



WebSphere software

Providing a messaging backbone for SOA connectivity.

Contents

- 2 Executive summary**
- 4 Messaging backbone:
A first step to SOA**
- 8 Messaging concepts**
- 16 Meeting enterprise demands**
- 23 Connecting virtually anything**
- 24 Interfaces and standards**
- 26 Messaging for Web services**
- 27 Transferring files reliably**
- 29 Foundation for your ESB**
- 31 Summary**
- 31 For more information**

Executive summary

Today, you likely have a disparate, widely distributed, increasingly complex, enterprise-computing infrastructure. One that's made of different kinds of systems – located in and managed through different departments and geographic locations. A flexible, robust messaging backbone can enable you to quickly connect new applications and use existing ones cost-effectively, while minimizing risks to business data. And you can integrate these applications across your entire organization and with those of key trading partners, suppliers and customers.

To stay competitive, you can't continue to rely on manual processes to manage information that's distributed through a wide range of disconnected systems. It's expensive to maintain, more prone to human error and doesn't accommodate future growth. A strong messaging backbone can provide you with the security-rich foundation to provide your goods and services over the Web, facilitate more-effective interactions, streamline critical processes and enhance productivity across your value chain. So the flow of transactions, information and ideas can ripple immediately through your enterprise – and beyond.

As many organizations look to service oriented architecture (SOA) to increase the flexibility of IT, it becomes increasingly important to be able to connect the new (service-oriented applications and assets) and the now (existing IT assets that are the lifeblood of your organization). As SOA enables greater interoperability to IT assets and increases the ability to dynamically interconnect these, the need to provide a robust, reliable backbone for these interactions is ever-more acute, preserving a bet-the-business quality of robustness whenever services and non-service-oriented assets interact.

IBM WebSphere® MQ provides a flexible, robust messaging backbone that enables integration for SOA and for your existing IT assets. WebSphere MQ provides the industry's leading messaging backbone with assured, once-only delivery of data across a wide range of operating systems. Across industries, from banks and telecommunications companies to government agencies, IT departments have reaped the benefits of using a common technology to connect disparate systems.

In the last decade, the software industry has converged on certain standards: Web services for service discovery and invocation, Java™ 2 Platform, Enterprise Edition (J2EE) as an enterprise programming model or XML as a canonical data format. Messaging technology – particularly WebSphere MQ – complements these standards by playing an important role in building a strong, open, IT infrastructure.

This white paper examines the value of implementing a messaging backbone instead of building a complex web of custom coding to integrate your business processes. It explores WebSphere MQ, and the key concepts that comprise the product, in more detail. This white paper also discusses some of the standards that currently shape the industry – and why the messaging backbone is still a critical part of the picture, especially in an SOA. Finally, it provides insight about the overall industry landscape, so you can more clearly understand WebSphere MQ, how it fits into today's IT industry and how it can work for you.

Messaging backbone: A first step to SOA

Today's business goals are driving IT priorities. Competitive pressure and market forces drive business leaders to radically change their organizations. The goal is strategic flexibility through innovation. And SOA enables IT to respond to change faster. Typically, a major barrier to achieving flexible IT is a veritable rat's nest of applications, developed in isolated silos, often aligned with departments. Significant portions of IT budgets are expended simply maintaining the connectivity, and the cost of extending it tends to increase as it grows.

The messaging backbone provides a first step to SOA (see Figure 1). It bridges the gap between new service-oriented assets and existing core assets, and provides the transport foundation for an enterprise service bus (ESB). It also frees applications from the connectivity logic needed to determine how each application communicates with the others.

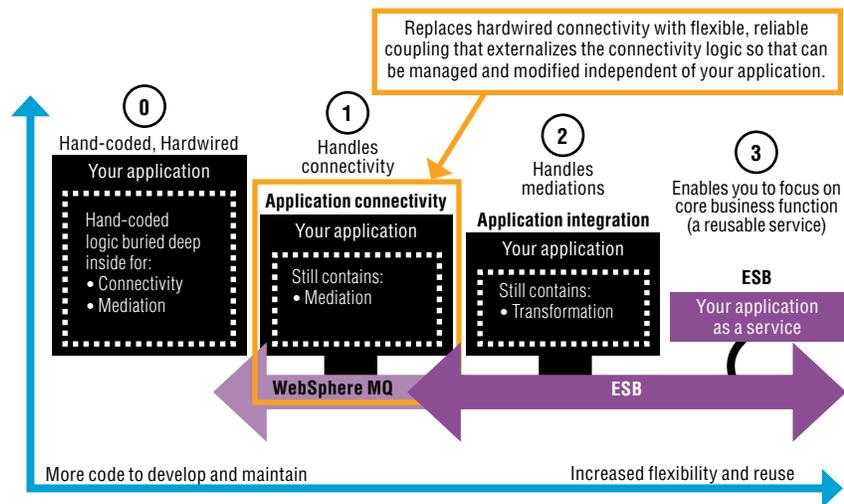


Figure 1. WebSphere MQ provides a first step to SOA.

A messaging backbone enables you to reuse what you have already. Because you do not have to rip and replace applications, you can increase the return on investment (ROI) of the applications you have. Because the WebSphere MQ messaging backbone is supported on virtually any commercial IT system, you don't have to replace your choice of hardware and operating system, either. Its choice of interfaces in a range of popular programming languages means that you can use the skills that you have available.

Business processes can be made more reliable with a messaging backbone helping to assure that the data is delivered and that transactions are preserved, even when these processes touch multiple IT systems spanning across your organization. Reducing the risk of IT systems losing integrity can help to prepare you for the challenges of regulatory compliance, where financial reports must be based on demonstrably accurate business data.

To further understand the benefits of messaging software, consider the alternatives. Most enterprises have several systems, applications or islands of automation that stand alone – often on different operating systems. Data usually resides in more than one place, causing duplication and synchronization issues. Employees manually enter data into several different systems. And if you develop or purchase a new application – or a merger or acquisition occurs – the situation becomes even more complex. To solve the problem, you need to connect your applications together, allowing them to share information and unlock the data distributed across your enterprise.

To achieve this level of connectivity, you might decide to write code, typically embedded into your application, to communicate with other systems. This means your developers must write connectivity logic – and grapple with the nuts and bolts of a particularly difficult area of software development. It could involve issues like the handling of TCP/IP sockets, which can vary depending on the operating system and programming language used. And it can require a wide array of specialized skill sets within your development team. The code must be able to handle situations where the network fails, or where the receiving applications are unavailable. Because each piece of connectivity logic is specific to the applications it connects, you limit the possibility of reuse and make it more difficult to add applications as your business needs change. The likely result is your IT staff writes, owns, extends and maintains a large quantity of complex, unwieldy connectivity logic.

You can avoid this situation by using a software product designed to handle all these connectivity issues for you – a messaging backbone. Rather than riddling your applications with connectivity logic, you let your applications talk to the backbone – through a simple, common application programming interface (API) – to deliver the data to other applications.

When you implement a messaging backbone, you can adopt industry-standard programming models and make them available on a selection of operating systems. An effective messaging backbone should be ubiquitous to maximize reuse of skills and code across your enterprise. Your application developers can simply concentrate on writing business logic without having to maintain large quantities of connectivity code.

A vital aspect of the messaging backbone is assured delivery. You have to be able to control the required quality of service on data delivery. For example, it might be acceptable to send noncritical data in a fire-and-forget model, where you're aware that the data might be lost, given certain failure scenarios. However, for critical business information, like a banking transaction, you want assured once-and-once-only delivery. When the application sends this critical data to the messaging layer, the processing should continue. If messaging events – like sending and receiving data – can also act as part of a transaction, it helps ensure that actions, such as database updates, can occur in the same unit of work as messaging operations, with coordinated commit or rollback.

Another central concept of the messaging backbone is time-independent – or asynchronous – processing. This concept means that applications don't rely on each other's availability, or the availability of the network, to send data. In a purely synchronous model, in the case of a network failure, your applications would require sophisticated retry logic and could be blocked waiting for the network to recover. Asynchronous messaging is best viewed as a delivery model, not as the opposite to synchronous messaging. Asynchronous messaging simply decouples applications from each other and from the network. It operates on a fastest-possible delivery model. If you wish to send data from application A to application B, and the network is available, the data will be delivered almost immediately. However, if the network or the receiving application is unavailable, the sending application isn't necessarily affected. The messaging backbone temporarily stores the data if required.

Because your messaging backbone handles your valuable business data, you want to ensure that the infrastructure you choose is reliable. You have to make sure data doesn't get irrecoverably lost. Your messaging software must operate in a failover model. It needs built-in capabilities to help ensure high availability. You might also want features to help with workload balancing, so you can be sure your messaging backbone can grow and adapt as your business requirements change.

Messaging concepts

The fundamental components of a messaging backbone are its messages and queues. This section briefly discusses these and some other key components. For more in-depth information, refer to the WebSphere MQ product manuals.

Messages

WebSphere MQ enables applications and services to communicate by sending messages rather than calling each other directly. Messages are simply strings of bytes, containing the data you wish to deliver from one application to another. A message has two parts: a header that describes the message, and the data itself, which could be any kind of data, such as XML, binary data or a bit stream. The message header identifies the message with a unique message ID and contains other fields such as message type, information about its origin, priority, expiration time, the queue to send any replies to and so on.

Each WebSphere MQ message can be up to 100 MB in size. Larger messages and files can be transported by segmenting these into smaller chunks.

WebSphere MQ can segment large messages automatically, and receiving applications can choose to retrieve the larger messages when these have been recombined by WebSphere MQ, or receive each segment. Alternatively, programming controls enable messages to be split into segments based on logical boundaries or the size of the buffer available to the receiving application. WebSphere MQ helps ensure that the order of the segments is preserved.

Similarly, messaging traffic can be reduced when many very small messages are being sent to the same destination by grouping these together into larger WebSphere MQ messages. When the composite message arrives at its destination, WebSphere MQ disassembles the message and can again preserve the order in which these constituent messages are delivered.

Distribution lists provide another way to reduce messaging traffic. When the same message is being sent to multiple queues owned by the same queue manager, only one single copy of that message needs to be sent to the queue manager. WebSphere MQ uses a distribution list to determine which of the queues it owns needs a copy of that message and acts accordingly. These distribution lists can be updated at any time, as required.

Message persistence

WebSphere MQ can handle messages in both a persistent and nonpersistent manner. All persistent messages are logged by WebSphere MQ, which synchronously writes these messages to disk or other nonvolatile storage at the same time as sending them. This capability enables the delivery and recovery of messages even if the applications, networks or WebSphere MQ server goes down at any stage of transmission.

Nonpersistent messages are not logged in this way; however, these messages are also delivered no more than once, avoiding the problems associated with duplicate messages arriving and causing transactions to partially run more than once.

Semipersistent messaging is also possible, in cases where you need a more-granular trade-off between the robustness of delivery and the throughput of the messaging system. Performance of semipersistent messaging is typically greater than persistent messages because logs of the message are not taken at all the steps during delivery where failures could possibly occur. In this instance, semipersistent message logging occurs asynchronously – typically after the message is actually dispatched. This capability can increase the speed of throughput because it avoids waiting for disk input/output (I/O) operation to complete before sending, but introduces some risk that the messages and queue manager cannot be completely recovered if a failure occurs in the window before the log is updated.



Types of queues

The most common types of queues include:

- *Local queues*, the queues that actually reside on a particular machine
- *Alias queues*, alternate names for referring to local queues
- *Remote queues*, references or handles to queues that reside on other machines
- *Transmission queues*, special queues that WebSphere MQ uses internally to move messages across a network (where messages reside if the network is unavailable)
- *Dynamic queues*, local queues that are created on demand
- *Dead-letter queues*, queues that store messages that cannot be delivered (such as when a destination queue is full and cannot be enlarged)
- *Repository queues*, queues that hold clustering information

It is important to understand the level of persistence needed by each application or service and select the appropriate persistence service to help optimize the messaging backbone and provide the level of recoverability needed by the business. By default, WebSphere MQ uses persistent messaging. In addition, to make accurate performance benchmark comparisons of message backbones, you should use equivalent persistence levels.

Queues

WebSphere MQ enables applications and services to communicate asynchronously, without each having to be available at the same time. This capability is possible because of *queues*, which are data structures used to store messages. When messages move across the WebSphere MQ backbone, queues are used, even momentarily, to store messages so that these can be retrieved when the recipient is available. Typically, when sending and receiving applications are all available, messages sent through these queues arrive at their destination in near real time. However, when applications or the network are unavailable or busy, the queue is able to hold the messages until they can be received and processed. This asynchronous messaging model is a powerful way of loosely coupling applications and services so that their communication is time-independent.

As a result, the messaging backbone is able to shield applications and services from interruptions and failures, as well as enabling them to continue doing meaningful work instead of being locked in conversations. In addition, you can also use WebSphere MQ to start applications – through the use of application-triggering – when sufficient messages arrive for it to process. And WebSphere MQ can preserve the order of messages, delivering them to applications in the same order as they were dispatched. First-In-First-Out (FIFO) is the default.

Queue managers

At the heart of a messaging backbone are its *queue managers*, which provide the messaging services and manage objects like queues and channels. Queue managers use transmission queues to move messages to remote queues owned by other queue managers. They provide triggering services, enabling applications to be started when sufficient messages arrive for processing. They also handle the conversion of character sets within messages between platforms. On distributed systems, WebSphere MQ queue managers can act as transaction coordinators, using two-phase commit to preserve the transactionality of operations to databases and queues.

Queue managers handle the recovery, persistence and assured delivery of messages. In persistent or semipersistent messaging, the queue manager logs message data to disk. WebSphere MQ queue managers are often backed up in high-availability environments.

Channels

WebSphere MQ uses channels to connect its queue managers, and to connect WebSphere MQ clients to them. Channels are logical communication links. A message channel is defined to connect one queue manager to another – referred to as server-to-server communication. These channels are unidirectional, and are often defined in pairs. At either end of these message channels, sender and receiver agents – or movers – coordinate the communications link. WebSphere MQ clients also use channels to connect to the queue managers of WebSphere MQ servers, although a different kind of channel is used in this case, because clients do not have queue managers. Client channels are bidirectional. Some channels can be defined automatically by WebSphere MQ. Queue managers contain a message channel agent (MCA) that is responsible for channels.

Clients

WebSphere MQ supports a range of clients to enable applications and services to connect into its messaging backbone. WebSphere MQ clients connect into remote queue managers. WebSphere MQ clients for a range of platforms are included with the product and can also be downloaded at no charge. Most clients are fully supported.

The main difference between clients and servers is that clients do not have queue managers, whereas servers can. This difference means that clients do not provide a local queue to store messages and so do not support asynchronous messaging between the client and the servers it connects with. Clients connect using dedicated, bidirectional channels, and can only make connections to WebSphere MQ servers when the network is available. The application or service can retry the connection in this case or try another route to reach messaging servers.

Whether to run a WebSphere MQ server with its queue manager or a client depends on the quality of delivery needed by the applications and services local to a particular machine. If reliable delivery of messages is needed from that machine, then a client is unlikely to meet your needs.

WebSphere MQ also supports a special client called an *extended transactional client*. As with other clients, it does not provide a queue manager. Extended transactional clients enable applications to perform several tasks – including putting messages to a server and updating the resources of another resource manager, such as a transaction coordinator, database or application server – all within a single unit of work (UOW) transaction. This capability can enhance the transactionality of a connection, while removing the need to deploy a local queue manager. Extended transactional clients are licensed as part of WebSphere MQ. When installing WebSphere MQ, you can choose to deploy either a server or an extended transactional client.

To learn more about clients, visit ibm.com/websphermq/clients.

Messaging topologies

Topologies refer to the shape of the messaging backbone – how its messaging nodes are connected. Different messaging styles and topologies have certain strengths and can be combined within your messaging backbone. Figure 2 shows examples of different messaging topologies.

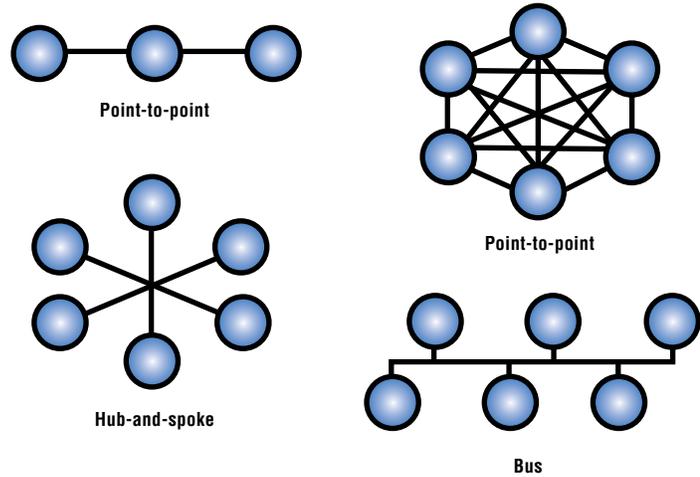


Figure 2. Messaging backbone topologies

The point-to-point topology is a simple connection between two messaging nodes or applications. The logic defining where messages are sent resides either in the originating application, or in the messaging artifacts defined locally to this application. If the target application moves to a new location, the logic at the source application must also be updated so that messages still reach it. This capability is useful for simple scenarios where a few applications are connected. However, to connect n applications to each other, you must define $n(n-1)/2$ connections in your messaging middleware – to connect each point to each other point. For example, for two applications this means making only one connection; for five applications, 10 connections are needed; for 10 applications, 45 connections are needed. To add one more application now requires an additional 10 new connections to be defined and likely requires changes to most of the existing connections. This combinatorial explosion of configuration information, with the need to make many updates for each changing application, means that it is not ideal for larger deployments.

In a hub-and-spoke topology, each application connects into a centralized node in the messaging backbone. This helps to reduce the number of connections needed for a given number of applications, because the number of connections is only as many as the applications being connected. So in the example of 10 applications, only 10 connections to the messaging backbone need to be defined. Adding another application means defining only one extra connection – only one-tenth of the work of a point-to-point topology. Logic determining where messages are sent can be centralized within the hub of messaging nodes. Although this topology might require more planning to set up initially, it can reduce the reconfiguration required when applications are added or changed. Single points of failure can be eliminated by clustering several messaging nodes within the central hub.

In a bus topology, as with a hub-and-spoke, the number of connections needed between a given number of applications is also only as many as the applications being connected. Fundamentally, this topology is equivalent to hub and spoke, although the bus topology places more emphasis on distributing the connectivity logic across the backbone, helping to reduce the single point of failure often thought to be associated with the hub-and-spoke topology.

Often, hub-and-spoke topologies are associated with “traditional” messaging, and buses are positioned as a new approach. This comparison depends largely on how one chooses to define or implement these topologies. Ideally a messaging backbone should support a mix of topologies so that the optimal approach can be used.

Messaging styles

You can use a variety of approaches to set up a messaging backbone that can optimize performance or simplify ongoing configuration (see Figure 3).

Fire-and-forget is a style of messaging where the sending application does not require a reply or confirmation from the receiving application or applications. Such messages are sometimes called *datagrams*. Request-response is a style of messaging where applications request messages specifying where replies need to be sent. Applications sending requests can choose to wait for replies or continue processing.



Examples of publish-and-subscribe topics

Topics can be simple strings:
"New Sale", "New Customer",
"Credit Rejected",...

Topics can be organized into hierarchies (such as organizing sales activities by location and industry for each customer):
"North-East/New York/Retail/ACME Corp /New Sale"

Richer subscriptions can be described using "*" wildcards (for example, get messages whenever the credit for retailers is rejected):
"***/Retail*/Credit Rejected"

(or get messages whenever there are new customers in New York):
"North-East/New York/*New Customer"

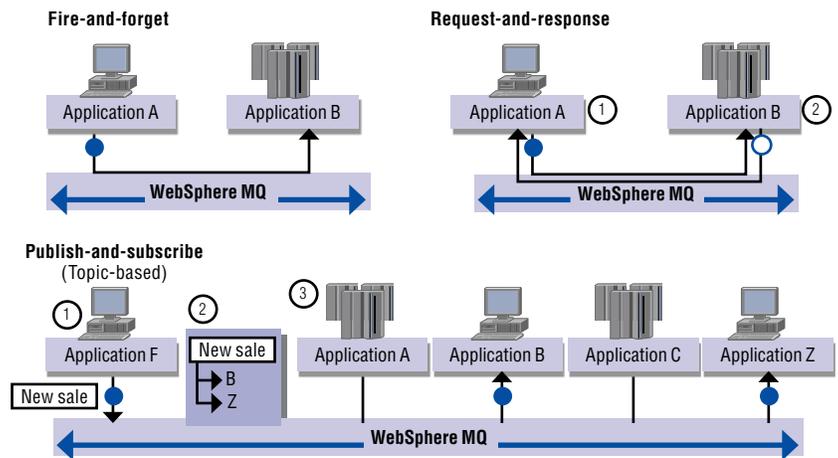


Figure 3. Messaging styles

Publish-and-subscribe provides an event-driven style of messaging that enables the messaging backbone to dynamically determine where messages should be delivered. This approach relieves applications and services of the burden of containing up-to-date information about which applications and services need to receive messages and precisely where these are currently located. It can be useful in progressing toward an SOA, and is especially valuable when you are more likely to move or replace applications and services.

In the publish-and-subscribe model, messages are tagged with keywords or topics – strings that represent a subject for the message. These keyword topics can be organized into hierarchies to enable more-complex classification. Applications and services define the messages they need to receive by logging a subscription with the messaging backbone, using topics to describe their information space, and optionally using wildcards to define richer subscriptions.

WebSphere Message Broker, IBM’s advanced ESB, takes topic-based publish-and-subscribe messaging even further, enabling messages to be routed based on the content of messages and more-sophisticated message-routing definitions. You can apply statistical- and causal-analysis tools to streams of messages so that patterns can be detected that signify even more-complex events and situations.

Meeting enterprise demands

The messaging backbone plays a crucial role in an IT infrastructure as the conduit for linking assets and services together. This backbone must satisfy the high demands of business today to be entrusted as the courier of one of its most valuable resources – its business data. WebSphere MQ has the heritage and credentials to be an excellent choice as the messaging backbone for all sizes of organization, across all industries.

Reliability

The business costs, implications and penalties are steep when critical business data is lost or when the integrity of data and applications is compromised. Lost or duplicated business transactions can severely disrupt business processes, causing a chain reaction of expensive after-effects that might confuse supply-chain interactions and erode customer satisfaction. Regulatory-compliance initiatives such as the U.S. Sarbanes-Oxley Act (SOX) impose heavy penalties on businesses that cannot demonstrate that their financial reports are accurate – an unattainable goal if business transactions are at risk of being lost or duplicated. An organization's messaging backbone must be robust, and able to ensure that business transactions are preserved regardless of failures in hardware, software and the network.

WebSphere MQ guarantees the delivery of messages transported over its backbone and helps ensure that these messages are not duplicated during transmission. It automatically manages its reliable messaging by using receipts to confirm delivery and resends messages as needed so that these operations are invisible to application programmers. By using queues to store the messages, reliable delivery is assured even when the network, hardware and receiving applications are not available.

WebSphere MQ employs the two-phase-commit protocol as a mechanism for preserving the integrity of IT systems when transferring messages over its backbone as part of transactions – where the entire exchange must complete and partial completion or failure leaves these systems in an unreconciled state. It can coordinate transactions that involve messages moving right across its backbone, not just between one server and another, enabling transactionality to be preserved from end to end.



WebSphere MQ: Proven and trusted

- More than 10 000 customers worldwide use WebSphere MQ.
- WebSphere MQ moves more than 10 billion messages every day, supporting more than US\$1 quadrillion (US\$1 000 000 000 000 000) worth of business transactions.
- WebSphere MQ has won multiple awards including the prestigious Royal Academy of Engineering MacRobert Award in 2004.
- WebSphere MQ provides the underlying backbone for IBM's advanced ESB, WebSphere Message Broker, and is fully supported by all IBM ESB offerings.
- More than 800 IBM Business Partners worldwide support WebSphere MQ with software, solutions and services.

High availability

In today's 24x7 world, the business impact of applications, networks and hardware failing can be severe and far-reaching. Similarly, the messaging backbone that connects these applications and services also needs to meet the enterprise's needs for high availability. High availability is the most fundamental part of a strategy to maximize the resilience of an IT environment and requires that this environment be able to rapidly and completely recover from outages – especially unplanned ones.

High availability is achieved by eliminating single points of failure in the backbone by providing backup or redundant systems that can take over should failures occur. It is also accomplished by helping to ensure that whatever data is being transported, and the state of the messaging system itself, is logged up to date and can be used to restore the data and the messaging system following a failure.

Replication-based techniques are often promoted as the simple, straightforward approach to increasing the availability of messaging systems, requiring only software and no specific hardware. However, these approaches are not recommended. Asynchronous replication puts messages at risk of being duplicated or lost, and synchronous replication requires real-time replication of all messages to work around this shortcoming and can result in significant performance degradation.

To avoid these issues, WebSphere MQ uses a persistent messaging approach that logs its messages to disk synchronously with message transmission, along with the state of its queue managers, so that it can reconstruct messages and recover to a consistent state following a failure. Synchronously logging messages to disk at the same time they are dispatched is very important; otherwise you risk being unable to recover a failure to a coherent state before or after the message is sent. WebSphere MQ can take steps to roll back messages that were in transit and commit transfers that had completed to help preserve the integrity of the messages and the applications exchanging them.

WebSphere MQ uses platform-specific high-availability facilities such as High Availability Cluster Multi-Processing (IBM HACMP™) on the IBM AIX® platform and Automatic Restart Manager (ARM) on the IBM z/OS® platform. On IBM System z™ running on the z/OS operating system, WebSphere MQ provides shared queues that can be accessed by different queue managers so that should any fail, another queue manager can automatically access the shared messages.

Clustering

The clustering features of WebSphere MQ enable messages to be redirected around parts of the messaging backbone that have become isolated by network failure or otherwise unavailable without the applications.

Clustering can help reduce administration of the messaging backbone. It enables many WebSphere MQ queue managers to be administered together as a group. Configuring queues in a cluster with the same name enables WebSphere MQ to determine how the messaging workload should be shared between them. This clustering can be based on the workload-balancing algorithm supplied or your own custom algorithm. Alternatively, individual queue managers can be targeted within a cluster by giving them unique names and referencing them when directing messages to their cluster. By balancing the workload, clustering enables you to improve the performance of a messaging backbone, and enable more messaging engines to be added as needed.

Security

In today's business climate, the security of valuable and confidential information and IT systems is a major concern. Making IT systems as secure as possible is an holistic challenge that needs to address areas such as encryption, authentication, authorization, nonrepudiation and privacy.

WebSphere MQ provides built-in security features to help protect data moving across its messaging backbone. It supports the industry-standard Secure Sockets Layer (SSL) for strong authentication of message channels before messages are exchanged to help prevent malicious attacks to the backbone. SSL can also provide bulk data encryption similar to virtual private networks (VPNs).

IBM WebSphere MQ Extended Security Edition adds even more security capabilities to the messaging backbone, taking advantage of IBM Tivoli® technologies to provide individual, message-level security encapsulation. It extends protection beyond the messaging channels to the application layer. It also provides granular audit records down to the message level and can be retrofitted to an existing WebSphere MQ network without the need to modify it. Security is configured using policies that can be administered remotely using a Web browser interface, helping to avoid the need to write security-specific code for each application.

Scalability and performance

As business requirements grow and as more applications and services use messaging to communicate, the messaging backbone needs to be able to scale to support increasing volumes of messaging traffic and enable you to take full advantage of the power of your IT infrastructure.

WebSphere MQ takes advantage of multiprocessor and multicore machines so that you can scale your messaging backbone by using parallel processing to accelerate messaging. This approach can significantly improve point-to-point messaging performance and scalability over single-threaded alternatives. A single instance of WebSphere MQ can run across multiple processors and cores, so there is no need to configure routing or load balancing between multiple instances of a server. It requires practically no tuning or configuration to run across multiple processors or cores, and does not require multiple message logs. To help ensure your messaging backbone can grow with your business, insist on performance comparisons where a range of processors are used. Another useful approach to scaling a messaging backbone is clustering, which is described previously in this white paper.

When comparing the performance of messaging servers, it is important to use the same quality of service for message delivery – such as persistence or nonpersistence. Performance reports for WebSphere MQ provide throughput analysis, capacity planning and tuning information specific to each platform and are published as IBM SupportPacs™ at ibm.com/websphermq/support.

Configuration

Your organization's IT infrastructure is as unique as a fingerprint and ever-changing to take advantage of new technologies and IT assets, and to respond to business pressures and opportunities. It is vital that the messaging backbone enables these changes rather than inhibits them. To reduce the time, risk and cost of configuration, it is important that the whole messaging backbone is able to be centrally configured and administered, despite being a widespread, distributed infrastructure.

WebSphere MQ enables its entire messaging backbone to be remotely configured from a single console, called *WebSphere MQ Explorer* (see Figure 4). WebSphere MQ, Version 6.0 introduced this new configuration tool, which is based on open-source Eclipse Workbench technology. The Eclipse framework is common across IBM software products, so that WebSphere MQ Explorer can be combined with the tools of other products, such as WebSphere Message Broker, to provide a single integrated console. This graphical tool enables you to explore and configure all WebSphere MQ objects and resources, including Java Message Service (JMS), and publish and subscribe. Because it is based on Eclipse technology, WebSphere MQ Explorer is highly customizable and fully extensible. You can add new tools as plug-ins to WebSphere MQ Explorer to add new features in a way that is integrated into the console. Documentation shipped with the WebSphere MQ Explorer provides the interfaces for plug-ins, together with examples of how to develop them so that IBM Business Partners and users can join IBM in augmenting its capabilities.

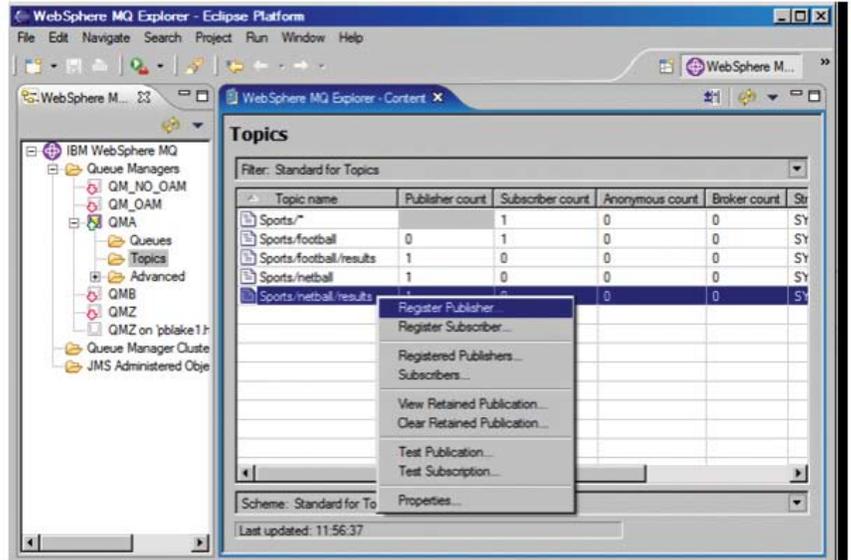


Figure 4. WebSphere MQ Explorer enables you to explore, test and configure your entire messaging backbone remotely.

WebSphere MQ Explorer runs on Microsoft® Windows® and Linux® x86 machines. It does not need to be deployed with a WebSphere MQ client or server, and you can install as many copies as required. You also don't have to directly connect to a queue manager to explore or configure. You can configure queue managers remotely through intermediate queue managers. The WebSphere MQ Explorer can remotely connect to queue managers on any supported platform – enabling your entire messaging backbone to be viewed, explored and altered from the console. You can even remotely configure WebSphere MQ for z/OS running on the System z platform, provided it is running the Version 6.0 (or later) release, because of the added support for the programmable command format (PCF) administration messages that WebSphere MQ Explorer uses.



WebSphere MQ supports more than 80 platform configurations, covering virtually any commercial IT system, including:

- AIX
- IBM System i™ (IBM eServer™ iSeries™ and IBM OS/400®)
- IBM z/OS (IBM eServer zSeries® and IBM OS/390®)
- HP-UX
- HP NonStop Server and OpenVMS
- Linux on Intel®
- Linux on System z
- Sun Solaris Operating Environment
- Microsoft Windows XP and 2000

For the latest support details, visit ibm.com/websphermq/requirements.

You can customize views of the messaging backbone, for example, using filters to show queues or other resources that match certain criteria such as the number of messages in a queue or its name. You can adjust the refresh rates of these filtered views to update them at a machine or queue-manager level. You can also compare attributes, for example, to see whether two queues have the same characteristics.

To prevent unauthorized changes, the WebSphere MQ Explorer uses SSL security. It provides graphical tools to manage authority and access based on the Object Authority Manager (OAM) to help make governance control easier. For example, you can use WebSphere MQ Explorer to show at a glance all the users or groups that have permissions to certain queues and objects. Figure 5 shows this feature.

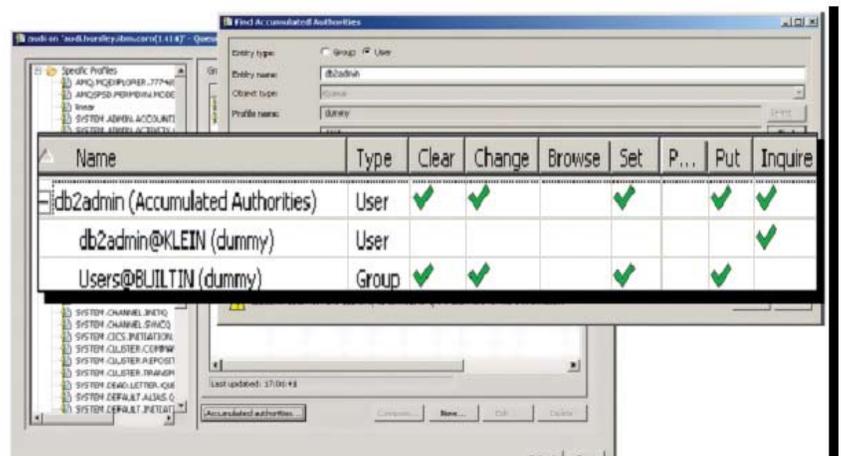


Figure 5. WebSphere MQ Explorer enables you to control access to your messaging backbone.

New problem-diagnostic tools have recently been added to enable you to run tests against your messaging backbone to discover errors and potential problems with the configuration. With one click, this tool searches for problems and provides advice on solutions and improvements. User-defined checks can be added to its suite of tests.



A simple interface

MQI provides 13 simple commands that are consistent across supported platforms:

- MQCONN connects to the queue manager named as a parameter.
- MQCONNX connects using fast-path bindings for trusted applications.
- MQDISC disconnects from the queue manager.
- MQOPEN opens a message queue on the queue manager.
- MQCLOSE closes the message queue.
- MQPUT puts a message onto the queue.
- MQPUT1 is equivalent to the sequence MQOPEN, MQPUT, MQCLOSE.
- MQGET retrieves a message from the message queue.
- MQINQ inquires about the properties of a queue.
- MQSET sets the properties of a queue.
- MQBEGIN begins a transaction or UOW.
- MQBACK rolls back a transaction before it is completed.
- MQCMIT commits a transaction, ending the UOW.

Connecting virtually anything

A messaging backbone is only as valuable as the range of applications and services it can connect. Even if certain platforms and environments are not part of your IT domain today, you can't afford to risk leaving applications stranded as a result of a cross-department initiative, or merger and acquisition, or be prevented from using a new technology or platform. WebSphere MQ is designed to be able to connect virtually any commercial IT system, from the latest technologies to those core systems your organization depends upon – even less-common platforms.

WebSphere MQ provides consistent functionality across its range of supported platforms. WebSphere MQ for z/OS is built natively for z/OS, yet is designed consistently with the distributed version. Therefore, it can take specific advantage of the z/OS environment to offer unique capabilities that make it a powerhouse for the messaging backbone. WebSphere MQ for z/OS is not an adaptation of the distributed product, but a unique code-base designed from the ground up to take full advantage of the z/OS platform and fit tightly within the mainframe environment.

WebSphere MQ for z/OS takes advantage of IBM Parallel Sysplex® technology, increasing availability, capacity and performance for persistent and nonpersistent messages, by enabling multiple queue managers to access the same queue. WebSphere MQ for z/OS also provides workload balancing based on a pull model that enables very high-availability messaging on the z/OS platform.

In addition, WebSphere MQ for z/OS provides tight integration with IBM CICS® and IBM IMS™ using the IBM MQSeries®-CICS Dynamic Program Link (DPL) Bridge, the IBM MQSeries-CICS 3270 Bridge and IBM MQSeries-IMS Bridge.

Interfaces and standards

A messaging backbone needs to tie into what you have today, enable you to make the most of the skills at your disposal, and support the decisions you make in the future. WebSphere MQ has evolved to provide a variety of ways to connect to its messaging backbone.

MQI

The message-queuing interface (MQI) is the original programming interface for WebSphere MQ. It provides a simple, small set of verbs that are consistent across platforms. Depending on the operating system you are running, you can use C, C++, Java, PL/I, COBOL, Visual Basic, ActiveX/COM, Assembler, Report Program Generator (RPC) or Typed Assembly Language (TAL) to program the MQI. Support for other languages, like Perl, are available as downloads.

MQI provides structures (groups of fields) that you can use to supply input to, and get output from, calls. It also provides a large set of named constants to help you supply options in the parameters of the calls. Data-definition files supply the definitions of the calls, structures and named constants for each of the supported programming languages. Default values are set within the MQI calls.

JMS

JMS is an industry-standard programming interface for messaging based on J2EE. It performs the same role as the MQI but is standardized and increasingly popular with Java developers. J2EE technology-compliant application servers, such as IBM WebSphere Application Server, provide the JMS interface and can also provide an implementation of messaging services. JMS is also supported by many stand-alone messaging products, including WebSphere MQ. J2EE applications servers can use WebSphere MQ as their JMS provider, regardless of whether one is already supplied. It is a misconception that JMS technology-compliant products must be developed entirely in Java.

WebSphere MQ supports the latest version of the JMS standard, Version 1.1. Like the MQI described previously, JMS is an interface for programming. Because JMS is an industry standard, applications programmed to JMS can be ported between messaging products that support it without modification, and developers need only learn one set of commands to use a range of vendor products. In practice, vendors often provide extensions to the JMS API. Because JMS standardizes only the interface, it does not standardize the technology used for the underlying data delivery. As a result, the wire protocol used by JMS technology-compliant products is proprietary and vendor-specific. This means that products compliant with JMS cannot interoperate.

Although the JMS standard does not deliver interoperability between messaging providers, IBM provides the capabilities needed to bring JMS implementations together to form a combined messaging backbone.

First, WebSphere MQ and WebSphere Application Server have been designed so that their JMS implementations do interoperate. WebSphere MQ and WebSphere Application Server can exchange messages to form a combined messaging backbone where transactionality is preserved between them and publish-and-subscribe definitions can be shared. This means that a Java application hosted in WebSphere Application Server can use JMS to talk to other applications connected to WebSphere MQ, either by JMS or by its MQI. IBM products based on WebSphere Application Server, including IBM WebSphere ESB and WebSphere Process Server can also connect to WebSphere MQ in this manner, as well as use native MQI calls to connect to WebSphere MQ.

In addition, WebSphere Message Broker, the advanced ESB, provides unique mediations that can consolidate any JMS, Version 1.1 technology-compliant messaging products into a single messaging backbone.

Multilanguage message service

The industry-standard JMS interface described in the previous section requires that programmers have Java skills. Although Java is in widespread use today, it is not always the preferred choice of programming language, nor is it optimal for connecting to many non-Java environments. IBM has developed an API that is consistent with JMS but implemented in additional languages. Whereas JMS is geared specifically toward Java, the IBM API is provided in a range of languages, and is referred to as *multilanguage message service*. Often it is abbreviated to XMS, where X stands as a wildcard for “any” message service.

WebSphere MQ clients provide the XMS interface. Supported languages include C, C++ and a fully managed client for Microsoft .NET environments, which can be used with any .NET language such as C#.

Messaging for Web services

Web services standards define mechanisms for classifying, externalizing, finding and invoking services. These services could range from operations residing within your existing applications that can be made available to the rest of your enterprise, to new J2EE components sitting within an application server. And as Web services standards develop, concepts like Web Services-Reliable Messaging and Web Services-Notification are emerging.

Web Services Description Language (WSDL) defines the standards for Web services. Universal Description, Discovery and Integration (UDDI) provides directory and search capabilities. SOAP is the data format used when communicating with a Web service. However, a predicated standard for data transmission doesn't exist. You can send your SOAP messages using whatever transport suits you. One common method is to send SOAP messages over HTTP or HTTP Secure (HTTPS).

If you want your SOAP-formatted data to be delivered with the quality of service and benefits of a messaging backbone, you need to use a messaging product for the underlying data transfer. WebSphere MQ can send and receive SOAP data within a Web services implementation – most common within the J2EE environment – and often referred to as SOAP over JMS. This approach enables Web services to take advantage of the benefits of a messaging backbone.

Transferring files reliably

Many analysts estimate that as much as 80 percent of business information is shared around – and between – organizations using File Transfer Protocol (FTP) technology. You might be surprised by how much of your valuable business information is exposed to risk, and could be lost or unknowingly corrupted. Applications and data can become unreconciled, business processes can cease or work less effectively, and financial reports can be inaccurate. And perhaps worst of all – because the transfers are far from transparent – you might not even know when this has happened.

FTP technologies proliferate because of their simple appeal. An abundance of free FTP packages are available, the notion of file transfer is very intuitive and it usually only requires a basic level of skill to get going. There might not be any review or analysis of either the initial cost or the ongoing cost of deployment and maintenance of this solution. However, as dependency on these approaches increases, IT departments invest more time and skills to engineer additional function to try and address deficiencies in the reliability or security of these solutions. As more senders and receivers of files participate, the complexity of the environment rapidly increases and the business finds it is now trapped into maintaining and patching these solutions, which inhibits it from investing these resources in other IT projects, for example, to advance its progress to becoming more flexible or service-oriented.

Typically without realizing it, such an IT department has unwittingly now entered the middleware business. Even organizations that have made great strides in adopting integration middleware might still have a significant dependency on file transfers, although they might not be aware of this. This dependency not only reduces the flexibility of your business and affects responsiveness but can rack up hours of your staff's time in diagnosing errors or reworking supposedly simple solutions. Is this really the best use of your precious IT resources? What's needed is a way to incorporate files into your messaging backbone.

IBM resells PM4Data from IBM Business Partner MetaStorm Incorporated. PM4Data uses WebSphere MQ as its underlying transport and runs on most platforms supported by WebSphere MQ, including a specialized edition of PM4Data for z/OS. The combination of PM4Data and WebSphere MQ enables file transfers to be managed end to end, even remotely, and audited with end-to-end visibility. The transfers are assured by WebSphere MQ and take advantage of the enterprise-level security provided by WebSphere MQ including SSL. PM4Data can compress large files to optimize network traffic and has no theoretical bound on the size of file it can move over WebSphere MQ. Differences between platforms and between file types are handled automatically during transfers.

MetaStorm PM4Data enables you to manage all aspects of your file transfers, such as viewing and logging when file transfers started and ended, and seeing what transfers are in progress. PM4Data can help in determining whether transfers failed for any reason and monitor the rate of file transfers or the underlying performance of WebSphere MQ. With PM4Data you can restart or redirect file transfers on the fly so that there is usually no need to resend the file across its entire journey again. The file is simply redirected from where it became marooned.

As required, PM4Data can automatically transfer multiple files within a single WebSphere MQ message, optimizing use of your network. It is very extensible with customizable exit points and a powerful scripting language.

In addition to transferring files in a managed, reliable fashion, organizations are increasingly seeing the value in processing, transforming or enriching their files. WebSphere Message Broker File Extender enables you to feed files directly into WebSphere Message Broker where they can then be transformed, reformatted and enriched using its powerful mediation functions. It is fully integrated with the WebSphere Message Broker Toolkit palette, giving new and existing users of WebSphere Message Broker immediate productivity with this new extension.

With WebSphere Message Broker File Extender, you can extend WebSphere Message Broker to become a file broker as well as a message broker. This capability enables you to realize the same level of reuse and value from the data held in files as you currently get from the data being sent around your business in messages. You can apply brokering services such as transformation, enrichment, logging and routing to file content, and seamlessly convert messages into files, file records and vice versa, by simply wiring and configuring the relevant I/O nodes into the appropriate message flow.

Mobile and wireless devices

IBM WebSphere MQ Everyplace® provides the core features of a messaging backbone for devices with limited resources and optimized for fragile networks. This product supports the JMS standard and interoperates through a built-in bridge with WebSphere MQ.

Sensors and actuators

WebSphere Message Broker, IBM's advanced ESB, provides a transport that can extend the messaging backbone to reach very small (and embedded) devices, like sensors, valves and meters. This transport is called *MQ Telemetry Transport (MQTT)*.

Foundation for your ESB

A messaging backbone – even when used to connect just a few applications – can deliver substantial benefits, such as helping to reduce IT costs and making the infrastructure and organization more flexible. However, a messaging backbone also serves as a foundation for enabling other integration capabilities, providing a basis for deep, end-to-end integration and service orientation. A messaging backbone provides a springboard for launching IT projects aimed at addressing more-complex integration challenges and reaping even greater rewards.

An ESB provides an abstraction layer on top of the messaging backbone that enables it to be augmented with richer integration capabilities without the need to write code. Whereas messaging backbones are connected directly to applications, an ESB is primarily used by connecting it to services – loosely coupled, interoperable pieces of code that are independent of their underlying platform and programming language. An ESB also adds these capabilities to a messaging backbone:

- *Dynamically matching and routing messages between services*
- *Converting transport protocols or adjusting transport service-levels between services*
- *Transforming message formats, and enriching or altering message content in-flight between services*
- *Distributing business events between services*

WebSphere MQ provides a messaging backbone that can be used by all of IBM's ESB offerings – IBM WebSphere ESB, IBM WebSphere Message Broker and IBM WebSphere DataPower® XI50 Integration Appliance. WebSphere ESB is focused on using Web services standards and provides mediation services for XML-formatted data. WebSphere Message Broker is an advanced ESB, with extended capabilities to handle a broad range of standardized and nonstandard situations, mediating data in any kind of message format.

In turn, an ESB provides the grounding for business process management (BPM), enabling business tasks to be automated and managed more effectively. It also enables you to make more-informed decisions about how to evolve and change the way business activities are handled based on quantitative business-level analysis, and existing processes to be measured and optimized using key performance indicators (KPIs) and other business or financial metrics.

Summary

WebSphere MQ delivers a robust, innovative messaging backbone that provides a first step to SOA. Although WebSphere MQ is a mature product with a strong reputation and market presence, it continues to evolve. Today, as Web services gain momentum, IBM stands at the vanguard of standards definition in this area. And WebSphere MQ plays a key role as these exciting technologies develop.

However you plan to use SOA for greater IT flexibility, a messaging backbone provides compelling value for your organization, regardless of its size. If your organization wants to deploy an ESB, WebSphere MQ provides a foundational connectivity layer to build on. Small or midsize businesses with immediate needs to connect applications can still realize substantial benefits from implementing a messaging solution. Whatever business you are in, you can feel confident in the knowledge that IBM continues to invest in WebSphere MQ. IBM plans to continue to add new function and support new technologies, while improving existing function.

For more information

IBM WebSphere MQ software isn't just a product with a distinguished past. It's also a product with a long and exciting future dedicated to solving the fundamental connectivity problems faced within today's IT industry.

To learn more about messaging backbone and IBM WebSphere MQ, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/webspheremq

To join the Global WebSphere Community, visit:

www.websphere.org



© Copyright IBM Corporation 2007

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
03-07
All Rights Reserved

AIX, CICS, DataPower, eServer, Everyplace, HACMP, IBM, the IBM logo, IMS, iSeries, MQSeries, OS/390, OS/400, Parallel Sysplex, SupportPac, System i, System z, Tivoli, WebSphere, z/OS and zSeries are trademarks of International Business Machines Corporation in the United States, other countries or both.

Intel is a trademark of Intel Corporation in the United States, other countries or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries or both.

Other company, product and service names may be trademarks or service marks of others.

All statements regarding IBM future direction or intent are subject to change or withdrawal without notice and represent goals and objectives only.