

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 4, 2008

D. Xiao
BEA
R. Merrick
IBM
P. Easton
Progress
T. Frank
Software AG
E. Johnson
TIBCO
January 2008

URI Scheme for Java(tm) Message Service 1.0
draft-merrick-jms-uri-01

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on July 4, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Abstract

This document defines the format of Uniform Resource Identifiers

(URI) as defined in [[RFC3986](#)], for designating connections and destination addresses used in the Java(tm) Messaging Service (JMS) [[REF-JMS](#)]. It was originally designed for particular uses, but should have general applicability wherever a JMS URI is needed to describe the connection to a JMS provider, and access to a JMS destination. The syntax of this 'jms' URI is not compatible with any known current vendor implementation, but the expressivity of the format should permit all vendors to use it.

Table of Contents

| | | |
|-----------------------|---|--------------------|
| 1. | Introduction | 3 |
| 1.1. | Requirements notation | 3 |
| 2. | URI Scheme Name | 3 |
| 3. | Syntax of a jms URI | 3 |
| 4. | URI scheme semantics | 4 |
| 4.1. | Shared Parameters | 5 |
| 4.2. | JNDI Variant | 5 |
| 4.3. | Context Variant | 8 |
| 4.4. | Custom parameters | 8 |
| 5. | Encoding considerations | 9 |
| 6. | Applications/protocols that use the JMS URI scheme name | 9 |
| 7. | Interoperability considerations | 9 |
| 8. | Security Considerations | 10 |
| 8.1. | Reliability and Consistency | 10 |
| 8.2. | Malicious Construction | 10 |
| 8.3. | Back-end Transcoding | 11 |
| 8.4. | Semantic Attacks | 11 |
| 8.5. | Other Security Concerns | 12 |
| 9. | IANA Considerations | 12 |
| 10. | Contributors | 12 |
| 11. | Acknowledgements | 13 |
| 12. | References | 13 |
| 12.1. | Normative References | 13 |
| 12.2. | Informative References | 13 |
| | Authors' Addresses | 14 |
| | Intellectual Property and Copyright Statements | 16 |

1. Introduction

The "jms" URI scheme is used to designate a `javax.jms.Destination` object and an associated `javax.jms.ConnectionFactory` object, and optionally provide additional information concerning the way that the Destination object is to be used. Probably the most common, and certainly the most compatible way in Java to retrieve such destinations, is via Java Naming and Directory Information (JNDI) [[REF-JNDI](#)] methods. So as to extend compatibility to existing vendor mechanisms beyond JNDI lookup, the "jms" URI syntax allows variants on the core syntax. The variant exists as an explicit part of the syntax so that tools that are otherwise unfamiliar with the variant can recognize the presence of a URI with an alternate interpretation.

In its simplest and most interoperable form, this URI scheme starts with "jms:jndi:" plus a JNDI name for a Destination. Since interaction with some resources may require JNDI contextual information or JMS headers and properties to be specified as well, the "jndi" variant of the "jms" URI scheme includes support for supplying this additional JNDI information as query parameters.

While the "jndi" variant provides compatibility, vendors may define additional variants. This specification defines two variants, "jndi" and "context".

1.1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)].

All syntax descriptions use the ABNF specified by [[RFC4234](#)], Augmented BNF for Syntax Specifications: ABNF.

2. URI Scheme Name

The name of the URI scheme is 'jms'.

3. Syntax of a jms URI

The following ABNF describes the jms scheme URI syntax:

```
jms-uri = "jms:" jms-variant ":" jms-dest
         [ '?' param *([ '&' param ]) ]

jms-variant = segment-nz-nc

jms-dest = path-rootless ; specific meaning per variant

param = param-name "=" param-value

param-name = 1*(unreserved | pct-encoded)

param-value = *(unreserved | pct-encoded)

segment-nz-nc = ; as defined in RFC 3986

path-rootless = ; as defined in RFC 3986

unreserved = ; as defined in RFC 3986

pct-encoded = ; as defined in RFC 3986
```

4. URI scheme semantics

The required particles in the JMS URI are the scheme name ("jms"), the variant identifier, and the "jms-dest" portions. The two recognized variants (jms-variant above) are "jndi", and "context". The "jms-dest" portion identifies the JMS destination object in a way that is determined by the particular variant.

Each variant may have query parameters specific to that variation. All such parameters that cannot be shared across schemes should use the name of the variant as the prefix to the parameters. Parameters that apply across multiple variants, perhaps because they are generally applicable, such as JMS settings, should not have any particular prefix, and should not begin with any known prefix. This latter convention enables tools that are otherwise unfamiliar with a particular variant to recognize that a particular URI includes parameters specific to that variant.

Examples of the URI scheme include:

```
jms:jndi:
SomeJndiNameForDestination?jndiInitialContextFactory=com.example.jndi
.JndiFactory&priority=3

jms:context:SomeContextName?timeToLive=1000
```

[4.1.](#) Shared Parameters

In addition to the required particles, the jms URI scheme supports the following "shared" parameters, which may be included as parameters in any order (following the '?' parameter-start indicator, and separated by '&'). This pattern is consistent with other non-hierarchical URI specifications.

[4.1.1.](#) deliveryMode

Indicates whether the request message is persistent or not. This property corresponds to the JMS message header "JMSDeliveryMode" defined in [section 3.4.2.](#) of the JMS 1.1 specification. This may be "PERSISTENT" or "NON_PERSISTENT". If this parameter is not specified then the JMS default SHOULD be used.

[4.1.2.](#) timeToLive

The lifetime, in milliseconds, of the request message. This property corresponds to the JMS Time-To-Live value defined in [section 4.8](#) of the JMS 1.1 specification. If this parameter is not specified then the JMS default SHOULD be used.

[4.1.3.](#) priority

The JMS priority associated with the request message. As per [section 3.4.10](#) of the JMS 1.1 specification this must be a number between 0 and 9, inclusive, and corresponds to the JMS message header "JMSPriority". If this parameter is not specified then the JMS default SHOULD be used.

[4.1.4.](#) replyToName

This property corresponds to the JMS message header "JMSReplyTo" defined in [section 3.4.6](#) of the JMS 1.1 specification. Specifies the JMS destination object to which a response message should be sent in a way that is determined by the particular variant.

[4.2.](#) JNDI Variant

The "jndi" variant implies the use of JNDI for discovering the Destination object. When this is specified as the variant, the jms-dest portion of the syntax is the name for JNDI lookup purposes. Additional JNDI specific parameters may be specified. The JNDI specific parameters SHOULD only be processed when the URI variant is "jndi".

[4.2.1.](#) JNDI Parameters

[4.2.1.1.](#) `jndiConnectionFactoryName`

Specifies the JNDI name of the Java class providing the connection factory.

[4.2.1.2.](#) `jndiInitialContextFactory`

Specifies the fully qualified Java class name of the "InitialContextFactory" implementation class to use.

[4.2.1.3.](#) `jndiURL`

Specifies the JNDI provider URL, in a form consistent with `javax.naming.spi.NamingManager.getURLContext(String scheme, Hashtable environment)` as defined in the JNDI specification.

[4.2.2.](#) Performing a JNDI Look-up

To perform a look-up based on a JNDI variant URI an application must create a JNDI InitialContext object. The InitialContext object can then be used to look up the JMS ConnectionFactory object (using the "jndiConnectionFactoryName" URI parameter); the target JMS Destination object (using the "jms-dest" portion of the JMS URI); and the "replyToName" JMS Destination object (if the "replyToName" parameter is specified on the URI).

The application creates the InitialContext object by first setting up two properties: "Context.INITIAL_CONTEXT_FACTORY", with the value of the `jndiInitialContextFactory` JMS URI parameter; and "Context.PROVIDER_URL", with the value of the `jndiURL` URI parameter, and then passing the two properties to the InitialContext constructor.

To locate a connection factory or destination object, the application passes the name of the object into the `InitialContext.lookup()` method.

For example, the JMS URI...

```
jms:jndi:REQ_QUEUE?jndiURL=file:/C:/JMSAdmin
&jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory
&jndiConnectionFactoryName=CONNFACT
&replyToName=RESP_QUEUE
```

...would be used by the following code sample to locate and retrieve

a JMS ConnectionFactory called "CONNFACT", and JMS Destinations called "REQ_QUEUE" and "RESP_QUEUE", from a file system JNDI context called "c:/JMSAdmin".

```
/*
 * Preconditions on URI:
 * - portion "jms-dest" has been parsed into variable "jms_dest"
 * - parameters "jndiConnectionFactoryName",
 *   "jndiInitialContextFactory", "replyToName" and "jndiURL" have
 *   been parsed into variables of the same name
 */
Hashtable environment = new Hashtable();
environment.put(Context.INITIAL_CONTEXT_FACTORY,
    jndiInitialContextFactory);
environment.put(Context.PROVIDER_URL, jndiURL);
/*
 * Create File System Initial Context
 */
Context ctx = new InitialContext(environment);
/*
 * Now get the JMS ConnectionFactory and Destination. These will be
 * used later on in the application to create the JMS Connection and
 * send / receive messages
 */
ConnectionFactory jmsConnFact = (ConnectionFactory)
    ctx.lookup(jndiConnectionFactoryName);
Destination requestDest = (Destination) ctx.lookup(jms_dest);
Destination replyDest = (Destination) ctx.lookup(replyToName);
```

The ConnectionFactory is used to create a Connection, which itself is used to create a Session. The session can then be used to create the MessageProducer - which sends messages to the target destination, and the MessageConsumer which receives messages from the replyToName destination (as shown in the following code extract)


```
/*
 * Create a producer to send a message to the request destination
 * that was specified in the URI, then create the message, setting
 * the replyToName destination in the message to the one specified
 * in the URI, and send it.
 */
MessageProducer producer = sess.createProducer(requestDest);
BytesMessage reqMsg = sess.createBytesMessage();
reqMsg.setJMSReplyTo(replyDest);
producer.send(reqMsg);
/*
 * Create a consumer to get a message from the replyToName
 * destination using a selector to get the specific response to this
 * request. The responder must set the correlation ID of the response
 * to the message ID of the request message
 */
MessageConsumer consumer = sess.createConsumer(replyDest,
        "JMSCorrelationID = '" + reqMsg.getJMSMessageID() + "'");
Message respMsg = (Message) consumer.receive(300000);
```

[4.3.](#) Context Variant

The "context" variant intends that the `jms-dest` is purely a logical name, and the application will somehow bind that logical name to an actual destination. How this binding is done is outside the scope of this specification. Note that all of the shared parameters defined above may be used with this variant, although it defines no additional parameters.

[4.3.1.](#) Performing a Context Lookup

The lookup mechanism for the context variant must result in the same set of JMS objects as those produced by a JNDI variant URI lookup - that is a JMS ConnectionFactory; a JMS Destination object for the target destination; and a JMS Destination for the `replyToName` (if specified) .

[4.4.](#) Custom parameters

The set of parameters is extensible. Any other vendor- or application-defined parameter may be supplied, in the URI, by passing it as "param-name=param-value" just like the set of well-known parameters.

Warning: Vendors and applications MUST NOT include sensitive information (such as authorization tokens) in a URI. Other means of authorization, authentication, and identification should be used.

Also see the security discussion below about properties that may be duplicated as JMS message properties.

5. Encoding considerations

The jms URI scheme distinguishes between "unreserved" characters and "pct-encoded" characters, as defined in [\[RFC3986\]](#). Apart from these encoding considerations, the characters '?' and '&' MUST be encoded when they appear within the "jms-dest" particle (for example, a JNDI name) or in query parameters. The character ':' SHOULD be escaped, when appearing in the "jms-dest" portion of the syntax.

Conversions to and from IRIs should follow the rules of [RFC 3987](#), sections [3.1](#) and [3.2](#). As per sections [1.2c](#) and [6.4](#) of [\[RFC3987\]](#), all parts of the jms URI MUST use the UTF-8 encoding when converting to and from IRI format.

6. Applications/protocols that use the JMS URI scheme name

A variety of vendors provide implementations of the JMS Service Provider Interface. These products interoperate at the API level, in the Java programming language.

Some vendors have provided additional products which interoperate with their own SPI implementations. These extensions may also be able to make use of this URI scheme.

The vendors working on this URI scheme are also working on a specification for carrying SOAP messages over their respective implementations of JMS [\[REF-SOAPJMS\]](#). In addition, the Service Component Architecture Bindings TC [\[1\]](#) at OASIS will employ the jms URI scheme to identify JMS Destinations in appropriate circumstances.

7. Interoperability considerations

This jms URI scheme focuses on identifying a JMS Destination object, and some characteristics of communication using that Destination, and specifically excludes any notion of describing how JMS itself is implemented and how it delivers messages. As a consequence of this focus, interoperability concerns are limited to how implementations obtain and use a Destination object.

This scheme definition describes two variants for obtaining a Destination. These variants achieve their aims with the use of JNDI and JMS APIs, with no new APIs or protocols defined here. As a

consequence, interoperability concerns may arise as a result of implementations that do not conform to the specifications for those APIs. Further, the use of Java, and JNDI in particular, means that the configuration of the execution environment, and the use of Java ClassLoaders may affect the interpretation of any given URI. Consumers of these URIs are urged to consider the scope and consistency of the environment across which these URIs will be shared.

As described in [Section 4](#), others can define additional variants, which provide the means to describe how to look up JMS Destination objects in a manner specific to some environment. For any new variant, the shared parameters defined in [Section 4.1](#) MUST have the same meaning in that variant as they do here. That way, tools and people can safely copy these parameters between environments. Customers should be aware that use of variants not defined here may make it more difficult to switch to an alternate JMS provider.

[8.](#) Security Considerations

[Section 7 of \[RFC3986\]](#) identifies some of the security concerns that should be identified in this specification.

[8.1.](#) Reliability and Consistency

This specification identifies only the variant ("jms-variant") and variant specific details ("jms-dest") as an essential part of the URI. For reliability and consistency purposes, these are the only part that can reasonably be expected to be stable. Other optional JMS configuration and message properties, indicated as URI parameters, like the "timeToLive", may reasonably be determined by the sender of a message, without affecting the recipient. Insofar as a recipient may wish to dictate certain parameters, such as the "jndiConnectionFactoryName", those parameters can be specified.

[8.2.](#) Malicious Construction

[8.2.1.](#) Recipient Concerns

A malicious consumer of a service using a JMS URI could send, as part of a JMS message, a URI with a parameter such as "timeToLive" with a value specified in the URI that differs from the corresponding JMS message property ("JMSExpiration" header, in this example). In the case of such messages with such URIs, recipients are strongly cautioned to avoid applying processing logic based on particular URI parameters. Discrepancies in the message could be used to exploit differences in behavior between the selectors that a JMS-based

application might use to affect which messages it sees, and the processing of the rest of the application. As defined in this document, the parameters of concern include:

```
deliveryMode
timeToLive
priority
```

Message senders are strongly urged to remove from the URI extra parameters like the above in environments where the data will be redundant with information specified elsewhere in the JMS message.

Any use of additional parameters, either as a part of a definition of a new variant, or for more general use, should also specify whether those parameters should be removed by a sender as specified here. If a recipient is aware of the jms URI scheme, and it receives a message containing a JMS URI, it MUST ignore or discard parameters that it does not recognize.

[8.2.2.](#) Sender Concerns

A third party could intercept and replace a URI containing any of the JMS/JNDI configuration parameters, such as "jndiConnectionFactoryName", "jndiInitialContextFactory", "jndiURL". As these parameters may affect how an implementation establishes an initial connection, such parameters could be used as a means to subvert communications. This could possibly result in re-routing communications to third-parties, who could then monitor sent messages. Clients should use these URI parameters only when assured of their validity in trusted environments.

[8.3.](#) Back-end Transcoding

This specification, in using the URI specification, and building around the JMS specification, has no particular transcoding issues. Any such issues are problems with the underlying implementation of Java and Java Messaging Service being employed.

[8.4.](#) Semantic Attacks

A possible semantic attack on the "jndi" variant could be accomplished by replacing characters of the JMS URI from one language with equivalent looking characters from another language, known as an "IDN homograph attack" (IDN) [[REF-Homograph](#)]. This kind of attack could occur in a variety of ways. For example, if an environment allows for the automatic registration of JNDI destination names, a malicious actor could register and then publicize an alternate of an existing destination name. Such an environment ought to prevent the

use of homograph equivalents, perhaps by restricting allowed characters, so that clients do not accidentally send their requests to unintended destinations.

The "context" variant likewise relies on an application or a human to associate an intended JMS destination with an actual destination, perhaps during a deployment process. Particularly when a human is involved, concerns similar to the JNDI environment apply.

[8.5](#). Other Security Concerns

This specification does not define or anticipate any use for IP addresses as part of the URI, so no issues around IP addresses, rare or otherwise, are raised by this specification.

This specification does not define any characteristics of a jms scheme URI that contain sensitive information.

[9](#). IANA Considerations

The IANA is asked to register the Java Message Service URI scheme described in this document, according to the following scheme registration request:

- o URI scheme name: "jms" is requested
- o Status: Permanent is requested
- o URI scheme syntax: See [Section 3](#)
- o URI scheme semantics: See [Section 4](#)
- o Encoding considerations: See [Section 5](#)
- o Applications/protocols that use this URI scheme name: See [Section 6](#)
- o Interoperability considerations: See [Section 7](#)
- o Security considerations: See [Section 8](#)
- o Contact: See Authors section
- o References See References section

[10](#). Contributors

The authors gratefully acknowledge the contributions of:

Phil Adams - International Business Machines Corporation -
phil_adams@us.ibm.com
Glen Daniels - WS02 - glen@wso2.com
Peter Easton - Progress Software - peaston@progress.com
Tim Frank - Software AG. - tim.frank@softwareag.com

Lei Jin - BEA Systems, Inc. - ljin@bea.com
Eric Johnson - TIBCO Software Inc. - eric@tibco.com
Vinod Kumar - BEA Systems, Inc. - vkumar@bea.com
Amelia A. Lewis - TIBCO Software Inc. - alewis@tibco.com
Roland Merrick - International Business Machines Corporation -
roland@uk.ibm.com
Mark Phillips - International Business Machines Corporation -
m8philli@uk.ibm.com
Stephen Todd - International Business Machines Corporation -
stephen_todd@uk.ibm.com
Dongbo Xiao - BEA Systems, Inc. - xiaod@bea.com
Prasad Yendluri - Software AG - prasad.yendluri@softwareag.com

11. Acknowledgements

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

This document was produced using the xml2rfc tool [[RFC2629](#)].

12. References

12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, [RFC 3986](#), January 2005.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", [RFC 3987](#), January 2005.
- [RFC4234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", [RFC 4234](#), October 2005.

12.2. Informative References

- [REF-Homograph] Unknown, "IDN Homograph attack", any 2005-2006, http://en.wikipedia.org/wiki/IDN_homograph_attack.
- [REF-JMS] Hapner, M., Burridge, R., Sharma, R., Fialli, J., and K. Stout, "Java Message Service (JMS)", April 2002, <http://java.sun.com/products/jms/>.

[REF-JNDI]

Sun Microsystems, Inc., "Java Naming and Directory Interface Application Programming Interface", July 1999, <<http://java.sun.com/products/jndi/docs.html>>.

[REF-SOAPJMS]

Daniels, G., Easton, P., Frank, T., Johnson, E., Lewis, A., Merrick, R., Phillips, M., and D. Xiao, "SOAP over JMS", October 2007, <<http://www.w3.org/Submission/SOAPJMS/>>.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", [RFC 2629](#), June 1999.

URIs

[1] <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sca-bindings>

Authors' Addresses

Dongbo Xiao
BEA Systems Inc.
140 Allen Road
New Jersey, NJ 07938
United States

Email: xiaod@bea.com

Roland Merrick
International Business Machines Corporation
PO Box 31, Birmingham Road
Warwick, CV34 5JL
United Kingdom

Email: roland@uk.ibm.com

Peter Easton
Progress Software Corporation
14 Oak Park Drive
Bedford, MA 01730
United States

Email: peaston@progress.com

Tim Frank
Software AG.
3877 Fairfax Ridge
Fairfax VA 22030
United States

Email: tim.frank@softwareAG.com

Eric Johnson
TIBCO Software Inc.
3303 Hillview Avenue
Palo Alto CA 94304
United States

Email: eric@tibco.com

Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).