1. Introduction

~~1.1. Abstract~~

This specification describes the objectives and functionality of the Java^{TM} Message Service (JMS).~~.~~

JMS provides a common way for Java programs to create, send, receive and read an enterprise messaging system's messages.

~~1.2.~~1.1. Overview of JMS

Enterprise messaging products (or as they are sometimes called, message-oriented middleware products) are ~~becoming~~ an essential component for integrating intra-company operations. They allow separate business components to be combined into a reliable, yet flexible, system.

JMS was initially developed to provide a standard Java API for the established messaging products that already existed. Since then many more messaging products have been developed.

JMS provides a common way for both Java client applications and Java middle-tier services to use these messaging products. It defines some messaging semantics and a~~In addition to the traditional MOM vendors, enterprise messaging products are also provided by several database vendors and a number of internet related companies.~~

~~Java language clients and Java language middle-tier services must be capable of using these messaging systems. JMS provides a common way for Java language programs to access these systems.~~

~~JMS is a~~ corresponding set of Java ~~interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product.~~interfaces.

Since messaging is a peer-to-peer technology, ~~all~~ users of JMS are referred to generically as *clients*. A JMS *application* is made up of a set of application defined messages and a set of clients that exchange them.

Messaging p~~P~~roducts that implement JMS do ~~this~~ so by supplying a *provider* that implements the JMS interfaces. Messaging products may support clients which use programming languages other than Java. Although such support is beyond the scope of JMS, the design of JMS has always accommodated the need for messaging products to support languages other than Java

~~1.2.1.~~1.1.1. ~~Is this a mail API~~What is messaging?

The term *messaging* is quite broadly defined in computing. It is used for describing various operating system concepts; it is used to describe email and fax systems; and here, it is used to describe asynchronous communication between enterprise applications.

Messages, as described here, are asynchronous requests, reports or events that are consumed by enterprise applications, not humans. They contain vital information needed to coordinate these systems. They contain precisely formatted data that describe specific business actions. Through the exchange of these messages each application tracks the progress of the enterprise.

~~1.2.2. Existing messaging systems~~

~~Messaging systems are peer-to-peer facilities. In general, each client can send messages to, and receive messages from, any client. Each client connects to a messaging agent which provides facilities for creating, sending and receiving messages.~~

~~Each system provides a way of addressing messages. Each provides a way to create a message and fill it with data.~~

~~Some systems are capable of broadcasting a message to many destinations. Others only support sending a message to a single destination.~~

~~Some systems provide facilities for asynchronous receipt of messages (messages are delivered to a client as they arrive). Others support only synchronous receipt (a client must request each message).~~

~~Each messaging system typically provides a range of service that can be selected on a per message basis. One important attribute is the lengths to which the system will go to ensure delivery. This varies from simple best effort to guaranteed, only once delivery. Other important attributes are message time-to-live, priority and whether a response is required.~~

~~1.2.3.~~1.1.2. ~~JMS~~The objectives of JMS

~~If JMS provided a union of all the existing features of messaging systems it would be much too complicated for its intended users. On the other hand, JMS is more than an intersection of the messaging features common to all products. It is crucial that~~The objectives of JMS ~~include~~ are

- to provide Java ~~the~~ applications with the messaging functionality needed to implement sophisticated enterprise applications

to define ~~.~~

- ~~JMS defines~~ a common set of ~~enterprise~~ messaging concepts and facilities
- to ~~. It attempts to~~ minimize the ~~set of~~ concepts a Java language programmer must learn to use enterprise messaging products
- to ~~. It strives to~~ maximize the portability of Java messaging applications between different messaging products~~.~~

~~1.2.3.1. JMS provider~~

~~As noted earlier, a JMS provider is the entity which implements JMS for a messaging product.~~

~~Ideally, JMS providers will be written in 100% Pure Java so they can run in applets; simplify installation; and, work across architectures and OS's.~~

~~An important goal of JMS is to minimize the work needed to implement a provider.~~

~~1.2.3.2. JMS messages~~

~~JMS defines a set of message interfaces.~~

~~Clients use the message implementations supplied by their JMS provider.~~

~~A major goal of JMS is that clients have a consistent API for creating and working with messages which is independent of JMS provider.~~

~~1.2.3.3.~~1.1.3. JMS domains

JMS supports the two major styles of messaging provided by enterprise m~~M~~essaging products~~.~~:

~~can be broadly classified as either *point-to-point* or *publish-subscribe* systems.~~

- Point-to-point (PTP) ~~products~~ messaging ~~are~~ allows a client to send a message to another client~~built around the concept of message queues.~~ via an intermediate abstraction called a *queue*. The client that sends the message sends it to a specific queue. The client that receives the message ~~Each message is addressed to a specific queue; clients extract~~extracts ~~messages~~ it from ~~the~~ that queue.~~(s) established to hold their messages.~~

- Publish and subscribe (Pub/Sub) messaging allows a client to send messages to multiple clients via an intermediate abstraction called a *topic*. The client that sends the message *publishes* it to a specific topic. The message will then be delivered to all the clients that are *subscribed* to that topic.~~clients address messages to some node in a content hierarchy. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a node's multiple publishers to its multiple subscribers.~~

- ~~JMS provides a set of interfaces that allow the client to send and receive messages in both domains, while supporting the semantics of each domain. JMS also provides client interfaces tailored for each domain. Prior to version 1.1 of the JMS specification, only the client interfaces that were tailored to each domain were available. These interfaces continue to be supported to provide backward compatibility for those who have already implemented JMS clients using them. The preferred approach for implementing clients is to use the domain-independent interfaces. These interfaces, referred to as the "common interfaces", are parents of the domain-specific interfaces.~~

~~1.2.3.4. Portability~~

~~The primary portability objective is that new, JMS only, applications are portable across products within the same messaging domain.~~

~~This is in addition to the expected portability of a JMS client across machine architectures and operating systems (when using the same JMS provider).~~

~~Although JMS is designed to allow clients to work with existing message formats used in a mixed language application, portability of such clients is not generally achievable (porting a mixed language application from one product to another is beyond the scope of JMS).~~

~~1.2.4.~~1.1.4. What JMS does not include

JMS does not address the following ~~-~~functionality:

- Load ~~B~~balancing/f~~F~~ault t~~T~~olerance - Many products provide support for multiple, cooperating clients implementing a critical service. The JMS API does not specify how such clients cooperate to appear to be a single, unified service.
- Error/a~~A~~dvisory n~~N~~otification - Most messaging products define system messages that provide asynchronous notification of problems or system events to clients. JMS does not attempt to standardize these messages. By following the guidelines defined by JMS, clients can avoid using these messages and thus prevent the portability problems their use introduces.
- Administration - JMS does not define an API for administering messaging products.
- Security - JMS does not specify an API for controlling the privacy and integrity of messages. It also does not specify how digital signatures or keys are distributed to clients. Security is considered to be a JMS provider-specific feature that is configured by an administrator rather than controlled via the JMS API by clients.
- Wire ~~P~~protocol - JMS does not define a wire protocol for messaging.
- Message ~~T~~type r~~R~~epository - JMS does not define a repository for storing message type definitions and it does not define a language for creating message type definitions.

~~*1.2. What is required by JMS*~~

~~The functionality discussed in the specification is required of all JMS providers unless it is explicitly noted otherwise.~~

~~JMS is also used within the Java Platform, Enterprise Edition (Java EE). See section 1.4 "Relationship to other Java APIs" for additional requirements for JMS when it is integrated into a Java EE environment.~~

~~*1.3. Relationship to other Java APIs*~~

~~*1.3.1. Java DataBase Connectivity (JDBC™) software*~~

~~JMS clients may also use the JDBC API. They may desire to include the use of both the JDBC API and the JMS API in the same transaction. In most cases, this will be achieved automatically by implementing these clients as Enterprise JavaBeans™ components. It is also possible to do this directly with the Java Transaction API (JTA).~~

~~*1.3.2. JavaBeans™ components*~~

~~JavaBeans components can use a JMS session to send/receive messages. JMS itself is an API and the interfaces it defines are not designed to be used directly as JavaBeans components.~~

~~*1.3.3. Enterprise JavaBeans™ component model*~~

~~The JMS API is an important resource available to Enterprise Java Beans (EJB™) component developers. It can be used in conjunction with other resources like JDBC to implement enterprise services.~~

~~The EJB specification defines beans that are invoked synchronously via method calls from EJB clients. It also defines a form of asynchronous~~

~~bean that is invoked when a JMS client sends it a message, called a message-driven bean. The EJB specification supports both synchronous and asynchronous message consumption. In addition, EJB specifies how the JMS API participates in bean-managed or container-managed transactions. The EJB specification restricts how to use JMS interfaces when implementing EJB clients. Refer to the EJB specification for the details.~~

~~*1.3.4. Java Transaction API (JTA)*~~

~~The *javax.transaction* package provides a client API for delimiting distributed transactions and an API for accessing a resource's ability to participate in a distributed transaction.~~

~~A JMS client may use JTA to delimit distributed transactions; however, this is a function of the transaction environment the client is running in. It is not a feature of JMS.~~

~~A JMS provider can optionally support distributed transactions via JTA.~~

~~The JTA specification also defines a scope `@TransactionScope` which is referred to in section 12.4.4 "Scope of injected JMSContext objects".~~

~~*1.3.5. Java Transaction Service (JTS)*~~

~~JMS can be used in conjunction with JTS to form distributed transactions that combine message sends and receives with database updates and other JTS aware services. This should be handled automatically when a JMS client is run from within an application server such as an Enterprise JavaBeans server; however, it is also possible for JMS clients to program this explicitly.~~

~~*1.3.6. Java Naming and Directory Interface™ (JNDI) API*~~

~~JMS clients look up configured JMS objects using the JNDI API. JMS administrators use provider-specific facilities for creating and configuring these objects.~~

~~This division of work maximizes the portability of clients by delegating provider specific work to the administrator. It also leads to more administrable applications because clients do not need to embed administrative values in their code.~~

1.1.5. Java SE and Java EE support

The JMS API is designed to be suitable for use by both Java client applications using the Java™ Platform, Standard Edition (Java SE), and Java middle-tier services using the Java™ Platform, Enterprise Edition (Java EE).

All JMS providers must support its use by Java client applications using Java SE. It is optional whether a given JMS provider supports its use by middle-tier applications using Java EE.

~~*1.3.7.* Java Platform, Enterprise Edition (Java EE)~~

The ~~Java™ Platform, Enterprise Edition (~~Java EE~~)~~ Specification requires a full Java EE platform implementation to include a messaging provider which supports the JMS API in both Java SE and Java EE applications.

Java EE makes a number of additional features available to messaging applications in addition to those defined in the JMS specification itself, most notably message-driven beans (MDBs) and JTA transactions. Java EE also imposes a number of restrictions on the use of the JMS API.

For more information on the use of JMS by Java EE applications, see chapter 12 "Use of JMS API in Java EE applications".

~~1.3.8. Contexts and dependency injection (CDI)~~

~~This specification defines how `JMSContext` objects may be injected into Java EE web or EJB applications. See section 12.4 "Injection of JMSContext objects" for more information. The CDI (Contexts and dependency injection) specification defines the concepts and technology on which this is based.~~

~~1.4.~~1.2. *What is new in JMS 2.0?*

A full list of the new features, changes and clarifications introduced in JMS 2.0 is given in section A.1 "Version 2.0" of the "Change history~~Change history~~" chapter. Here is a summary:

The JMS 2.0 specification now requires JMS providers to implement both P2P and Pub-Sub.

The following new messaging features have been added in JMS 2.0:

- Delivery delay: a message producer can now specify that a message must not be delivered until after a specified time interval.
- New send methods have been added to allow an application to send messages asynchronously.
- JMS providers must now set the `JMSXDeliveryCount` message property.

The following change has been made to aid scalability:

- Applications are now permitted to create multiple consumers on the same durable or non-durable topic subscription. In previous versions of JMS only a single consumer was permitted.

Several changes have been made to the JMS API to make it simpler and easier to use:

- `Connection`, `Session` and other objects with a `close()` method now implement the `java.jang.AutoCloseable` interface to allow them to be used in a Java SE 7 try-with-resources statement.
- A new "simplified API" has been added which offers a simpler alternative to the previous API, especially in Java EE applications.
- New methods have been added to create a session without the need to supply redundant arguments.
- Although setting client ID remains mandatory when creating an unshared durable subscription, it is optional when creating a shared durable subscription.
- A new method `getBody` has been added to allow an application to extract the body directly from a `Message` without the need to cast it first to an appropriate subtype.

A new chapter has been added which describes some additional restrictions and behaviour which apply when using the JMS API in the Java EE web or EJB container. This information was previously only available in the EJB and Java EE platform specifications.

A new chapter has been added which adds a new recommendation for a JMS provider to include a resource adapter, and which defines a number of activation configuration properties.

New methods have been added to `Session` which return a `MessageConsumer` on a durable topic subscription. Applications could previously only obtain a domain-specific `TopicSubscriber`, even though its use was discouraged.

The specification has been clarified in various places.