Element[1] - Truck ID
       // Element[6] - Incident Type
       if (inMsg.contains("+"))
       {
          String[] msgElements = inMsg.split("\\+");
          if (msgElements[6].compareToIgnoreCase("ACMalfunction") == 0)
          {
             respPayload = msgElements[0]+", AC engineer dispatched to your
             location. "+dtOut+" Dispatcher";
          }
          if (msgElements[6].compareToIgnoreCase("FlatTire") == 0)
          {
            respPayload = msgElements[0]+", Vehicle mechanic dispatched to your
           location. "+dtOut+" Dispatcher";
          }
          if (msgElements[6].compareToIgnoreCase("Accident") == 0)
          {
             respPayload = msgElements[0]+", Emergency staff notified and
             dispatched to your location. "+dtOut+" Dispatcher";
          }
       } else {
             throw new IllegalArgumentException("Received request payload " +
             inMsg + " does not contain a correct separator \'+\'");
          }

       System.out.println("*** publishResp RespTopic: "+rspTopic+" payload:
       "+respPayload);
       rspMsg.setPayload(respPayload.getBytes());
       rspMsg.setQos(QoS);
       myTopic.publish(rspMsg);
       // The following code is just for the convenience of sample purposes
       // Find a more efficient way to do this processing in a real app
       myTopic = null;
       rspMsg = null;
    } catch (MqttException e) {
          e.printStackTrace();
       }
}
```

This publishResp method identifies the request type, and constructs a suitable response payload. Three different request message types are used in this sample:

- An accident involving the truck.
- A flat tire on the truck.
- An air conditioning malfunction in the truck's cargo bay. This might lead to the loss of perishable goods that require an air-conditioned environment.

A temporary response topic and MQTT message are created, used, and deleted each time that a response must be sent. We realize that this is not an efficient or elegant way of implementing this, and it should not be used in real applications. It has been used for sample purposes only.

10.2 A stand-alone server application implemented using IBM MessageSight JMS classes

This section explains the development of a stand-alone server application that is implemented using the IBM MessageSight JMS classes. In a similar manner to the prior example, this sample server application illustrates how applications can communicate directly with the IBM MessageSight appliance without using any intermediate middleware, such as WebSphere MQ.

The required JMS classes can be obtained as previously explained in section 3.3.2 (IBM MessageSight JMS Client Pack).

This sample application, implemented using JMS, demonstrates that messages can be exchanged with applications using MQTT APIs. The JMS application uses JMS BytesMessage objects as the message format for message exchange with MQTT applications.

This JMS application also uses a MessageListener to asynchronously accept request messages. Because this application exchanges messages with an MQTT application, it uses the JMS publish/subscribe paradigm.

This server application receives requests from, and sends responses to, the mobile application described in Chapter 5 (MQTT with mobile applications).

The sample server application carries out the following steps:

1. Parse the command-line arguments and process the properties file.
2. Create a connection to the JMS server, the IBM MessageSight appliance.

3. Create a subscription for the required request Topic. This request Topic uses a wildcard, because requests can be received from multiple different requesting mobile applications.
4. Establish a MessageListener for the MessageConsumer for the request Topic.
5. For demonstration purposes *only,* the server application creates three MessageProducers for responding to the mobile applications. The identities of these three message producers are configurable using the properties file.
6. Establishes an ExceptionListener on the connection. If a break in the connection to the JMS server is detected, the application will attempt to re-establish a connection to the JMS server.
7. All request and response processing is done in the MessageListener.
8. The server application is capable of processing both JMS TextMessage and BytesMessage messages.

Example 10-6 shows the configuration file (properties file) for this sample JMS server application.

Example 10-6 The JMS server application configuration file

```
#
# IBM MessageSight sample JMS Server app
# Configuration file
#
ima.servers=192.168.121.135
ima.port=16102
client.id=ITSOJMSSVR01
req.topic=itso/driver/assistance/#
resp.topic=itso/driver/response/
#
# JMS Delivery Mode - true=persistent false=non-persistent
delivery.mode=true
#
# The following time is in millisecs
sleep.time=100
#
# Username and Password are used for additional security
#
#user.name=testuser
#user.passwd=testpassword
#
# Up to 3 possible Truck Identifiers
#
dest1=ITSOTRUCK01
dest2=ITSOTRUCK02
dest3=ITSOTRUCK03
```

Example 10-7 shows the required JMS imports for this sample JMS server application.

Example 10-7 JMS imports

```
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.MessageConsumer;
import javax.jms.Topic;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.BytesMessage;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ExceptionListener;

import java.io.*;                    // Standard Java imports
import java.text.SimpleDateFormat;
import java.util.*;

import com.ibm.ima.jms.ImaJmsException;
import com.ibm.ima.jms.ImaJmsFactory;
import com.ibm.ima.jms.ImaJmsObject;
import com.ibm.ima.jms.ImaProperties;
```

Example 10-8 depicts the creation of a JMS connection, session, and the required Topics. A single MessageConsumer and three MessageProducers are also created. It also shows optional use of additional security credentials, such as the username and password. The Server and Port, required identifiers, and necessary topic names are passed into the application using the properties file.

In this code snippet, the application subscribes to the request message topic using a wildcard. Note the configuration of the asynchronous MessageListener for the MessageConsumer. Also note the creation of an ExceptionListener for the connection.

Example 10-8 JMS object creation

```
Topic respDest1 = null;
Topic respDest2 = null;
Topic respDest3 = null;

// Parse the arguments
parseArgs(args);
```

```
/*
* Connect to server
*/
boolean connected = doConnect();

if (!connected)
throw new RuntimeException("Failed to connect!");

// Create a session.
sess = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
Topic reqTopic = getTopic(reqTopicName);
if (duraSubsName != null)
{
   // Create a Durable Subscriber
   cons = sess.createDurableSubscriber(reqTopic, duraSubsName);
   System.out.println("Created a Durable Subscription named "+duraSubsName);
} else {
   // Create a standard message consumer
   cons = sess.createConsumer(reqTopic);
   }

// Create some sample responders - up to 3
respDest1 = getTopic(rspTopicPref+dest1Id);
respDest2 = getTopic(rspTopicPref+dest2Id);
respDest3 = getTopic(rspTopicPref+dest3Id);
responder1 = sess.createProducer(respDest1);
responder2 = sess.createProducer(respDest2);
responder3 = sess.createProducer(respDest3);

// Create Consumer objects
// Using a wildcard on the request topic for subscription

try
{
   rqstListener msgListener = new rqstListener();
   cons.setMessageListener(msgListener);
} catch (JMSException je) {
   System.err.println("Error setting MsgListener" + je.getMessage());
   System.err.println(je);
}

// Do not forget to start the connection
conn.start();
//System.out.println("Consumer Session started OK! Using Topic: "+consDest);

// Create the keyboard reader, so that we can get user input
BufferedReader in = new BufferedReader( new InputStreamReader(System.in ));

char answer = '\0';
```

```
System.out.println("*** To end program, enter Q or q, then <return>");
while (!((answer == 'q') || (answer == 'Q')))
{
   Thread.sleep(SLEEP_TIME*1000);// Has to be in milliseconds
   try {
      answer = (char) in.read();
   } catch (IOException e) {
      System.err.println("I/O exception: " + e.toString());
       }
}

// We must remember to call the close() method on all the objects we
// have used, to ensure a clean tidy-up of all resources held

cons.close();// Close the consumer
responder1.close();
responder2.close();
responder3.close();

sess.close();
conn.close();
//respSession.close();
//respConn.close();
System.out.println("*** IMAJMSServer terminated!");
```

Example 10-9 shows the creation of the required Java Naming and Directory Interface (JNDI)-administered objects (the ConnectionFactory and the required request and response Topics).

Example 10-9 JNDI-administered object creation

```
/**
 * Get connection factory administered object.
 *
 * @return ConnectionFactory object.
 *
 * Note: In production applications, this method would retrieve a
 *   ConnectionFactory object from a JNDI repository.
 */
public static ConnectionFactory getCF() throws JMSException {
ConnectionFactory cf = ImaJmsFactory.createConnectionFactory();
/*
 * NOTE: For high availability configurations, the serverList
 * must contain the list of IBM MessageSight server host names or IP addresses
 * so that the clientcan attempt to connect to alternate hosts if connection
 * (or reconnection)to the primary fails.
 */
((ImaProperties) cf).put("Server", serverList);
((ImaProperties) cf).put("Port", serverPort);
if (disableACK)
{
   ((ImaProperties) cf).put("DisableACK", true);
   System.err.println("ACKs disabled in ConnectionFactory!");
}

/*
 * After setting properties on an administered object, it is a best practice to
 * run the validate() method to assure all specified properties are recognized
 * by the IBM MessageSight JMS client.
 */
ImaJmsException errors[] = ((ImaProperties) cf).validate(true);
if (errors != null)
   throw new RuntimeException("Invalid properties provided for the connection
   factory.");
   return cf;
}

/**
 * Create topic administered object.
 *
 * @return Topic object.
 *
 * Note: In production applications, this method would retrieve a
 *   Topic object from a JNDI repository.
 */
public static Topic getTopic(String topicName) throws JMSException {
   //System.out.println("getTopic: "+topicName);
   Topic dest = ImaJmsFactory.createTopic(topicName);
   return dest;
}
```

Example 10-10 shows the initial JMS connection establishment. Note the creation of an ExceptionListener in the connect method. Connection re-establishment is attempted here after a connection with the JMS server has been broken.

Example 10-10 Connection and re-connection methods

```
/**
 * Establish a connection to the server.  Connection attempts are retried until
 * successful or until the specified timeout for retries is exceeded.
 */
public static boolean doConnect()
{
   int connattempts = 1;
   boolean connected = false;
   long starttime = System.currentTimeMillis();
```

```
      long currenttime = starttime;

      /*
       * Try for up to connectTimeout milliseconds to connect.
       */
      while (!connected && ((currenttime - starttime) < connectTimeout))
      {
         try { Thread.sleep(5000); } catch (InterruptedException iex) {}
         if (debug)
         System.out.println("Attempting to connect to server (attempt " +
                            connattempts + ").");
         connected = connect();
         connattempts++;
         currenttime = System.currentTimeMillis();
      }
      return connected;
   }

   /**
    * Connect to the server
    *
    * @return Success (true) or failure (false) of the attempt to connect.
    */
   public static boolean connect()
   {
      /*
       * Create connection factory and connection
       */
      try
      {
         fact = getCF();
         // If username AND user password have been defined in the
         // config file, then set these as additional connection security.
         if (userPass != null)
            conn = fact.createConnection(userName, userPass);
         else
            conn = fact.createConnection();
         // Set the Client Identifier for the connection
         conn.setClientID(clientID);
         // Set an Exception Listener
         // for the connection.
         conn.setExceptionListener(new ExceptionListener()
         {
            public void onException(JMSException jmse)
            {
               System.err.println("An exception received through the
                                  ExceptionListener");
               if (isConnClosed())
               {
                  // Attempt to reconnect
                  System.err.println("*** Connection is closed.  Attempting to
                                     reconnect and continue.");
                  boolean reconnected = doConnect();

                  if (reconnected)
                  {
                     // Simply continue
                     System.out.println("*** Reconnected OK!  Continuing!");;
                   } else {
                         System.err.println("*** Failed to reconnect. Exiting!");
                         System.exit(0);
                          }
               } else {
                     System.err.println("*** Not a closed connection. Exiting!");
                     jmse.printStackTrace();
                     System.exit(0);
                      }
             }
         });

         /*
          * Check that we are using IBM MessageSight JMS administered objects
          * before calling * provider specific check on property values.
          */
         if (fact instanceof ImaProperties)
         {
            /*
             * For high availability, the connection factory stores the list of
             * IBM MessageSight server
             * host names or IP addresses.  When connected, the connection object
            * contains only the host name or IP
             * to which the connection is established.
             */
            System.out.println("Connected to " +
            ((ImaProperties)conn).getString("Server"));
         }
         /*
          * Connection has succeeded.  Return true.
          */
         return true;
      } catch (JMSException jmse) {
           System.err.println(jmse.getMessage());
            /*
             * Connection has failed.  Return false.
             */
            return false;
            }
   }
```

```
/**
 * Check to see if connection is closed.
 *
 * When the IBM MessageSight JMS client detects that the connection to the
 * server has been * broken, it marks the connection object closed.
 *
 * @return True if connection is closed or if the connection object is null,
 * false otherwise.
*/
public static boolean isConnClosed()
{
   if (conn != null)
   {
      /*
       * Check the IBM MessageSight JMS isClosed connection property
       * to determine whether the connection state is closed.
       * This mechanism for checking whether the connection is closed
       * is a provider-specific feature of the IBM MessageSight
       * JMS client.  So check first that the IBM MessageSight JMS
       * client is being used.
       */
      if (conn instanceof ImaProperties)
      {
         return ((ImaProperties) conn).getBoolean("isClosed", false);
       } else {
            // It cannot be determined whether the connection is closed so
            // return false.
            return false;
            }
     } else {
         return true;
         }
}
```

Example 10-11 shows the request and response processing in the MessageListener. Note that both JMS TextMessage and BytesMessage objects are handled. BytesMessage messages are used when message exchanges take place between JMS applications and MQTT applications.

Example 10-11   MessageListener request and response processing

```
public static class rqstListener implements MessageListener
{
   public void onMessage(Message message)
   {
      String reqTopic = "";
      String respTopic = "";
      String reqMsgId = "";
      String reqCorrelId = "";
      String respTruck = "";// The Truck Id that originated the request message
      String reqPayload = "";
      String respPayload = " ";
      String driverID = "";
      String incidentType = "";
      int plLgt = 0;// Payload length (size)
      Message respMessage = null;

      try {
         reqTopic = ((Topic)message.getJMSDestination()).getTopicName();
         // Get the request message's            Message ID
         reqMsgId = message.getJMSMessageID();
         // Get the request message's Correlation ID
         reqCorrelId = message.getJMSCorrelationID();
      } catch (JMSException je) {
            je.printStackTrace();
         }

      if (reqTopic.contains("/")) {
           // Split it.
         String[] topElements = reqTopic.split("/");
         respTruck = topElements[4];   // Get the Truck Number
         // Get the message payload
         try {
               if (message instanceof TextMessage)
               {
                   reqPayload = ((TextMessage) message).getText();
               } else if (message instanceof BytesMessage)
                        {
                            byte[] msgBytes = new byte[128];;
                            plLgt = ((BytesMessage)
                                      message).readBytes(msgBytes);
                            reqPayload = new String(msgBytes);
                            reqPayload = reqPayload.trim();  // Trim whitespace

                            //System.out.println("--- Length: "+plLgt+"
                              Payload: "+reqPayload);
                        }
         } catch (JMSException je)
         {
             je.printStackTrace();
         }
         if (reqPayload.contains("+")) {
               String[] msgElements = reqPayload.split("\\+");
               incidentType = msgElements[6];
               driverID = msgElements[0];
           } else {
```

```
                throw new IllegalArgumentException("Received request
                  payload " + reqPayload + " does not contain a correct
                  separator \'+\'");
                }
      Date now = new Date();
      SimpleDateFormat dtFormat = new SimpleDateFormat("E yyyy.MM.dd 'at'
       HH:mm:ss zzz");
      String dtOut = dtFormat.format(now);

      if (incidentType.compareToIgnoreCase("ACMalfunction") == 0) {
           respPayload = driverID+", AC engineer dispatched to your
              location. "+dtOut+" Dispatcher";
      }
     if (incidentType.compareToIgnoreCase("FlatTire") == 0) {
           respPayload = driverID+", Vehicle mechanic dispatched to your
              location. "+dtOut+" Dispatcher";
     }
    if (incidentType.compareToIgnoreCase("Accident") == 0) {
          respPayload = driverID+", Emergency staff notified and dispatched
             to your location. "+dtOut+" Dispatcher";
    }
   } else {
         throw new IllegalArgumentException("Received topic name " +
            reqTopic + " does not contain /");
         }

   System.out.println("MsgArr: Req Topic: "+reqTopic+" payload
      "+reqPayload+" from TruckID "+respTruck);
   //System.out.println("*** Message ID: " + reqMsgId+" Correl ID: " +
     reqCorrelId);

   if (message instanceof TextMessage)
   {
         try {
                     respMessage = sess.createTextMessage();
                     // Set the response payload
                     ((TextMessage) respMessage).setText(respPayload);
         } catch (JMSException je) {
               je.printStackTrace();
         }
   } else if (message instanceof BytesMessage) {
         // Probably a message from a MQTT client
         try {
               respMessage = sess.createBytesMessage();
               // Set the response payload
               ((BytesMessage)
                 respMessage).writeBytes(respPayload.getBytes());
         } catch (JMSException je) {
               je.printStackTrace();
```

```
                    }
            }
            try {
                  // Set the required response delivery mode
                  if (respDelMode)
                  {
                  respMessage.setJMSDeliveryMode(DeliveryMode.PERSISTENT);
                  } else {
                        respMessage.setJMSDeliveryMode(DeliveryMode.NON_PERSISTENT);
                        }
                  // Set the response Correlation ID = request Message ID
                  respMessage.setJMSCorrelationID(reqMsgId);

                  // We decide on which Topic to send the response
                  // based on the ClientID (i.e TruckId) in the
                  // incoming request message.

                  if (respTruck.equalsIgnoreCase(dest1Id))
                  responder1.send(respMessage);
                  else if (respTruck.equalsIgnoreCase(dest2Id))
                  responder2.send(respMessage);
                  else if (respTruck.equalsIgnoreCase(dest3Id))
                  responder3.send(respMessage);
            } catch (JMSException je) {
                  System.err.println("Error processing the Response" +
                     je.getMessage());
                  System.err.println(je);
                  }
             }
}
```

10.3 Conclusion

The intent of this chapter was to demonstrate how mobile applications can communicate directly with back-end server applications using IBM MessageSight. The two types of applications, mobile and back-end, can use any mix of MQTT and JMS (MessageSight classes) messaging systems.

Remember: JMS is an API, not a messaging protocol.

The mobile applications can use any suitable implementation language, such as Java, JavaScript, Objective-C, and so on, if they can access the required messaging APIs. Similarly, the back-end applications can use any implementation language that provides access to the MQTT or JMS APIs.

Note that both MQTT and JMS messaging systems can use different sets of options when interacting with the IBM MessageSight server. The application developers must become familiar with the similarities and differences with the options of the two messaging APIs. MQTT is only for publish/subscribe, but JMS provides both publish/subscribe and queue-based messaging. The IBM MessageSight JMS classes implement both types of JMS messaging.

The back-end applications have a choice of using synchronous or asynchronous message reception, by using either the MQTT or JMS APIs.

For additional information, guidance, and examples, see the *Mobile development* topic on the IBM developerWorks website:

`http://www.ibm.com/developerworks/mobile/learn.html`

The following websites provide more information about JMS-related topics:

- IBM MessageSight JMS client API reference:

  `https://pic.dhe.ibm.com/infocenter/ism/v1r0m0/topic/com.ibm.ism.doc/Developing/jmsapireference.html`

- Package javax.jms:

  `http://download.oracle.com/javaee/6/api/javax/jms/package-summary.html`

The following website provides information about programming MQTT for C, Java, and JavaScript:

`http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mm.tc.doc/tc00200_.htm`

A

MQTT protocol

The MQTT protocol is a lightweight network protocol used for publish/subscribe messaging, optimized for wireless remote devices that might have limited bandwidth and intermittent connectivity. This appendix describes the following concepts about the MQTT protocol:

- Quality of service (QoS) levels and flows
- QoS determination
- QoS effect on performance
- Client ID
- Durable and non-durable subscribers
- MQTT persistence
- MQTT header
- Keep alive
- Retry message delivery
- Last Will and Testament
- Retained flag on messages
- TCP/IP stack

Quality of service levels and flow

QoS levels determine how each MQTT message is delivered, and must be specified for every message sent through MQTT.

A useful comparison can be made with letters sent through a postal service. An informal letter to a friend can be dropped into a mailbox, and the sender might never think about it again. The sender expects that the letter will get to its destination, but there are no significant consequences if it does not arrive.

In contrast, a letter to which the recipient must respond is more important, so the sender might choose a higher level of delivery service that provides proof that the letter was received at its destination. In each situation, the choices made by the sender of the letter are comparable to choosing QoS levels in MQTT.

It is important to choose the appropriate QoS value for each message, because this value determines how the client and the server communicate to deliver the message.

QoS 0: At most once delivery

It is termed as *At most once message delivery* because messages are delivered according to the best effort of the underlying network. A response is not expected, and no retry semantics are defined in the protocol. The message arrives at the MQTT server either once or not at all.

This is the lowest level of QoS. The MQTT client or the server attempts to send the message without waiting for any kind of receipt. There are no steps taken to ensure message delivery, other than the features provided by the TCP/IP layer. Also, there is no retry attempted by the MQTT layer if there is a failure to deliver the message. Therefore, if the client is sending a message, it can arrive at the MQTT server once or never.

A QoS 0 message can get lost if the client unexpectedly disconnects or if the server fails. From a performance perspective, this adds value, because it is the fastest way to send a message using MQTT.

The MQTT command message used is **PUBLISH**. No other command messages flow for the QoS 0 messages. Table A-1 shows the QoS level 0 protocol flow.

Table A-1 QoS level 0 protocol flow

Client	Message and direction	MQTT server
QoS = 0	PUBLISH ------------>	Action: Publish message to subscribers

QoS 1: At least once delivery

With QoS set to 1, the message is delivered *at least once*. The MQTT client or the server attempts to deliver the message at least once, but there can be a duplicate message. The receipt of a message by the MQTT server is acknowledged by a PUBACK message.

If there is an identified failure of either the communications link or the sending device, or the acknowledgment message is not received after a specified period of time, the sender resends the message with the DUP bit set in the message header. The message arrives at the MQTT server at least once. Both SUBSCRIBE and UNSUBSCRIBE messages use QoS level 1.

If the client does not receive a PUBACK message (either within a time period defined in the application, or if a failure is detected and the communications session is restarted), the client resends the PUBLISH message with the DUP flag set. When it receives a duplicate message from the client, the MQTT server republishes the message to the subscribers, and sends another PUBACK message.

At the client side, the implementations of the MQTT protocol also provides an additional feature known as *MQTT persistence*. The MQTT persistence layer is not described in the specification of MQTT, but is normally available with MQTT client implementations. When a QoS 1 message is published to the server, the client needs to wait for the acknowledgement to arrive.

There can be a program termination or a crash at the client device. When the client is started again, it will need to resume from the point it left before the crash. Therefore, the message is stored in a persistence layer, such as a disk, and retrieved soon after the reconnection to the MQTT server.

The MQTT command messages used are **PUBLISH** and **PUBACK**. When the publish happens, the message will be logged to the MQTT persistence layer, and removed when a PUBACK message is received. A message with QoS level 1 has a Message ID in the message header.

Table A-2 shows the QoS level 1 protocol flow.

Table A-2 QoS level 1 protocol flow

Client	**Message and direction**	**MQTT server**

QoS =1
DUP = 0
Message ID = x
	PUBLISH
	------------>
		Actions:
		▸ Store message in database
		▸ Publish message to subscribers

Action: Discard message	PUBACK <-----------	N/A

QoS 2: Exactly once delivery

Exactly once is the highest level of QoS. Additional protocol flows higher than QoS level 1 ensure that duplicate messages are not delivered to the receiving application. The message is delivered *once and only once* when QoS 2 is used. The MQTT client or the server will ensure that the message is sent only once.

This QoS must be used only when duplicate messages are not wanted. From a performance perspective, there is a price to be paid in terms of network traffic and processing power.

The MQTT command messages used are **`PUBLISH`**, **`PUBREC`**, **`PUBREL`**, and **`PUBCOMP`**. The message is sent in the PUBLISH flow, and the client will store that message in the MQTT persistence layer (if it is used). The message will remain locked on the server. PUBREC is sent by the server in response to PUBLISH. PUBREL will be dispatched to the server from the client in response to PUBREC.

After PUBREL is received by the MQTT server, it can dispatch the messages to the subscribers and send back PUBCOMP to the PUBREL. A message with QoS level 2 has a Message ID in the message header.

If a failure is detected, or after a defined time period, each part of the protocol flow is tried again with the DUP bit set. The additional protocol flows ensure that the message is delivered to subscribers once only.

Because QoS1 and QoS2 indicate that messages must be delivered, the MQTT server stores messages in a database. If the MQTT server has problems accessing this data, messages might be lost.

Table A-3 shows the QoS level 2 protocol flow.

Table A-3 QoS level 2 protocol flow

Client	**Message and direction**	**MQTT server**

QoS = 2
DUP = 0
Message ID = x
	PUBLISH
	----------->
		Action:
		Store message in a database

	PUBREC <-----------	Message ID = x
Message ID = x	PUBREL ----------->	Actions: ► Update database ► Publish message to subscribers
Action: Discard message	PUBCOMP <-------------	Message DI = xb

Assumptions for QoS levels 1 and 2

In any network, it is possible for devices or communication links to fail. If this happens, one end of the link might not know what is happening at the other end. These are known as *in doubt windows*. In these scenarios, assumptions have to be made about the reliability of the devices and networks involved in message delivery.MQTT assumes that the client and server are generally reliable, and that the communications channel is more likely to be unreliable.

If the client device fails, it is typically a catastrophic failure, rather than a transient one. The possibility of recovering data from the device is low. Some devices have non-volatile storage (for example, flash ROM). The provision of more persistent storage on the client device protects the most critical data from some modes of failure.

Beyond the basic failure of the communications link, the failure mode matrix becomes complex, resulting in more scenarios than the specification for MQTT can handle.

The time delay (retry interval) before resending a message that has not been acknowledged is specific to the application, and is not defined by the protocol specification.

QoS determination

When the client subscribes to the topic, it will specify a QoS at which it wants to receive messages from the server. Consider a scenario where publisher A is sending the messages to the topic at QoS 2. A subscriber can subscribe at QoS 0, and then the messages to the client will be delivered with QoS 0.

The QoS value is further used in SUBSCRIBE and UNSUBSCRIBE requests. A subscribe request from the client is a QoS 1 request, so the server will respond with SUBACK, which will ensure that the subscription has happened.

QoS effect on performance

There is a simple rule when considering the performance effect of QoS. It is: *The higher the QoS, the lower the performance*. Evaluate performance corresponding with a higher QoS. Suppose the time taken for sending a PUBLISH message is *pt*. If QoS is used, the total time taken to transfer *n* number of messages will be *npt*. Now in case of QoS 1, the PUBACK message (that is, the reply for the PUBLISH message) will flow from the server to the client.

This is a two-byte message, and can take much less time than *pt*. Therefore, call it *mt*. The time taken for transferring *n* messages will be *n(pt + mt)*. Also, for QoS 2, the PUBREC, PUBREL, and PUBCOMP messages will be flowing. Therefore, the *n* number of messages will take approximately *n(pt + 3mt)*.

So, if 10 messages need to be transferred from the client to the server, and *pt* is 1 second and *mt* is 0.4 seconds, a QoS 0 message takes 10(1) = 10 seconds. A QoS 1 message takes 10(1 + 0.4), which is 14 seconds, and a QoS 2 message takes 22 seconds.

MQTT client identifier

The MQTT protocol defines a *client identifier* (client ID) that uniquely identifies a client in a network. In simple terms, when connecting to a server, a client needs to specify a unique string that is not used currently and will not be used by any other client that will connect to the MQTT server. There are several ways of choosing a client identifier.

The following list includes a few examples:

- A Sensor installed in a particular location can use the location code as the client ID.
- A mobile device having network ability can choose the MAC address or a unique device ID as the client ID

MQTT restricts the client ID length to 23 characters, so some situations will require that the client ID be shortened. When shortening the client ID, you need to ensure that the client ID is not the same as any other client ID used in the network. To keep the identifier short and unique, introduce a reliable identifier generation mechanism.

For instance, you might create a client ID from the 48-bit device MAC address. If transmission size is not a critical issue, you might use the remaining 17 bytes to make the address easier to administer, such as including some human-readable text in the identifier.

If two clients were to have the same client identifier, one of the clients might receive a message and the other one might not. The MQTT server keeps track of the pending messages to be sent to a client based on the client ID. Therefore, if a client has been using QoS 1 or QoS 2, and subscribed to any topic and disconnected from the server, the server saves the messages that arrived for the client when it was disconnected.

After the client reconnects, the server will send those messages to the client. If some other MQTT device uses the same client ID and connects to the server, the server will send the messages it had saved to that device instead.

Another scenario related to the client IDs is duplicate connections. Suppose that a particular device using client ID `DeviceA` is connected to the MQTT server. If another client includes the same client ID (`DeviceA`), the server can allow the new client to connect and disconnect the existing client, or keep the old connection open and disallow the new client. This is an optional feature of an MQTT server.

Durable and non-durable subscribers with MQTT

A *durable subscriber* is any client that can receive all of the messages published on a topic, including messages published while the client was inactive. A client is said to be inactive in the following cases:

- A client has connected and subscribed to a topic, and later disconnected from the server without unsubscribing to that topic.
- A client has connected and subscribed to a topic, and later disappeared from the network due to some network issue or lost server connection.

A *non-durable subscriber* is any client that does not intend to receive messages published while it is inactive. A client is said to be non-durable in the following cases:

- A client always uses the clean session flag set to `true` when it connects to the server.
- A client always unsubscribes from all of the topics it subscribed for, before disconnecting.

In MQTT, QoS 0 messages are never persisted while a durable subscriber is inactive. Only QoS 1 and QoS 2 messages are persisted by the server and sent when the client becomes active. MQTT further provides a facility that enables MQTT clients to refrain from receiving messages while they were disconnected by setting the clean start flag to `true`. Here is an example flow for a typical durable MQTT subscriber:

1. MQTT ClientA connects to the server by specifying the clean session flag as `false`.
2. ClientA subscribes to the topic/test topic.
3. MQTT ClientB connects to the server and publishes messages from QoS 1 and QoS 2 to the topic/test topic.
4. ClientA receives the messages.
5. ClientA disconnects from the server (note that ClientA did not unsubscribe).
6. ClientB publishes more messages from QoS 1 and QoS 2.
7. ClientA connects to the server by specifying the clean session flag as `false`.
8. ClientA receives all of the messages that were published while it was inactive, and all of the future messages that are published to the topic/test topic.

Durable subscribers are useful when the subscribing client needs all of the messages that are published to the topic it subscribes to. Therefore, a client does not really need to worry about being in the network, because it is assured of receiving the messages while it was disconnected or even lost connection.

Although durable subscription is a nice feature to have, it adds additional responsibility to the server to hold the messages until the client connects back to the server.

MQTT persistence

The MQTT protocol is designed with the assumption that the client and server are generally reliable. What this means is that the client and the server will not fail, hang, or get into power failure issues. The assumption might prove to be costlier in some situations for the clients and therefore a feature known as *MQTT client persistence* is provided. The local persistence store is used to achieve QoS 1 and QoS 2 level message flows:

- Before the client actually sends the **`PUBLISH`** command, the client stores the message data on a disk (or any other available storage).
- The client, when it receives an acknowledgement, deletes the message from the disk. Therefore, in case of a power failure and restart of the client application, the first action soon after the reconnect is to check the pending messages in the client persistence, and then send them. On the server side, this feature is generally managed by the messaging engine.

Message delivery from JMS to MQTT

When messages are sent from a Java Message Service (JMS) application to MQTT applications, the JMS reliability and persistence map to MQTT QoS levels. For the fastest but least reliable messaging between JMS and MQTT, non-persistent JMS messages can be sent with the IBM MessageSight DisableACK property set to `true`. This configuration provides QoS 0 for MQTT subscriber applications.

For the best reliability but slowest messaging rates, send persistent JMS messages and user per-message acknowledgements. This configuration provides up to QoS 2 for MQTT subscribers.

Table A-4 shows the possible combinations for message delivery from JMS to MQTT.

Table A-4 Mapping of JMS to MQTT messages

JMS message type sent	QoS of matching MQTT subscription	Reliability of delivered message	Persistence of delivered message
Either non-persistent or persistent	QoS 0	At most once (QoS 0)	Non-persistent
Non-persistent, acknowledgements turned off	QoS 1	At most once (QoS 0)	Non-persistent
Non-persistent	QoS 1	At least once (QoS 1)	Non-persistent
Persistent	QoS 1	At least once (QoS 1)	Persistent
Non-persistent, acknowledgements turned off	QoS 2	At most once (QoS 0)	Non-persistent
Non-persistent	QoS 2	At least once (QoS 1)	Non-persistent
Persistent	QoS 2	Exactly once (QoS 2)	Persistent

MQTT header

The MQTT protocol only requires a two byte header. Consider a scenario where a client must send a message (for example, `Hello`) 10 times. Additionally, the client must tell the server that the destination is a/b for every message. We can calculate how many additional bytes of data will flow from client to server for one of the scenarios.

For sending the messages using MQTT, here is the computation of the total number of bytes:

1. CONNECT: Fixed (2 bytes) + Variable (12 bytes) = 14 bytes.
2. CONNACK: Fixed (2 bytes) + Variable (2 bytes) = 4 bytes.

3. PUBLISH: Fixed (2 bytes) + Variable: Length(a/b) = 3.

   So, two bytes for representing the topic name, followed by the topic name, followed by the msg ID, followed by the payload, which is 2 + 3 + 2 + 5 = 12 bytes.
4. There are 10 publications happening... therefore (12) 10 = 120 bytes.
5. DISCONNECT: Fixed (2 bytes) = 2 bytes.

Therefore, this scenario needs 14 + 4 + 120 + 2 = 140 bytes.

As you can see from this example, the fixed portion of every message is two bytes. The total fixed amount is 140 - 80 = 60 bytes for this scenario.

MQTT keep alive

An MQTT server is able to determine if the MQTT client is still in the network (and vice versa) by using the *keep alive timer*. The TCP/IP time out and error handling mechanisms are at the network layer, but the MQTT client and server need not depend on that. The client says hello to the server and the server responds back acknowledging it. That is how simple keep alive works.

According to MQTT, the client needs only to tell the server that it is alive when there has not been any interaction between them for a period of time. The time period is a configurable option. This option can be set while connecting to the server by the client. It is the client that chooses the keep alive time. The server just keeps a record of the value in the client information table on the server side.

There are two messages that constitute the keep alive interaction:

- PINGREQ

  The ping request is sent by the client to the server when the keep alive timer expires.
- PINGRESP

  The ping response is the reply sent by the server for a ping request from the client.

Both these messages are short 2-byte messages. They are not associated with any QoS. The PINGREQ is sent only once, and the client waits for the PINGRESP.

Internally, there is a simple timer that triggers in the client. The timer checks if there has been any other recent activity between the client and the server. If not, a PINGREQ is sent. The client waits until a PINGRESP is received.

On the server side also, if there is no message received from the client within the keep alive time period, the server disconnects the client. But the server offers a grace period to the client, which is an additional 50% of the keep alive time period.

When choosing a keep alive time for a client, there are a few considerations. If the MQTT client is less active and sends one or two messages an hour, and if the keep alive is set to a low value (for example, 10 seconds), the network can be flooded with PINGREQ and PINGRES messages.

Alternatively, if the keep alive timer is set to a high value (for instance, one hour), and the client goes out of the network, the server will not know that the client has gone away for a long time. This can affect administrators who monitor the connected clients. In addition, the server keeps trying PUBLISH messages again. The keep alive timer setting ought to be chosen based on the message flow and amount of time that the client can be idle.

Retry message delivery

Consider a scenario when a message of QoS 1 is published from a client to a server, but the PUBACK is not received from the server. The MQTT client tries sending the message again after a specified time period. This time out is different from the keep alive. The retry time out is the maximum amount of time that the client waits for the server to send a response (or vice versa).

This retry happens until there is an error from the TCP/IP layer. That is, if there is any type of socket exception, retry processing ceases. In cases where QoS 2 message delivery is not used, there might be duplicate messages seen due to retry by the MQTT client or server. Therefore, when a retry happens, the message is marked with a duplicate flag. The application receiving the message knows if the received message is a duplicate.

It is possible that a duplicate message is received by an MQTT client while using QoS 1 due to MQTT retry. On some slow and fragile networks, one can observe a large number of duplicate messages due to the retry processing. If the retry time out is low, the network can be clogged with a large number of duplicates.

Last Will and Testament

When a client connects to the server, it can define a topic and a message that needs to be published automatically when it unexpectedly disconnects. This is called the *Last Will and Testament* (LWT).

If an unexpected disconnect happens wherein the client does not send a DISCONNECT command to the server, the publication is sent by the telemetry service when it detects that the connection to the client has broken without the client requesting a disconnect. The client might have experienced a loss in network connection, or an abrupt termination of the client program.

When a client connects to the server, the following “Will” parameters can be specified:

- The Will topic at which the Will message needs to be published.
- The Will message that needs to be published.
- The Will QoS of the message that will get published on the topic.
- The Will retain flag that signifies if the message needs to be retained.

When the client unexpectedly disconnects, the keep alive timer at the server side detects that the client has not sent any message or the keep alive PINGREQ. Therefore, the server immediately publishes the Will message on the Will topic specified by the client.

The LWT feature can be useful in some scenarios. For remote MQTT clients, this feature can be used to detect when the device goes out of the network. The LWT feature can be used to create notifications for an application that is monitoring the client activity.

Retained flag on messages

MQTT provides a feature for holding a message for a topic even after it is delivered to the connected subscribers. This is achieved through a retained flag on the publish message.

The publisher of the message sets this flag on the message while publishing. Here is an example flow to understand retained messages:

1. Client A connects to the server and subscribes to the `a/b` topic.
2. Client B connects to the server and publishes the message `Hello` with the retain flag set to `a/b`.
3. Client A receives the message without the retain flag set.
4. Client C connects to the server and subscribes to `a/b`.
5. Client C receives the message with the retain flag set.

Even if the server is restarted, the retained message will not be lost. It is also important to note that there is only one retained message that is held per topic. The retained publications are primarily used to maintain state information.

If a particular topic is used to publish a state message from a device, the messages can be retained messages. The advantage is that the new monitoring program that connects can subscribe to this topic, and gets to know the last published state messages from the device.

TCP/IP

When designing applications for mobile wireless networks, give consideration to TCP/IP. MQTT is just like any other protocol, such as HTTP or FTP, for the TCP/IP stack. TCP/IP is the communication protocol of the Internet. It defines how the client and server need to communicate with each other. All of the major operating systems on the server support TCP/IP.

The client devices also comes with the operating system that has a built-in capability of TCP/IP. Although MQTT adds only two bytes of additional header to a message, there is more data that flows in the network layers. The additional header data cannot be exactly computed on a per-MQTT message basis. The IP header, frame header, and other keep alive flows at the network layer add to additional data flows while using MQTT clients.

TCP/IP port 1883 is reserved with for use with MQTT. TCP/IP port 8883 is also registered for using MQTT over Secure Sockets Layer (SSL).

B

IBM Mobile Messaging and M2M Client Pack MA9B

The IBM Mobile Messaging and M2M Client Pack MA9B provides a software development kit (SDK) for building MQTT-based mobile messaging and machine-to-machine (m2m) applications.

MQTT is an open lightweight messaging protocol that is well suited for use in mobile, device, and m2m applications. The SDK contains client libraries and samples for use in building these types of applications. This appendix describes the following topics about IBM Mobile Messaging and M2M Client Pack MA9B:

- Client libraries
- Samples
- IBM Mobile Messaging and M2M Client Pack MA9B system requirements
- Download location

Client libraries

IBM Mobile Messaging and M2M Client Pack MA9B provides two Java client libraries. To read more details about MA9B, see the following website:

http://www-01.ibm.com/support/docview.wss?uid=swg27038199

Java Client: org.eclipse.paho.client.mqttv3

The Java client library is found in the `../SDK/clients/java` directory of the MA9B client pack. This Java library implements the client portion of the MQTT V3.1 protocol implementation. The Java client runs on any suitable Java run time (Java 1.5 and later), including Android. Separate interfaces are provided for synchronous and asynchronous styles of operation.

The synchronous application programming interface (API) supports a style where an operation only returns to the caller when the operation has completed. This is a traditional style, which might be used to implement a simple client. However, the blocking nature of this API limits its usage in environments where threads are not allowed to block, or when high performance or large numbers of clients are required.

The asynchronous API supports a different style in which a call returns immediately. The application can then either be notified through a callback when the operation completes, or can use a token returned to the application to block until the operation completes. This style of API is better suited to mobile, event-oriented, and high-performing applications. Applications where it might take time to complete an operation, or where a thread must not be blocked, are good candidates for the asynchronous API.

This client is a version of the open source MQTT client that is available from the Eclipse.org Paho project.

Java Client: com.ibm.micro.client.mqttv3

The Java client library is found in the `../SDK/clients/java` directory of the MA9B client pack. This Java library implements the client portion of the MQTT V3.1 protocol. An interface is provided for the synchronous style of operation. This is the same interface that was first included with WebSphere MQ V7.0.1, and is suitable for use with existing applications. This interface has been stabilized.

For new applications, use the client in the `org.eclipse.paho.client.mqttv3` package. To use this library, the Eclipse Paho version of the client (`org.eclipse.paho.client.mqttv3v3.jar`) must be included on the class path.

Samples

IBM Mobile Messaging and M2M Client Pack MA9B provides a variety of samples.

Java sample

A set of Java samples that shows how to program using both synchronous and asynchronous programming styles is provided.

The pre-built Java archive (JAR) file is `org.eclipse.paho.sample.mqtv3app.jar`.

Source code for this sample is available in the `../SDK/clients/java/samples` directory of the MA9B client pack.

Android sample

The Android sample for the MQTT mobile application can be found in the `../SDK/clients/android` directory of the MA9B client pack. Native Android applications are made up of one or more activities and, optionally, one or more services.

Although a native Android application can call the asynchronous Java client directly from an activity, it often makes more sense to call the asynchronous Java client from a service. A service can continue running (and in particular, can be receiving incoming messages) when the application activities have stopped.

The MA9B client pack provides a sample Android application (called `mqttExerciser`) that connects to an MQTT server and sends and receives messages. This application is made up of an Android activity that is bound to an Android service that encapsulates all of its MQTT API calls.

C client

The C sample client for MQTT is in the `../SDK/clients/c` directory of the MA9B client pack. The C implementation of the MQTT V3.1 protocol, similar to the Java equivalent, has separate interfaces provided for synchronous and asynchronous styles of operation. These clients are provided for a variety of platforms.

This client is a version of the open source MQTT client that is available from the Eclipse.org Paho project.

Client for JavaScript

The client for JavaScript is in the `../SDK/WebContent/WebSocket` directory of MA9B client pack. JavaScript implementation of the MQTT V3.1 protocol enables HTML web pages to connect to servers and send MQTT messages over a WebSocket. The JavaScript implementation can be loaded by the browser from a suitably configured WebSphere MQ queue manager, or from a web server.

Restriction: User interfaces resulting from the Sample code are *not* warranted to be compliant to any accessibility standards or accessibility requirements.

IBM Mobile Messaging and M2M Client Pack MA9B system requirements

Support is available on the latest level of code, and clients must upgrade to the most recent level of the IBM Mobile Messaging and M2M Client Pack MA9B from the IBM developerWorks Messaging website:

https://www.ibm.com/developerworks/mydeveloperworks/blogs/c565c720-fe84-4f63-873f-607d87787327/entry/download?lang=en

The IBM Service team might also request that a problem is reproduced on a reference platform.

IBM Messaging client for JavaScript

The following sections provide information about the system requirements for JavaScript.

Supported platforms

All major browser versions that are compliant with the RFC 6455 (WebSocket) Standard are supported.

Reference platforms

The following list includes the reference platforms for JavaScript:

- Samsung Galaxy S III Android 4.1 Firefox (latest version)
- Apple iPhone 5 iOS 7 Safari (latest version)
- Intel (x86-32) Microsoft Windows 7 Chrome (latest version)
- Intel (x86-64) Linux Red Hat 6 Firefox (latest version)

Java clients

The following sections provide information about the system requirements for Java clients.

Supported platforms

The Java client runs on any platform that has a suitable Java runtime environment (JRE). The client can be run on the following Java run times:

- JRE 1.5 or later that is Java compatible

Reference platforms

The following list includes the reference platforms for Java clients:

- Samsung Galaxy S III/Android 4.1/Native (built-in) Java run time for Android
- Samsung Galaxy Tab 2 10.1/Android 4.0/Native (built-in) Java run time for Android
- Intel (x86-32)/Windows 7/IBM Java 1.5
- Intel (x86-32)/Red Hat 6/IBM Java 1.6

C clients (synchronous)

The following sections provide information about the system requirements for synchronous C clients.

Supported platforms

The client can be run on the following supported platforms:

- iOS 7.0 and future OS fix packs
- Currently supported platforms for C client in WebSphere MQTT

Reference platforms

The following list includes the reference platforms for synchronous C clients:

- Apple iPad 3/iOS 6.0
- Apple iPhone 5/iOS 7.0
- Intel (x86-32)/Windows 7 Professional/Microsoft Visual Studio 2010
- Intel (x86-32)/Red Hat Enterprise Linux (RHEL) Server 6.0 with glibc 2.12 and kernel 2.6.32
- Intel (x86-64)/RHEL Server 6.0 with glibc 2.12 and kernel 2.6.32

C clients (asynchronous)

The following sections provide information about the system requirements for asynchronous C clients.

Supported platforms

The client can be run on the following supported platforms:

- iOS 7.0 and later OS fix packs
- Intel Linux glibc 2.12 and kernel 2.6.32 and future OS fix packs
- Windows XP, Windows Vista, Windows 7 and future OS fix packs

Reference platforms

The following list includes the reference platforms for asynchronous C clients:

- Apple iPad 3/iOS 6.0
- Apple iPhone 5/iOS 7.0
- Intel (x86-32)/Windows 7 Professional/Visual Studio 2010
- Intel (x86-32)/RHEL Server 6.0 with glibc 2.12 and kernel 2.6.32
- Intel (x86-64)/RHEL Server 6.0 with glibc 2.12 and kernel 2.6.32

Download location

You can download the most recent level of the IBM Mobile Messaging and M2M Client Pack MA9B from the IBM developerWorks Messaging website:

https://www.ibm.com/developerworks/mydeveloperworks/blogs/c565c720-fe84-4f63-873f-607d87787327/entry/download?lang=en

C

MQTT hybrid application for Android code example

This appendix lists the example code of The ITSOTransport HTML user interface and JavaScript application logic, as described in 5.8.5, “Creating application logic and the user interface” on page 163.

ITSOTransport HTML user interface

Example C-1 shows details about the HTML user interface implementation of the ITSOTransport hybrid application.

Example C-1 ITSOTransport HTML user interface

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>ITSO Transport</title>
<meta name="viewport"
content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0, user-scalable=0">
<link rel="shortcut icon" href="images/favicon.png">
<link rel="apple-touch-icon" href="images/apple-touch-icon.png">
<link rel="stylesheet" href="css/ITSOTransport.css">
<script>
window.$ = window.jQuery = WLJQ;
</script>

</head>

<body onload="WL.Client.init({connectOnStartup:false})"
id="content"style="display: none">

<div id="connectContent" style="display: none;">
<img border="0" height="86" src="images/ITSOTransportHeader.png"
style="margin-left: 10px; margin-top: 80px" width="288">
<form class="tabContent" id="connect">
<fieldset>
<table>
<tr>
<td>Driver Id</td>
<td><input type="text" name="userName" size="27"
maxlength="20" />
</tr>
<tr>
<td>Password</td>
<td><input type="password" name="password" size="27"
maxlength="20" />
</tr>
<tr>
<td>Truck Number</td>
<td><input type="text" name="truckNumber" size="27"
maxlength="20" value="TruckNumberYouDriving"/>
</tr>
```

```
<tr>
<td><input name="loginButton" onclick="login(this.form)"
type="button" value="Login"></td>
<td><input name="logoutButton" onclick="logout(this.form)"
type="button" value="Logout"> <input
name="storeCredentials" type="checkbox" checked>Remember
Credential</td>
</tr>
</table>
</fieldset>
</form>
</div>

<div id="sendReceiveContent" style="display: none;">
<img border="0" height="86" src="images/ITSOTransportHeader.png"
style="margin-left: 10px; margin-top: 80px" width="288">
<form class="tabContent" id="sendReceive">
<fieldset>
<table>
<tr>
<td><input name="flatTire" onclick="flatTireClick(this.form)"
type="radio"></td>
<td>Flat Tire</td>
</tr>
<tr>
<td><input name="acmalfunction"
onclick="acMulfuntionClick(this.form)" type="radio"></td>
<td>Air Condition Malfunction</td>
</tr>
<tr>
<td><input name="truckAccident"
onclick="truckInAccidentClick(this.form)" type="radio"></td>
<td>Truck in Accident</td>
</tr>
</table>
</fieldset>
<fieldset>
<table>
<tr>
<td>Assistance Request Text</td>
<td>QoS <input type="text" name="requestQoS" size="2"
maxlength="2" /></td>
</tr>
</table>
<table>
<tr>
<td><textArea cols="40" name="assistanceText" rows="3"></textArea></td>
</tr>
</table>
```

```
<table>
<tr>
<td><input name="assistanceButton"
onclick="assistance(this.form)" type="button"
value="Ask Assistance"></td>
</tr>
</table>
</fieldset>
<fieldset>
<table>
<tr>
<td>Assistance Request Response</td>
<td>QoS <input type="text" name="responseQoS" size="2"
maxlength="2" /></td>
</tr>
</table>
<table>
<tr>
<td><textArea cols="40" name="assistanceResponse"
readonly="readonly" rows="3"></textArea></td>
</tr>
</table>
<table>
<tr>
<td><input name="clearResponse"
onclick="clearAssistanceResponse(this.form)" type="button"
value="Clear Response"></td>
</tr>
</table>
</fieldset>
</form>
</div>


<div id="notificationContent" style="display: none;">
<img border="0" height="86" src="images/ITSOTransportHeader.png"
style="margin-left: 10px; margin-top: 80px" width="288">
<form class="tabContent" id="notify">
<table>
<tr>
<td><textArea cols="42" name="notificationText"
readonly="readonly" rows="19"></textArea></td>
</tr>
<tr>
<td><input name="notCleanButton"
onclick="clearNotification(this.form)" type="button" value="Clear
Notification">
Last Notification's QoS <input type="text" name="notificationQoS"
size="2" maxlength="2" /></td>
```

```
</tr>
</table>
</form>
</div>


<div id="historyContent" style="display: none;">
<img border="0" height="86" src="images/ITSOTransportHeader.png"
style="margin-left: 10px; margin-top: 80px" width="288">
<form class="tabContent" id="history">
<table>
<tr>
<td><textArea cols="42" name="historyText" readonly="readonly"
rows="19"></textArea></td>
</tr>
<tr>
<td><input name="hisCleanButton"
onclick="clearHistory(this.form)" type="button" value="Clear History"></td>
</tr>
</table>
</form>
</div>

<script src="js/initOptions.js"></script>
<script src="js/ITSOTransport.js"></script>
<script src="js/messages.js"></script>
<script type="text/javascript" src="js/mqttCordovaAndroidClient.js"></script>
</body>
</html>
```

ITSOTransport JavaScript application logic

Example C-2 shows details about the JavaScript functional implementation of the ITSOTransport hybrid application.

Example C-2 ITSOTransport JavaScript application logic

```
function wlCommonInit(){
setupApplication();
}

/** MQTT API. */
var client;
var connectOptions;

/** The last message received. */
```

```
var receivedMessage;

/** Variable used for connection*/
var serverName = "***.***.***.***";
var serverPort = "*****";
var cleanSession = false;
var keepAliveInterval = 60;

/** Last Will and Testament */
var LWTTopicName ="";
var LWTMessage ="";
var LWTQos ="";
var LWTRetained ="";

/**Quality of Service Constants*/
var qos0 = 0;
var qos1 = 1;
var qos2 = 2;

/**Request/Response Topic**/
var rrPubTopicString;
var rrSubTopicString;
var rrPubTopicStringFinal;
var rrPubQoS = qos2;
var rrSubQoS = qos2;
//var assistanceRequestType;
var assistanceRequestSent = false;

/**GeoLocation Topic**/
var geoPubTopicString;
var geoSubTopicString;
var geoPubQoS = qos0;
var geoSubQoS = new Array(qos1,qos2);

/**message builder delimiter */
var mdlt = ",";

var watchID;
var cachedLatitude;
var cachedLongitude;
var currentTimeStamp;

/**
 * HTML Application logic.
 */
function setupApplication() {

// Initialise the UI for disconnected state
preConnectState();
```

```
//-------------------------------------------------------------------
// Tab Bar
//-------------------------------------------------------------------
WL.TabBar.init();
WL.TabBar.addItem("connectionTab", function(){ selectTab(0);
},"Login",{image:"images/Login.png", imageSelected:"images/Login.png"});
WL.TabBar.addItem("sendReceiveTab", function(){ selectTab(1);
},"Assistance",{image:"images/Assistance.png",
imageSelected:"images/Assistance.png"});
WL.TabBar.addItem("NotificationTab", function(){ selectTab(2);
},"Notification",{image:"images/Notification.png",
imageSelected:"images/Notification.png"});
WL.TabBar.addItem("historyTab", function(){ selectTab(3);
},"History",{image:"images/History.png", imageSelected:"images/History.png"});
WL.TabBar.setVisible(true);
selectTab(0);
}


/**
 * Display a tab.
 *
 * @param tab0=connect, 1=send/receive, 2=notification, 3 history
 */

function selectTab(tab) {

document.getElementById("connectContent").style.display = "none";
document.getElementById("sendReceiveContent").style.display = "none";
document.getElementById("notificationContent").style.display = "none";
document.getElementById("historyContent").style.display = "none";

if (tab === 0) {
document.getElementById("connectContent").style.display = "block";
}
else if (tab === 1) {
document.getElementById("sendReceiveContent").style.display = "block";
}
else if (tab === 2) {
document.getElementById("notificationContent").style.display = "block";
}
else if (tab === 3) {
document.getElementById("historyContent").style.display = "block";
}
}


function login(form) {
```

```
//if(form.userName.value.trim()!="" && form.password.value.trim()!="" &&
form.truckNumber.value.trim()!="")
//{
try
{
client = new Messaging.Client(serverName, Number(serverPort),
form.truckNumber.value);
} catch(exception){
WL.Logger.error("Exception creating client", exception);
alert("Exception creating client");
}

// Set up Connection Options, using values from the connect form.
var connectOptions = new Object();

connectOptions.userName = form.userName.value;
connectOptions.password = form.password.value;
connectOptions.cleanSession = cleanSession;
connectOptions.keepAliveInterval = keepAliveInterval;

//connectOptions.userName = "";
//connectOptions.password = "";

// Set up callback functions to be notified when the connect call completes
connectOptions.onSuccess = function() {

postConnectState();
logHistory("Connected to ITSO Transport  Message Sight Server");
logHistory("Connected with Client Id = "+ client.clientId);

//setup Topics
rrPubTopicString  = new
Array("/itso/driver/assistance/request/FlatTire/"+client.clientId,
"/itso/driver/assistance/request/ACMalfunction/"+client.clientId,
"/itso/driver/assistance/request/Accident/"+client.clientId);
rrSubTopicString  = "/itso/driver/assistance/response/"+client.clientId;
geoPubTopicString = "/itso/driver/geolocation/"+client.clientId;
geoSubTopicString = new Array("/itso/driver/notification/AllTrucks",
"/itso/driver/notification/"+client.clientId);

//setup navigation capturing
watchID = navigator.geolocation.watchPosition(geoLocationWatchSuccess,
geoLocationWatchFailure, { enableHighAccuracy: true, timeout: 30000});
navigator.geolocation.getCurrentPosition(function(position){cachedLatitude=posi
tion.coords.latitude;cachedLongitude=position.coords.longitude;},
geoLocationWatchFailure, { enableHighAccuracy: true, timeout: 30000});

//subscribe for assistance and critical alert topics
subscribeTopic(rrSubTopicString, rrSubQoS);
logHistory("Subscribe to Topic = "+ rrSubTopicString);
subscribeTopic(geoSubTopicString[0], geoSubQoS[0]);
logHistory("Subscribe to Topic = "+ geoSubTopicString[0]);
subscribeTopic(geoSubTopicString[1], geoSubQoS[1]);
logHistory("Subscribe to Topic = "+ geoSubTopicString[1]);

WL.Toast.show("Connection Established and Subscriptions created");
};

connectOptions.onFailure = onClientDisconnect;
// Set up the Client-wide callback handlers
client.onConnectionLost = onClientDisconnect;
client.onMessageArrived = onMessageArrived;
client.onMessageDelivered = function(message) {

};
client.connect(connectOptions);
//}else
//{
//WL.Toast.show("Please enter all credentials to login");
//}
}

function geoLocationWatchSuccess(position) {

//cache the changed GeoLocation for reference and assistance
cachedLatitude  = position.coords.latitude;
cachedLongitude = position.coords.longitude;

doSend(geoPubQoS,geoPubTopicString,buildGeoMessage());
}

function geoLocationWatchFailure(error) {

}

/**
 * Function that is called when the driver presses the Logout button
 *
 */
function logout(form) {

logoutCleanup();
WL.Toast.show("Client disconnected, Credentials and History deleted");
preConnectState();

if(watchID!=null) {
navigator.geolocation.clearWatch(watchID);
watchID=null;
```

```
}
}

function logoutCleanup() {

unsubscribeTopic(rrSubTopicString);
unsubscribeTopic(geoSubTopicString[0]);
unsubscribeTopic(geoSubTopicString[1]);

client.disconnect();
cleanCredentials();

form = document.getElementById("connect");
form.userName.value = "";
form.password.value ="";
form.truckNumber.value = "TruckNumberYouDriving";

form = document.getElementById("sendReceive");
form.flatTire.checked = false;
form.acmalfunction.checked = false;
form.truckAccident.checked = false;
form.assistanceText.value = "";
form.assistanceResponse.value="";
form.requestQoS.value="";
form.responseQoS.value="";

form = document.getElementById("notify");
form.notificationText.value = "";
form.notificationQoS.value = "";

form = document.getElementById("history");
form.historyText.value = "";
}

/**
 * Function that is called when the application is initialized
 *
 * @param {Object}
 *            The UI form containing subscribe options requested by the
end-user
 */

function subscribeTopic(topicFilter, qos) {

var options = {qos:qos,
invocationContext:{filter: topicFilter},
onSuccess:subscribeTopicSucceeded,
onFailure:subscribeTopicFailed};
```

```
try {
client.subscribe(topicFilter, options);

} catch (exception) {
alert("Subscribe to " + topicFilter + " failed");
}
}

function subscribeTopicSucceeded(result) {

}

function subscribeTopicFailed(result) {

}

/**
 * Function that is called when the driver disconnects
 */

function unsubscribeTopic(topicFilter) {
var options = { invocationContext:{filter: topicFilter},
onSuccess:unsubscribeTopicSucceeded,
onFailure:unsubscribeTopicFailed};
try {
client.unsubscribe(topicFilter, options);

} catch (exception) {
alert("Unsubscribe from " + topicFilter + " failed");
}
}

function unsubscribeTopicSucceeded(result) {
}

function unsubscribeTopicFailed(result) {

}

/**
 *  Client-level API callbacks.
 */
function onConnect() {
connectForms();
}

function onClientDisconnect(reason) {

WL.Logger.debug("connection lost >> reason: "+ reason.errorMessage);
WL.Toast.show(reason.errorMessage);
//disconnectForms();
// If normal disconnect

if (reason.errorMessage == undefined)
logHistory("Disconnected");
else
logHistory("Unexpected disconnect " + reason.errorMessage);
}

function onMessageArrived(message) {
this.receivedMessage = message;
displayMessage(message);
}

/**Functions to build assistance Messages and GeoLocation Messages*/

/**Scenario 1 Request Response Message**/
function  buildAssistanceMessage(textFromDriver)
{
var assistanceMessage = getAndriodTimeStamp()+mdlt+
localStorage.userName+mdlt+
cachedLongitude+mdlt+
cachedLatitude+mdlt+
textFromDriver;
return assistanceMessage;
}

/**Scenario 2 GeoLocation Message**/
function buildGeoMessage()
{
var geoMessage = getAndriodTimeStamp()+mdlt+
localStorage.userName+mdlt+
cachedLongitude+mdlt+
cachedLatitude;
return geoMessage;
}

function displayMessage(message) {

var topicName = message.destinationName;
if(topicName == rrSubTopicString && assistanceRequestSent)
{
logResponse(message.payloadString, message.qos);
WL.Toast.show("Response to your request is received");
}
if(topicName == geoSubTopicString[0] || topicName == geoSubTopicString[1])
{
logNotification(message.payloadString, message.qos);
WL.Toast.show("Critical Alert");
}
}

function doSend(qosSend,topicSend,data) {
var message            = new Messaging.Message(data);
message.destinationName = topicSend;
message.qos            = qosSend;
client.send(message);
}

function flatTireClick(form) {

form.flatTire.checked = true;
form.acmalfunction.checked = false;
form.truckAccident.checked = false;

form.assistanceText.disabled = "";
form.assistanceButton.disabled = "";
form.assistanceResponse.disabed = "";
form.clearResponse.disabled = "";
form.assistanceText.value="";
form.assistanceResponse.value="";
form.requestQoS.value = rrPubQoS;
form.responseQoS.value="";

rrPubTopicStringFinal = rrPubTopicString[0];
}

function acMulfuntionClick(form) {

form.flatTire.checked = false;
form.acmalfunction.checked = true;
form.truckAccident.checked = false;

form.assistanceText.disabled = "";
form.assistanceButton.disabled = "";
form.assistanceResponse.disabed = "";
form.clearResponse.disabled = "";
form.assistanceText.value="";
form.assistanceResponse.value="";
form.requestQoS.value = rrPubQoS;
form.responseQoS.value="";

rrPubTopicStringFinal = rrPubTopicString[1];
}

function truckInAccidentClick(form) {
```

```
form.flatTire.checked = false;
form.acmalfunction.checked = false;
form.truckAccident.checked = true;

form.assistanceText.disabled = "";
form.assistanceButton.disabled = "";
form.assistanceResponse.disabed = "";
form.clearResponse.disabled = "";
form.assistanceText.value="";
form.assistanceResponse.value="";
form.requestQoS.value = rrPubQoS;
form.responseQoS.value="";

rrPubTopicStringFinal = rrPubTopicString[2];
}


function assistance(form) {
if(form.assistanceText.value.trim()!="")
{
doSend(rrPubQoS,rrPubTopicStringFinal,buildAssistanceMessage(form.assistanceTex
t.value));
assistanceRequestSent = true;
}
}

function cleanCredentials() {
localStorage.removeItem("userName");
localStorage.removeItem("passWord");
localStorage.removeItem("truckNumber");
}

function clearAssistanceResponse(form) {

if(form.assistanceResponse.value.trim()!="")
{
form.assistanceResponse.value="";
form.responseQoS.value="";
}
if(form.assistanceText.value.trim()!="")
{
form.requestQoS.value="";
form.assistanceText.value="";
}

}

function clearNotification(form) {
```

```
if(form.notificationText.value.trim()!=""){
form.notificationText.value="";
form.notificationQoS.value="";
WL.Toast.show("Notifications Cleared");
}
else{
WL.Toast.show("No Notification to Clear");
}
}

function clearHistory(form) {
if(form.historyText.value.trim()!=""){
form.historyText.value="";
WL.Toast.show("History Cleared");}
else{
WL.Toast.show("No History to Clear");}
}

function preConnectState() {

form = document.getElementById("connect");
form.userName.disabled = "";
form.password.disabled ="";
form.truckNumber.disabled ="";
form.loginButton.disabled = "";
form.storeCredentials.disabled = true;
form.logoutButton.disabled = true;

form = document.getElementById("sendReceive");
form.flatTire.disabled = true;
form.acmalfunction.disabled = true;
form.truckAccident.disabled = true;
form.assistanceText.disabled = true;
form.assistanceButton.disabled = true;
form.assistanceResponse.disabed = true;
form.clearResponse.disabled = true;
form.requestQoS.disabled=true;
form.responseQoS.disabled=true;

form = document.getElementById("notify");
form.notificationText.disabled = true;
form.notCleanButton.disabled = true;
form.notificationQoS.disabled=true;

form = document.getElementById("history");
form.historyText.disabled = true;
form.hisCleanButton.disabled = true;
}
```

```
function postConnectState() {

form = document.getElementById("connect");
form.userName.disabled = true;
form.password.disabled = true;
form.truckNumber.disabled = true;
form.loginButton.disabled = true;
form.storeCredentials.disabled = true;
form.logoutButton.disabled = "";

if(localStorage && form.storeCredentials.checked)
{
localStorage.userName = form.userName.value;
localStorage.passWord = form.password.value;
localStorage.truckNumber = form.truckNumber.value;
}

form = document.getElementById("sendReceive");
form.flatTire.disabled = "";
form.acmalfunction.disabled = "";
form.truckAccident.disabled = "";
form.requestQoS.disabled=true;
form.responseQoS.disabled=true;
form.assistanceText.value= "";
form.assistanceResponse.value="";

form = document.getElementById("notify");
form.notificationText.disabled = "";
form.notCleanButton.disabled = "";
form.notificationQoS.disabled=true;
form.notificationText.value = "";

form = document.getElementById("history");
form.historyText.disabled = "";
form.hisCleanButton.disabled = "";
form.historyText.value = "";
}

function logHistory(text) {
var form = document.getElementById('history');
form.historyText.value = form.historyText.value + "\n" + "*
"+getAndriodTimeStamp()+" " + text;
form.historyText.scrollIntoView(false);
}

function logNotification(text, qos) {
var form = document.getElementById('notify');
form.notificationQoS.value=qos;
form.notificationText.value = form.notificationText.value + "\n" + "*
"+getAndriodTimeStamp()+" "+ text;
form.notificationText.scrollIntoView(false);
}

function logResponse(text, qos) {
var form = document.getElementById("sendReceive");
form.responseQoS.value=qos;
form.assistanceResponse.value = text;
assistanceRequestSent = false;
}

function getAndriodTimeStamp() {

var deviceDate = new Date();
var currentTimeStamp = padTime(deviceDate.getFullYear())+"-"+
padTime(deviceDate.getMonth())+"-"+
padTime(deviceDate.getDate())+"-"+
padTime(deviceDate.getHours())+"."+
padTime(deviceDate.getMinutes())+"."+
padTime(deviceDate.getSeconds())+"."+
padTime(deviceDate.getMilliseconds()+"0");
return currentTimeStamp;
}

function padTime(t) {
if (t < 10) return "0"+t;
else return t;
}
```

D

Additional material

This book refers to additional material that can be downloaded from the Internet, as described in the following sections.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server:

`ftp://www.redbooks.ibm.com/redbooks/SG248183`

Alternatively, you can go to the IBM Redbooks website:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248183.

Using the web material

The additional web material that accompanies this book includes the following files:

File name	*Description*
Chapter3SampleSourceCode.zip	This file includes the sample code described in Chapter 3, “Integration with enterprise systems” on page 39.
ITSOTransport.zip	This file includes the sample code described in Chapter 5, “MQTT with mobile platforms” on page 99.
ITSOAssistanceRequestResponse.zip	This file includes the sample code described in Chapter 8, “Scenario 2: Request and response using MQTT” on page 201.
ITSOGeoNotificationand Broadcast.zip	This file includes sample code described in Chapter 9, “Scenario 3: Push notifications with quality of service” on page 225.

System requirements for downloading the web material

The web material requires the following system configuration:

Hard disk space:	300 MB
Operating System:	Windows or Linux

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material `.zip` file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for providing more detailed information about the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only:

- *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*, SG24-8054
- *Extending Your Business to Mobile Devices with IBM Worklight*, SG24-8117

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, drafts, and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- IBM MessageSight

  http://www.ibm.com/software/products/us/en/messagesight/
- WebSphere MQ Telemetry

  http://www.ibm.com/software/products/us/en/wmq-telemetry/
- Eclipse Paho

  http://www.eclipse.org/paho/
- *Running App* at android.com

  http://developer.android.com/training/basics/firstapp/running-app.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM

Redbooks

Responsive Mobile User Experience Using MQTT and IBM

(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

Responsive Mobile User Experience Using MQTT and IBM MessageSight

Learn how to connect mobile customers to your existing messaging enterprise system

See a quick start guide for IBM MessageSight

Understand mobile application development scenarios

IBM MessageSight is an appliance-based messaging server that is optimized to address the massive scale requirements of machine-to-machine (m2m) and mobile user scenarios. IBM MessageSight makes it easy to connect mobile customers to your existing messaging enterprise system, enabling a substantial number of remote clients to be concurrently connected.

The MQTT protocol is a lightweight messaging protocol that uses publish/subscribe architecture to deliver messages over low bandwidth or unreliable networks. A publish/subscribe architecture works well for HTML5, native, and hybrid mobile applications by removing the wait time of a request/response model. This creates a better, richer user experience.

The MQTT protocol is simple, which results in a client library with a low footprint. MQTT was proposed as an Organization for the Advancement of Structured Information Standards (OASIS) standard. This book provides information about version 3.1 of the MQTT specification.

This IBM Redbooks publication provides information about how IBM MessageSight, in combination with MQTT, facilitates the expansion of enterprise systems to include mobile devices and m2m communications. This book also outlines how to connect IBM MessageSight to an existing infrastructure, either through the use of IBM WebSphere MQ connectivity or the IBM Integration Bus (formerly known as WebSphere Message Broker).

This book describes IBM MessageSight product features and facilities that are relevant to technical personnel, such as system architects, to help them make informed design decisions regarding the integration of the messaging appliance into their enterprise architecture.

Using a scenario-based approach, you learn how to develop a mobile application, and how to integrate IBM MessageSight with other IBM products. This publication is intended to be of use to a wide-ranging audience.

SG24-8183-00 ISBN 0738439150