Welcome to the WebSphere® MQ Version 6.0 Quick Tour

This section contains a text version of the interactive Quick Tour. It is included here as an accessibility feature for visually-impaired users who have access to a screen reader. Information about each topic covered in the tour is provided in a two-column table, with the tour text in the left column, and a description of the animation in the right column.

This tour aims to provide an overview of the key concepts and interfaces related to WebSphere MQ. The tour has two sections, core topics and additional topics. These contain the following sections.

Core Topics

- Business Needs - introduces the ways in which this product can work for your business. The topics covered are integration, asynchrony, assured delivery, and scalability.
- Introduction to Messaging and Queuing - introduces the concepts of messages and queues, shows how they are used together, and includes a simple messaging scenario.
- WebSphere MQ Fundamentals - introduces the concepts central to WebSphere MQ messaging environments. These include queue managers, MQ clients, and messaging communications.
- Interacting with WebSphere MQ - introduces the WebSphere MQ administration interfaces and the messaging interfaces available for programs to use.

Additional Topics

- Using Queue Manager Clusters - introduces how queue manager clusters can be used to simplify the administration of complex messaging networks and balance workload between programs.
- Publish/Subscribe messaging - introduces the concept of publish/subscribe and the advantages of using a publish/subscribe messaging environment.
- WebSphere MQ and the Java™ Messaging Service - introduces how WebSphere MQ can use JMS in both point-to-point and publish/subscribe messaging environments.
- WebSphere MQ with z/OS - introduces the concepts unique to the z/OS platform including shared queues and shared channels.

Core Topics

Business needs

This part of the tour shows how the product can work for your business. The topics covered here are integration, asynchrony, assured delivery, and scalability.

WebSphere MQ meets these business needs, and provides an ideal business information messaging and queuing environment.

Integration

Product tour text	Product tour animation
Typical business applications consist of groups of intercommunicating programs. Communication is harder to achieve when these programs: • span business boundaries • are written in different languages • are hosted on different operating systems or processors • use different communications protocols There is a need for a common method of communication, unaffected by languages, operating systems, and protocols.	This part of the animation shows how WebSphere MQ allows the integration of distributed business applications. Two programs appear, and a connection is created between them. A unit of information (a message) is passed over the connection from the first program to the second one. The second program is now enclosed by a business boundary. The names of the languages that the programs might be written in appear: C++, COBOL, Visual Basic, Java™, PL/1, C#. The names of the operating systems on which the programs might be running appear: Windows, Linux, z/OS, iSeries. The names of the communications protocols that might be used appear: HTTP, TCP/IP (SSL), LU6.2. A summary animation shows a message being sent from the first program to the second, across a business boundary, regardless of programming language, operating system or communications protocol.

Asynchrony

Product tour text	Product tour animation
Synchronous communication requires programs to be available at the same time, otherwise some programs are prevented from doing work until others become	This part of the animation shows how WebSphere MQ supports asynchronous communication between programs. Messages are passed from the sending program to the receiving program, while

Product tour text	Product tour animation
available. There is a need for programs to communicate independently of each other (that is, to use asynchronous communication).	the receiving program is available. The receiving program suddenly becomes unavailable. Messages now sent to the receiving program are stored on the receiving machine (outside the program). The receiving program becomes available again. The messages held on the receiving machine are transferred to the program. Note that messages were still sent, even while the receiving program was unavailable.

Assured Delivery

Product tour text	Product tour animation
When all or part of a business system fails, the integrity of the information in transit can be compromised: • information can be lost if it is not safely stored while en route • information can be duplicated if resent unnecessarily There is a need to assure that information is not lost in transit or duplicated.	This part of the animation shows how WebSphere MQ provides assured delivery. Messages are sent to the receiving program without using assured delivery. The receiving program becomes unavailable then available again, but the sent messages are lost. Messages are sent to the receiving program without using assured delivery. The sent messages are stored on the receiver. The receiving program becomes unavailable, then available again. The sending program resends the same messages which are now inadvertently duplicated. Messages are now sent using assured delivery. Messages are stored on a queue at the receiver and are therefore not lost. Messages are sent once and once only and are therefore not duplicated.

Scalability

Product tour text	Product tour animation
As businesses expand their operations (for example, when customer demand grows) they may require additional systems. However, the time and effort required to connect a new system into an existing network may be significant, and may require substantial downtime, interrupting customer service. There is a need for a way to integrate new systems with minimal interruption to service.	This part of the tour shows how WebSphere MQ provides a scalable messaging environment. Three external programs are connected to an existing business program, across the business boundary. The three external programs represent an increase in business. A second business program is added alongside the existing one, inside the business boundary. Messages are passed from the three external programs, to the first business program. Additional messages representing increased business are passed to the second business program.

Introduction to messaging and queuing

This part of the tour introduces the concepts of messages and queues, shows how they are used together, and includes a simple messaging scenario.

Messages

Product tour text	Product tour animation
A message is used by communicating programs to exchange data. The sending program constructs a message containing: • the data to send • a message header (control information, for example message ID and return address)	This part of the animation shows how WebSphere MQ messages are constructed by the programs that send them. The animation shows two programs connected by a message channel. One program needs to send a message to the other program. The sending program creates a message wrapper for holding the message header and data.

Product tour text	Product tour animation
When the message has been constructed by the program, it is then ready to send.	The sending program places a header containing the message ID, expiration and return address into the message wrapper. The sending program places the data to be sent into the wrapper. The completed message is now ready for sending.

Queues

Product tour text	Product tour animation
A queue is where messages are stored until a program receives them. The sending program puts messages onto the appropriate queue. When the receiving program is ready, it gets its messages from the queue.	This part of the animation shows how WebSphere MQ uses queues to safely hold messages in transit. The animation shows a queue on the receiving program. The sending program sends a message to the receiving program. The message is transferred over the connection, and is placed on the queue. The sending program sends a second message to the receiving program. The second message is transferred over the connection, and is placed on the queue next to the first message. The receiving program retrieves the first message from the queue. The receiving program retrieves the second message from the queue.

Messaging topologies

Product tour text	Product tour animation
Messaging topologies can contain different combinations of networked machines and programs. A	This part of the animation shows how WebSphere MQ messaging works on different messaging networks.

Product tour text	Product tour animation
queue is used for holding messages received from programs on the same machine, or other machines. Messages on the queue can also be accessed by programs on the same machine, or other machines.	The animation shows two machines (a sender and a receiver), with a program on each. The machines are connected, and a queue is created on the receiver. A second program is added to the receiver machine. A second machine with a program is now added to the sender side, and a connection to the receiver machine is established. A program on the receiver places a message onto its own queue. A program on a sender places a message on the receiver queue. Two more machines with a program on each are created, and connections established to the receiver machine. A program on the receiver machine retrieves a message from its own queue. A program on one of the new machines retrieves a message from the queue.

A simple messaging scenario

Product tour text	Product tour animation
The flexibility of WebSphere MQ allows it to provide messaging throughout the business. For example, shops can use messaging to send orders to headquarters. Headquarters can use messaging to send delivery requests to depots. Depots can use messaging to confirm delivery dates with	This part of the animation shows how WebSphere MQ allows programs running in different parts of the business to communicate. The animation shows how messaging might be applied in a simple business scenario. Headquarters is represented by a single machine, with two programs on it. Shops are represented by two machines, with a program on each.

Product tour text	Product tour animation
headquarters (which then sends confirmations to shops).	Depots are represented by two machines, with a program on each. The shop and depot machines each have their own connection to headquarters. Messages (for example orders) are placed by the programs on the shop machines, onto the queue on the headquarters machine. The programs on the headquarters machine retrieve these messages from the queue. Queues are created on the depot machines. Messages (for example stock queries) are placed by the programs on the headquarters machines, onto the queues on the depot machines. The programs on the depot machines receive these messages from their respective queues.

Making queue locations transparent

Product tour text	Product tour animation
A program can send messages to a remote queue without needing to know its location, if a remote queue definition has been defined locally. The remote queue definition holds the target queue and transmission queue names. The local queue manager uses this information to determine where to send the message. By using remote queue definitions, message	This part of the animation shows how a WebSphere MQ queue manager uses remote queue definitions to obtain the location of queues before sending messages. A remote queue definition is created on the sender machine. The definition points to a local queue on the receiving machine. A program on the sending machine wants to send a message to a program on the receiving machine. The queue manager on the sender machine obtains the name of the transmission queue for the required destination, from its

Product tour text	Product tour animation
channels and transmission queues, any program can send messages to any queue without needing to know where that queue is located.	remote queue definition. The message is placed on the correct transmission queue, and then transferred over the message channel, to the local queue on the receiving machine. The program on the receiving machine retrieves the message from its local queue. The animation now shows how messages might be sent to other remote queues on the topology, without the sending queue managers having to know explicitly, where the remote queues are physically located.

WebSphere MQ fundamentals

This part of the tour introduces the concepts central to WebSphere MQ messaging environments. These include queue managers, MQ clients and messaging communication.

Queue Managers

Product tour text	Product tour animation
Each computer that hosts queues requires a queue manager. Each queue manager has a unique name, and administers the queues that have been created on it (these are known as local queues). Each local queue also has a name which, along with the name of its queue manager, provides a unique address where messages can be sent.	This part of the animation shows how WebSphere MQ queue managers support a typical business scenario. Three machines have a program on each. Three more machines each have a program, a queue manager, and one or more queues. The queue managers and queues on these machines are: • A queue manager QM_A, and two queues Q1 and Q2 • A queue manager QM_B, and one queue Q3 • A queue manager QM_C, and one queue, Q4

Communicating with local queue managers

Product tour text	Product tour animation
A program communicates with a local queue manager using a supported application programming interface (API), such as the WebSphere MQ Message Queue Interface (MQI), or the Java Message Service (JMS). Messages are constructed by the program and sent to its queue manager using API calls. If the target queue is local, the queue manager can put messages onto it directly. API calls are also used by programs to retrieve messages from their local queues.	This part of the animation shows how a WebSphere MQ API provides the interface between a program and the queue manager on the same machine. A program on a machine that has a queue manager places a message directly onto a queue on that machine, using a WebSphere MQ `MQPUT` call. Another program on the same machine retrieves the message from the queue, using a WebSphere MQ `MQGET` call.

Communicating using WebSphere MQ Clients

Product tour text	Product tour animation
A program that does not have a queue manager on the same machine uses API calls via a local WebSphere MQ Client instead. The client communicates with the remote queue manager over a client connection. This allows the program to interact with the remote queue manager as though it were local.	This part of the animation shows how a business might use WebSphere MQ clients. Three machines, each with a program, an API, and a WebSphere MQ Client (but no queue manager or queues) are the client machines. A machine with two programs, an API, a queue manager, and two queues is the server machine. A client connection is created between each of the client machines and the server machine. A program on one of the client machines sends a message to the server machine.

Product tour text	Product tour animation
	The message is transferred over the client connection to a queue on the server machine. One of the programs on the server machine retrieves the message from the queue. The same program on the server machine places a message on the same queue. The message is transferred over the client connection to the client machine.

Sending messages to remote queues

Product tour text	Product tour animation
For a program to send messages to a remote queue (a queue on a different queue manager), the following must be available: • A message channel (connects the local and remote queue managers) • A transmission queue (used to store messages if the message channel is unavailable) The queue manager puts messages onto the transmission queue, which are then sent to the associated remote queue manager.	This part of the animation shows how a program puts WebSphere MQ messages onto a remote queue. A message channel is created between a queue manager on one machine (the sending machine), and a queue manager on another machine (the receiving machine). A transmission queue is created on the queue manager on the sending machine. A program on the sending machine wants to send a message to the receiving machine. The sending program places a message on the transmission queue. The message is sent over the channel to the local queue on the receiving machine.

Making queue locations transparent

Product tour text	Product tour animation
A program can send messages to a remote queue without needing to know its	This part of the animation shows how a WebSphere MQ queue manager uses remote queue definitions to obtain the

Product tour text	Product tour animation
location, if a remote queue definition has been defined locally. The remote queue definition holds the target queue and transmission queue names. The local queue manager uses this information to determine where to send the message. By using remote queue definitions, message channels and transmission queues, any program can send messages to any queue without needing to know where that queue is located.	location of queues before sending messages. A remote queue definition is created on the sender machine. The definition points to a local queue on the receiving machine. A program on the sending machine wants to send a message to a program on the receiving machine. The queue manager on the sender machine obtains the name of the transmission queue for the required destination, from its remote queue definition. The message is placed on the correct transmission queue, and then transferred over the message channel, to the local queue on the receiving machine. The program on the receiving machine retrieves the message from its local queue. The animation now shows how messages might be sent to other remote queues on the topology, without the sending queue managers having to know explicitly, where the remote queues are physically located.

Interacting with WebSphere MQ

This part of the tour introduces the WebSphere MQ administration interfaces, the messaging interfaces available for programs to use.

Creating and administering WebSphere MQ objects

Product tour text	Product tour animation
WebSphere MQ objects such as queues and channels can be administered by using: WebSphere MQ Explorer WebSphere MQ Script	This part of the animation shows how WebSphere MQ objects such as queues and channels are created and managed. The animation uses a single machine with two programs, two APIs, a queue manager QM_A, and a single queue Q1.

Product tour text	Product tour animation
Commands (MQSC) MQSC is a command line based interface which can be used to issue commands interactively or from scripts. For example, the MQSC command DEFINE (used to define objects such as queues), DISPLAY (used to display object attributes such as the number of messages on a queue) and CLEAR (used to remove messages from a queue).	The animation shows how these objects are represented in the MQ Explorer (the queue manager is highlighted in the navigation tree in the left pane, and the queue is highlighted in the pane on the right). Another machine with a single program and an API appears in the topology. This machine has a queue manager QM_C and two queues, Q4 and Q5. The animation shows how these objects are represented in the MQ Explorer. A command prompt window opens, the command runmqsc QM_A is issued, and the status message Starting WebSphere MQ script commands appears. A sequence of operations in the command prompt window now shows how MQSC commands can be used to define a queue, check the queue current depth (number of messages are on the queue), put a message onto the queue, check the queue current depth, clear the queue, check the queue current depth: The command DEFINE QLOCAL(Q2) CLUSTER(QMCLUS1) (create new queue) appears in the command prompt window. The new queue Q2 appears on the cluster. The status message AMQ8006 WebSphere MQ queue created (queue created) appears in the command prompt window. The command DISPLAY QLOCAL(Q2) CURDEPTH (display current queue depth) appears in the command prompt window. The status message AMQ8409 Display queue details appears in the command prompt window, followed by the message QUEUE(Q2) CURDEPTH(0) (zero messages on queue). A message is now put onto queue Q1. The command DISPLAY QLOCAL(Q2) CURDEPTH (display current queue depth) appears in the command prompt window. The

Product tour text	Product tour animation
	status message AMQ8409 Display queue details appears in the command prompt window, followed by the message QUEUE(Q2) CURDEPTH(1) (1 message on queue). The command CLEAR QLOCAL(Q2) (clear queue), followed by the status message AMQ8022 WebSphere MQ queue cleared appear in the command prompt window. The command DISPLAY QLOCAL(Q2) CURDEPTH (display current queue depth) appears in the command prompt window. The status message AMQ8409 Display queue details appears in the command prompt window, followed by the message QUEUE(Q2) CURDEPTH(0) (zero messages on queue).

WebSphere MQ application development

Product tour text	Product tour animation
Programs use WebSphere MQ application programming interfaces (APIs) to communicate with queue managers. • Procedural languages such as C use the Message Queue Interface (MQI). • Object oriented languages such as Java use WebSphere MQ classes. MQI commands such as MQCONN (to connect with queue managers), MQOPEN (to open queues for messaging), and MQPUT (to send messages), are provided by the MQI. MQI commands are standard across all supported procedural languages,	This part of the animation shows how applications interact with WebSphere MQ. The animation uses a single machine with two programs, two APIs, a queue manager QM_A, and two queues Q1 and Q2. A programming editor window containing a sample of C code opens, and the sample code in the editor window scrolls to reveal examples of how the MQI commands MQCONN (connect to queue manager), MQOPEN (open queue), and MQPUT (put message onto queue) are used in C application messaging. The programming editor window now shows a sample of Visual Basic code which scrolls to reveal how the MQI MQPUT command is used in Visual Basic application messaging.

Product tour text	Product tour animation
including C, COBOL and Visual Basic. Java applications connect to a queue manager by defining an object of class MQQueueManager, open a queue for messaging by calling the accessQueue() method, and receive messages by calling the get() method. C++ applications use similar objects and methods.	The programming editor window now shows a sample of COBOL code which scrolls to reveal how the MQI MQPUT command is used in COBOL application messaging. The programming editor window now shows a sample of Java™ code which scrolls to reveal how the MQI MQPUT command is used in Java application messaging. The programming editor window now shows a sample of C++ code which scrolls to reveal how the MQI MQPUT command is used in C++ application messaging.

Additional Topics

Using Queue Manager Clusters

This part of the tour introduces how queue manager clusters can be used to simplify the administration of complex messaging networks and balance workload between programs.

Simplifying queue administration

Product tour text	Product tour animation
Administration of complex messaging networks can become demanding, particularly when queues are regularly being added or removed. This administrative burden can be reduced by using queue manager clusters. Queue managers in the same cluster can exchange messages without requiring remote queue definitions, transmission queues, or message channels.	This part of the animation shows how a WebSphere MQ queue manager cluster reduces the administration overhead on a messaging network. Three machines each hold a queue manager, multiple transmission queues, and multiple remote queue definitions. The queue managers on this network are connected by multiple message channels. The queue managers on this complex network are not currently clustered. The queue managers are now clustered, and all the transmission queues, remote

Product tour text	Product tour animation
Information about all cluster queues on the cluster are held in a cluster full repository. When a queue manager in the cluster wants to send a message to a cluster queue, it obtains the queue location from the cluster full repository. A cluster channel can then be automatically established. Whenever a cluster queue is added to, or removed from a queue manager in the cluster, the cluster full repository is automatically updated.	queue definitions, and message channels that were previously required are removed, leaving only one or two local queues on each queue manager. The local queues are known as cluster queues, and the messaging network has already become much less complex. A cluster full repository containing information about the location of each queue on the cluster is created on one of the queue managers. The entries in the repository map each cluster queue to its owning queue manager, for example Q1 is on QM_A. A program within the cluster now wants to send a message to another program within the same cluster. The queue manager on the sending machine obtains the name of the queue manager where the target queue is located, from the full cluster repository. A cluster channel is automatically created between the sending queue manager and the receiving queue manager, and the message is sent over the cluster channel, to the cluster queue on receiving queue manager. The message is now retrieved from the local cluster queue by the receiving program. Another queue is added to one of the queue managers within the cluster. The full cluster repository is automatically updated with the queue name and the owning queue manager name. The new queue has now become a cluster queue and can easily be located by other queue managers in the cluster.

Workload balancing between programs

Product tour text	Product tour animation
If the volume of messages being sent to a cluster queue exceeds the processing capabilities of the receiving program(s), the workload can be balanced by defining cluster queues with the same name, on other queue managers in the same cluster. When messages are sent to a cluster queue that has been defined on multiple queue managers, WebSphere MQ automatically balances the workload by distributing the messages equally.	This part of the animation shows how WebSphere MQ allows an increasing amount of messaging traffic to be balanced between the cluster queues on a queue manager cluster. A queue named Q5 is defined on queue manager QM_B and queue manager QM_C, within the same queue manager cluster. The full cluster repository is automatically updated with the names of the cluster queues and the names of their queue managers. These cluster queues are now available to any queue manager within the cluster. Programs on machines outside the cluster now start sending multiple messages destined for queue Q5, to one of the machines within the cluster. As the first message arrives, the cluster queue manager on that machine obtains the locations of the two queues named Q5, from the full cluster repository. Subsequent messages arriving for queue Q5 can now be sent alternately to each Q5 queue, thus balancing the workload.

Publish/Subscribe Messaging

This part of the tour introduces the concept of publish/subscribe and the advantages of using a publish/subscribe messaging environment.

An alternative style of messaging

Product tour text	Product tour animation
This tour has described how messages are sent directly from one application to	This part of this animation demonstrates the publish/subscribe style of messaging.

Product tour text	Product tour animation
another. This style of messaging is known as point-to-point messaging. An alternative style of messaging, where applications communicate indirectly via a broker, is known as publish/subscribe. A publisher sends a message about a topic (for example, stock prices) to the broker. The broker passes the published message to the subscribers, applications who have registered interest in the topic. Publishers and subscribers are unaware of each others' existence.	One program sends a message directly to another program, illustrating the point-to-point style of messaging. A broker is now created and the two programs connect to it rather than each other. The sending program, now called a publisher, sends a message to the broker. The topic of the message is Stocks. The broker recognizes that Stocks is a registered topic of interest of the receiving program, now called the subscriber. The message is passed by the broker to the subscriber.

Distributing messages on demand

Product tour text	Product tour animation
Publish/subscribe messaging provides flexible distribution of messages between applications. • An application can publish information about one or more chosen topics, and a topic can have multiple publishers. • An application can subscribe to one or more chosen topics, and a topic can have many subscribers. The broker can distribute messages from many publishers, on many topics, to many subscribers.	This part of the animation shows how using a publish/subscribe messaging model can increase the flexibility of message distribution. Two more publishers are connected to the broker. Each has one or more topics that they publish messages about. Two more subscribers are connected to the broker, each subscribes to one or more topics of information. Messages are now sent by the publishers to the broker. For each message, the broker recognizes the topic and forwards a copy of the message to each subscriber that is interested.

Multiple brokers

Product tour text	Product tour animation
A publish/subscribe messaging environment can contain multiple brokers, each with their own group of publishers and subscribers. Brokers are arranged in a hierarchy to minimise the number of connections required. When a broker receives a publication on a topic that other brokers have registered interest in, copies are sent via the connection hierarchy to these other brokers.	This part of the quick tour shows how multiple brokers can be arranged in a publish/subscribe messaging environment. Two more brokers are now added to the messaging network. Each has its own publishers and subscribers. `Broker 1` is connected to both `Broker 2` and `Broker 3`. A message is published to `Broker 2`. `Broker 3` passes the message to both of the other two brokers who pass the message to their subscribers that are interested in the message's topic.

WebSphere MQ Publish/Subscribe

Product tour text	Product tour animation
Publish/subscribe messaging is supported by WebSphere MQ, which can provide a simple multiple broker service. Publishers send messages to stream queues, using the APIs described earlier in this tour, or by using JMS as described in the next section. The brokers examine each publications topic, and: • Forward copies of the publication to their subscribers via their local queue • Forward copies to other brokers interested in the topic, via their stream queue, who then forward copies to their	This part of the animation shows how brokers are used in a WebSphere MQ messaging environment. The animation now changes to a WebSphere MQ messaging topology containing clients and queue managers with their local queues, programs and remote queue definitions. Appended to each of the queue managers is a broker, each broker has its own stream queue. A message is sent from a publisher program to its local broker. The broker forwards a copy of the message to a local subscriber program via a local queue. It also passes a copy of the message to the stream queue of a second broker interested in the message topic. The message is then forwarded by the second broker to the subscribing programs that are local to it via the local

Product tour text	Product tour animation
subscribers.	queues.

WebSphere Business Integration Message Broker

Product tour text	Product tour animation
The basic publish/subscribe service provided by WebSphere MQ can be enhanced by using an additional IBM product, WebSphere Business Integration Message Broker. This product provides a more powerful message broker solution driven by business rules. Messages are formed, routed, and transformed according to these rules, which can be defined by an easy-to-use graphical interface. For further information on this product visit ibm.com	There is no animation for this section of the tour.

WebSphere MQ and the Java™ Messaging Service

This part of the tour introduces how WebSphere MQ can use JMS in both point-to-point and publish/subscribe messaging environments.

WebSphere MQ and the Java Message Service

Product tour text	Product tour animation
Java programs can be written to access WebSphere MQ through the open standard Java Message Service (JMS). Programs using JMS do not directly specify queues or queue managers. Instead they use generic objects such as Destination and ConnectionFactory.	This part of the animation shows how WebSphere® MQ interacts with the Java™ Messaging Service. The animation uses a single machine with two programs, two APIs, a queue manager QM_A, and two queues Q1 and Q2. A program editor window opens and a Java code example shows how the

Product tour text	Product tour animation
Mappings to WebSphere MQ objects are defined in a Java Naming Directory Interface (JNDI) namespace, using the command line based JMS Admin tool.	Destination and ConnectionFactory objects are invoked. A command prompt window opens, and the JMSAdmin command starts the JMS Admin tool. A JMS admin DEFINE command is issued to define an entry for the connection factory in the JNDI (Java Naming Directory Interface). This connection factory maps to queue manager QM_A A JMS admin DEFINE command is issued to define an entry for the destination in the JNDI. This destination maps to Q1. The Java code in the program editor window runs, the JMS object definitions are imported from the JNDI and a message is put onto Q1.

JMS and point-to-point messaging

Product tour text	Product tour animation
JMS programs can be used with both point-to-point and publish/subscribe messaging models (for an explanation of these terms, see the Publish Subscribe section). For point-to-point messaging the JNDI namespace holds mappings to queues. JMS programs import this information at runtime to: • Put messages on queues • Get messages from queues	This section of the animation shows how mappings are defined in the JNDI namespace for the point-to-point style of messaging. The Java™ code in the editor window runs again, the JMS objects definitions are imported from the JNDI and a message is put to Q1 on QM_A and retrieved from it. The connection factory and destination entries held in the JNDI namespace are used by WebSphere® MQ to determine directly where messages should be sent to and retrieved from.

JMS and publish/subscribe messaging

Product tour text	Product tour animation
JMS supports publish/subscribe messaging services such as that provided by WebSphere MQ Publish/Subscribe. The JNDI namespace holds mappings to the queue manager and queues each message broker uses, and the topics that publishers and subscribers are interested in. JMS programs import this information at runtime to: • Publish messages to brokers • Receive messages on topics	This section of the animation shows how mappings are defined in the JNDI namespace for the publish/subscribe style of messaging. The diagram now changes to show the publishers, subscribers, brokers and stream queues introduced in the Publish/Subscribe Messaging section of this tour. Additional mappings are defined in the JNDI namespace, these mappings point to the stream queue the broker uses to receive messages and the local queues it uses to publish messages. Mappings are also defined in the JNDI namespace about the topics of information the broker has registered interest in. The Java™ code in the editor window runs again, the JMS object definitions are imported from the JNDI and a message is then put onto the broker's stream queue. The broker forwards copies of the message to a local subscriber and to another broker that has registered interest in the message topic. The other broker then forwards copies of the message to it's local subscribers.

WebSphere MQ with z/OS

This part of the tour introduces the concepts unique to the z/OS platform including shared queues and shared channels.

Messaging between z/OS and other platforms

Product tour text	Product tour animation
WebSphere MQ for z/OS extends the benefits of integration, asynchrony, assured delivery and scalability to z/OS applications. The MQ concepts and objects	This part of the animation shows additional concepts which are unique on the z/OS platform are described in this section. The animation labels two of the three existing system as z/OS systems.

Product tour text	Product tour animation
described previously in this tour are equally applicable to the z/OS environment. This section describes some additional concepts which are unique on the z/OS platform.	Messages are sent between all three systems to show that messaging is unaffected. The animation shows that WebSphere MQ can be used on z/OS systems exactly as previously described in the tour. Messages are sent and retrieved by queue managers on z/OS systems.

Messaging and z/OS sysplexes

Product tour text	Product tour animation
Multiple z/OS systems can be grouped together to form a sysplex. A sysplex uses one or more coupling facilities to provide high-speed caching, list processing, and locking functions. WebSphere MQ for z/OS can group together queue managers within a sysplex to form a queue-sharing group. Queue managers within a queue-sharing group can exchange messages via the coupling facility, reducing the need for message channels.	This part of the animation shows how a z/OS sysplex reduces the administration overhead on a messaging network. The two z/OS queue managers are now grouped in a sysplex. A coupling facility belonging to the sysplex is created between the two z/OS machines. The two z/OS machines in the sysplex are now put in a queue-sharing group. A message is sent from a program on one of the z/OS queue managers to a program on the other z/OS queue manager. Because both queue managers are in the same queue-sharing group, the message is transported via the coupling facility without the need for the message channel between the queue managers. This message channel is removed from the diagram.

Sharing queues between queue managers

Product tour text	Product tour animation
Using MQ for z/OS, local queues can either be defined as private and hosted by a queue manager (as shown in earlier sections of this tour), or shared and hosted by the coupling facility.	This part of the animation shows how queues can be shared by queue managers within a queue-sharing group. In the animation a shared queue definition, SQ_1, is placed in the

Product tour text	Product tour animation
Shared local queues can be accessed and administered by any queue manager within the queue-sharing group. Messages destined for shared local queues can be delivered via any queue manager within the queue-sharing group. Any queue manager within the queue-sharing group can retrieve messages from a shared local queue If shared local queues are used for all inbound messages, the need for private local queues is reduced.	coupling facility. Two messages are sent from a program on queue manager QM_A, outside the sysplex, to the shared queue SQ_1. The remote queue definition for SQ_1 directs the first message to queue manager QM_B, which then directs it onto the shared queue. The remote queue definition for SQ_1 is then changed so that the second message is directed to queue manager QM_C, which then also directs it onto the shared queue. The first message on the shared queue is retrieved by a program on QM_B. The second message on the shared queue is then retrieved by a program on QM_C. The local queues on QM_B and QM_C, and the remote queue definitions for the queues on QM_B and QM_C are now removed as they are no longer needed, leaving the shared queue, SQ1. The messaging network has now become less complex.

Sharing inbound message channels

Product tour text	Product tour animation
Sysplexes can be configured to use a load balancing facility so that inbound connections are allocated to the least busy system. Inbound message channels can be configured to connect to the queue-sharing group using the load balancing facility. Messages destined for shared local queues can then be automatically routed via any queue manager in the queue-sharing group.	This part of the animation shows how a load balancing facility and shared message channels can be used by a queue-sharing group. In the animation a load balancing facility appears within the sysplex, with connections from it to each of the three systems in the animation. In queue manager QM_A, (which is outside the sysplex), a transmission queue QSG1 and a message channel to the load balancing facility are added. The remote queue definition for SQ1 is modified to route messages via this new message

Product tour text	Product tour animation
Using this configuration the need for inbound message channels to specific queue managers is reduced.	channel. Two messages are sent from a program on queue manager QM_A destined for SQ1. The first message is routed to the load balancing facility, which chooses to direct the message to queue manager QM_B, which then directs the message to shared queue SQ1. The second message is also routed to the load balancing facility, which chooses to direct it to queue manager QM_C, which then directs it to shared queue SQ1. Now that the inbound message channels to the specific queue managers within the queue-sharing group are not required, they are removed.

Sharing outbound message channels

Product tour text	Product tour animation
Outbound message channels and their transmission queues, can be configured to be shared, and hosted by the coupling facility. Using this configuration, the need for outbound message channels defined on specific queue managers is reduced. When a message is passed to a shared transmission queue, WebSphere MQ selects the least busy queue manager to establish the message channel.	This part of the animation shows how outbound message channels can be shared by queue managers within a queue-sharing group. A transmission queue now appears in the coupling facility for messages destined for QM_A outside the sysplex. Now that the outbound message channels to the specific queue managers within the queue-sharing group are not required, they are removed. Programs on each of the two queue managers within the queue-sharing group now send a message destined for a remote queue on queue manager QM_A, outside the sysplex. The messages are passed to the shared transmission queue hosted on the coupling facility. For the first message, the coupling facility establishes a message channel to queue manager QM_A via queue manager

Product tour text	Product tour animation
	QM_B. For the second message, the coupling facility establishes a message channel to queue manager QM_A via queue manager QM_C. Both messages are delivered safely to the remote queue.

Using queue-sharing groups to improve availability

Product tour text	Product tour animation
If a queue manager within a queue-sharing group becomes unavailable (for example if an unplanned outage occurs), messaging activity is automatically diverted to the remaining queue managers.	This part of the animation shows how queue-sharing groups can improve the availability of a messaging network. Two messages are put from a program on QM_A to the shared queue hosted by the coupling facility. They are delivered via the shared channel to the load balancing facility, then to the shared queue via QM_B. QM_B becomes unavailable, more messages are then sent from QM_A. The load balancing facility ensures that the message is still delivered to the shared queue by directing it via QM_C. QM_B becomes available again and applications on both QM_B and QM_C retrieve messages from the shared queue.

Using queue-sharing groups to increase performance

Product tour text	Product tour animation
When increased performance is needed, queue-sharing groups make adding extra processing power easy. • Because shared queues can be accessed from any queue manager in the group, additional applications can be started on any z/OS system that has spare	This part of the animation shows how queue-sharing groups are an easy way of increasing processing power. Many messages are sent to the shared queue, via the load balancing facility. These messages are retrieved by programs on QM_B and QM_C for processing, but a backlog of messages builds up on the shared queue. Additional programs then appear on both QM_B and QM_C. All these

Product tour text	Product tour animation
capacity. • When new queue managers are added to the queue-sharing group, they will automatically inherit the shared objects defined for the queue-sharing group. Workload will automatically be balanced across the queue-sharing group.	applications now retrieve messages from the shared queue increasing the speed at which all the messages are dealt with, and clearing the backlog. Two additional queue managers are added to the queue-sharing group. The remote queue definitions that were defined on the existing queue managers within the queue-sharing group automatically appear on the new queue managers. Many messages are sent from QM_A to the shared queue. WebSphere MQ automatically balances the workload across all four queue managers and applications on each of these queue managers retrieve the messages from the shared queue.