Internet Engineering Task Force (IETF)
Request for Comments: 6167
Category: Informational
ISSN: 2070-1721

M. Phillips
P. Adams
IBM
D. Rokicki
Software AG
E. Johnson
TIBCO

April 2011

URI Scheme for Java(tm) Message Service 1.0

Abstract

This document defines the format of Uniform Resource Identifiers (URIs) as defined in RFC 3986, for designating connections and destination addresses used in the Java(tm) Messaging Service (JMS). It was originally designed for particular uses, but applies generally 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 previously existing, but unregistered, "jms" URI schemes. However, the expressiveness of the scheme described herein should satisfy the requirements of all existing circumstances.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6167.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Requirements Notation 4
2. URI Scheme Name 5
3. Syntax of a JMS URI 5
4. URI Scheme Semantics 5
4.1. Shared Parameters 6
4.2. "jndi" Variant 7
4.3. Vendor Destination Names -- Variants "queue" and "topic" ..11
4.4. Custom Parameters 12
5. Encoding Considerations 13
6. Applications/Protocols That Use the JMS URI 13
7. Interoperability Considerations 13
8. Security Considerations 14
8.1. Reliability and Consistency 14
8.2. Malicious Construction 14
8.3. Back-End Transcoding 15
8.4. Semantic Attacks 15
8.5. Other Security Concerns 16
9. IANA Considerations 16
9.1. URI Scheme Registration 16
9.2. "jms" URI Scheme Registries 17
10. Contributors 18
11. Acknowledgements 19
12. References 20
12.1. Normative References 20
12.2. Informative References 21

1. Introduction

The "jms" URI scheme is used to designate a javax.jms.Destination object and an associated javax.jms.ConnectionFactory object [JMS], and, optionally, to 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) [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 might require JNDI contextual information or JMS header fields 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 can define additional variants. This specification defines three variants: "jndi", "queue", and "topic". Vendors defining additional variants are strongly encouraged to register them with IANA as documented in Section 9.2.1.

While the "jms" URI scheme allows the location of network resources, it does not map to a single underlying protocol, unlike most other URI schemes that do so. Instead, it achieves interoperability through the use of a common Java-based API [JAVA] for messaging. Because of this, interoperability is dependent upon the implementation of the API and its capabilities; two implementations of JMS might or might not interoperate in practice. Furthermore, it might be impractical to use JMS URIs in non-Java environments.

As a consequence of building upon an API, rather than a protocol, the utility of a JMS URI depends on the context in which it is used. That context includes agreement on the same JMS provider or underlying protocol; agreement on how to look up endpoints (JNDI); and, when using serialized Java object messages, sufficiently similar Java Class environments that serialized objects can be appropriately read and written. Users of this scheme need to establish the necessary shared-context parts as just enumerated -- a context that can span the globe, or merely a small local network. With that shared context, this URI scheme enables endpoint identification in a uniform way, and the means to connect to those endpoints.

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 [RFC5234], "Augmented BNF for Syntax Specifications: ABNF".

Note that some examples in this document wrap long JMS URIs for readability. The line breaks are not part of the actual URIs.

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 = segment-nz ; 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>
```

The URIs are percent-encoded UTF-8 [RFC3629]. Please see Section 5 of this document for encoding considerations.

4. URI Scheme Semantics

JMS URIs are used to locate JMS [JMS] Destination resources and do not specify actions to be taken on those resources. Operations available on JMS Destinations are fully and normatively defined by the JMS specification and as such are out of scope for this URI specification.

The required portions of the syntax include the terminal of "jms" for the URI scheme name; the <jms-variant> element to indicate the variant of the scheme; and the <jms-dest> element, which identifies the Destination based on the chosen variant. For the <jms-variant> element, this document defines three values: "jndi", "queue", and "topic". All the terminals resulting from <jms-variant> and <jms-dest> production rules are case-sensitive.

Parameters further refine how to locate and use the Destination. The parameter names and values are case-sensitive. They can occur in any order, and each parameter name SHOULD NOT appear more than once. In the event that a parameter appears multiple times, all but the last instance of the parameter MUST be ignored. For comparison purposes, the absence of a parameter does not mean the same thing as a URI with a parameter set to a default value, due to the potential variation in default values as determined by the context of a specific use.

Each variant can have query parameters specific to that variation. All such variant-specific parameters SHOULD use the name of the variant as the prefix to the parameters. For example, a vendor-specific variant of "vnd.example.ex" might also define a parameter with a name like "vnd.example.exParameter". Parameters that apply across multiple variants -- perhaps because they are generally applicable, such as JMS settings -- MUST NOT have a name that starts with the name of any known variant. This pattern enables tools that are otherwise unfamiliar with a particular variant to distinguish those parameters that are specific to a variant from those that are more generally applicable.

Examples of the URI scheme include:

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

jms:queue:ExampleQueueName?timeToLive=1000
```

4.1. Shared Parameters

In addition to the required particles, the "jms" URI scheme supports the following shared parameters, which are available to all variants. These parameters correspond to headers and properties on the JMS Messages to be sent. For the parameters deliveryMode, timeToLive, and priority, the default values might be specified in the context of a specific use, for example by environment variables, or in the configuration of a particular network application. JMS also defines default values for these properties. The context default is hereby defined as the default value in the context of a specific use, or the JMS default for a particular property if the context does not define a default.

4.1.1. deliveryMode

Indicates whether the request message is persistent or not. This property corresponds to the JMS message header field "JMSDeliveryMode" defined in Section 3.4.2 of the JMS 1.1 specification [JMS]. The value of this parameter MUST be "PERSISTENT" or "NON_PERSISTENT". If this parameter is not specified, then the context default MUST be used.

4.1.2. timeToLive

The lifetime, in milliseconds, of the request message, specified as a decimal number. 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 context default MUST 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 value between 0 and 9 inclusive, specified as a decimal number. This corresponds to the JMS message header field "JMSPriority". If this parameter is not specified, then the context default MUST be used.

4.1.4. replyToName

This property corresponds to the JMS message header field "JMSReplyTo" defined in Section 3.4.6 of the JMS 1.1 specification. As interpreted by the particular variant, this property value specifies the JMS Destination object to which a response message ought to be sent.

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 can 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 (see Section 3.8, "Identifiers", of [JLS] for the specification of a legal Java class name) 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 [JNDI].

4.2.1.4. Additional JNDI Parameters

It is possible that connecting to a JNDI provider requires additional parameters. These parameters can be passed in as custom parameters (see Section 4.4). To identify a custom parameter as JNDI specific, the parameter name needs to start with the prefix "jndi-".

For example, if the JNDI provider requires a parameter named "com.example.jndi.someParameter", you can supply the parameter in the URI as: jndi-com.example.jndi.someParameter=someValue

4.2.2. Example of Performing a JNDI Lookup

To perform a lookup based on a "jndi" variant URI using Java APIs, an application might start by creating 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 (non-normative) 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 sets 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);
```

4.2.2.1. Performing a JNDI Lookup with Custom Parameters

Any parameters with a prefix of "jndi-" MUST be used to set custom properties when establishing a connection to the JNDI provider. The name of the custom property is derived by removing the "jndi-" prefix from the URI parameter name, and the value of the property is the value of the parameter.

For example, the JMS URI...

```
jms:jndi:REQ_QUEUE
  ?jndiURL=file:/C:/JMSAdmin
  &jndiInitialContextFactory
  =com.sun.jndi.fscontext.RefFSContextFactory
  &jndiConnectionFactoryName=CONNFACT
  &jndi-com.example.jndi.someParameter=someValue
```

...instructs the consumer to use the following properties to connect to the JNDI provider:

```
java.naming.provider.url=file:/C:/JMSAdmin
java.naming.factory.initial=
  com.sun.jndi.fscontext.RefFSContextFactory
com.example.jndi.someParameter=someValue
```

4.3. Vendor Destination Names -- Variants "queue" and "topic"

The JMS Session object provides a means to directly access Queues and Topics. Specifically, it has the methods Session.createQueue(String name) and Session.createTopic(String name). These methods can be used to "create" the Java representation of an existing JMS Topic or Queue.

Since the Session interface requires external knowledge about whether a given name relates to a Queue or Topic, rather than introducing one new variant, this section defines two variants. A JMS URI can indicate which of these methods to use by specifying the appropriate variant -- either "queue" or "topic". For example:

```
jms:queue:ExampleQueueName
```

to identify a JMS Queue Destination, and

```
jms:topic:ExampleTopicName
```

to identify a JMS Topic Destination.

JMS only specifies one way to obtain the names used by these APIs. With a JMS Queue or Topic available, an implementation can call Queue.getQueueName() or Topic.getTopicName(), respectively, both of which return a String object. To create a correct corresponding URI, the resulting string MUST use standard URI escape mechanisms so that the resulting characters conform to the production <jms-dest>.

4.3.1. Treatment of replyToName Parameter

When used with the "queue" and "topic" variants, the replyToName parameter, specified in Section 4.1.4, always refers to a name of a JMS Queue to look up via the Session.createQueue() method, or its equivalent. For either variant, if a JMS Topic is instead required as a response Destination, a JMS URI can employ the "topicReplyToName" parameter. This parameter defines a name to look up with the Session.createTopic() method, or its equivalent.

A JMS URI MUST NOT specify both a "topicReplyToName" and a "replyToName" parameter.

4.3.2. Obtaining a Session via JNDI

Using the Session.createQueue() and Session.createTopic() methods assumes that a client program has already obtained a Session object. Where does that Session object come from -- how does a client get it? One way to get a Session is simply to access vendor-specific APIs.

Another way to get a Session object is to simply revert to using JNDI. That is, if a Session is not available to the client from some other context, the "queue" and "topic" variants MAY reuse the URL parameters specified in Section 4.2.1, "JNDI Parameters". Via JNDI, those parameters will identify a ConnectionFactory, which can then be used to obtain a Session object.

Combining the "queue" and "topic" variants with JNDI lookup for an implementation of ConnectionFactory raises an important consideration for JMS URI clients. Once clients employ JNDI for one part of discovering a Destination, they almost certainly could use a vendor-neutral JNDI lookup for a Destination object itself, rather than using vendor-specific means. As a result, clients need to carefully consider whether it makes sense to use JNDI for one part of this problem, without using it for the other.

4.3.3. Limitations of "queue" and "topic"

The JMS specification clearly identifies the two methods on the Session interface as returning vendor-specific names for Destinations. Consequently, users of the "jms" URI scheme ought to carefully consider when these two variants might be employed. If users plan on switching between JMS vendors, they might also need to plan on regenerating resources that contain URIs in this vendor-specific form.

A JMS vendor can provide alternate ways to obtain the names that can be passed to Session.createQueue() and Session.createTopic(). When using names derived from those alternate means, users of this URI specification are encouraged to verify that the obtained names work as expected in all circumstances.

4.4. Custom Parameters

The set of parameters is extensible. Any other vendor- or application-defined parameter can 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 ought to be used. Also see the security discussion below about properties that might 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 Internationalized Resource Identifiers (IRIs) follow the rules of RFC 3987, Sections 3.1 and 3.2. As per Sections 1.2-c. and 6.4 of [RFC3987], all parts of the JMS URI MUST use the UTF-8 encoding when converting to and from the IRI format.

6. Applications/Protocols That Use the JMS URI

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

Some vendors have provided additional products that interoperate with their own SPI implementations. These extensions might 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 [SOAP-JMS]. In addition, the Service Component Architecture Bindings technical committee (TC) [SCA-TC] at OASIS employs the "jms" URI scheme to identify JMS Destinations in [SCA-JMS].

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 three 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 of using JNDI and JMS, interoperability concerns might arise if implementations 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 can 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. Note that while additional variants might seem more flexible, employing variants not defined here might 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 ought to be addressed by 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 variants are the only part that can reasonably be expected to be stable. Other optional JMS configuration and message properties indicated as URI parameters, like "timeToLive", can reasonably be determined by the sender of a message, without affecting the recipient. Insofar as a recipient might 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 field, 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 ought to 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", or "jndiURL". As these parameters can 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 NOT use these URI parameters unless 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 the 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 "Internationalized Domain Name (IDN) homograph attack" [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 "queue" and "topic" variants are subject to the same concerns as the "jndi" variant. In addition, because the Destination names are vendor defined, URIs employing these two variants might employ special characters that significantly change the meaning of the URI. It is possible that the introduction of a single character -- difficult for a human to notice -- might dramatically change the intended meaning of a URI. In situations where this might be an issue, users of this URI are urged to strongly consider the "jndi" variant instead.

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

9.1. URI Scheme Registration

IANA registered the Java Message Service URI scheme described in this document, according to the following scheme registration request, using the template from [RFC4395]:

- URI scheme name: jms
- Status: Provisional
- URI scheme syntax: See Section 3
- URI scheme semantics: See Section 4
- Encoding considerations: See Section 5
- Applications/protocols that use this URI scheme name: See Section 6
- Interoperability considerations: See Section 7

- Security considerations: See Section 8
- Contact: See the Authors' Addresses section
- References: See the References section

9.2. "jms" URI Scheme Registries

Per this URI scheme, IANA has created a registry for possible "variants". IANA can reject obviously bogus registrations.

9.2.1. JMS URI Variants

This registry provides a listing of "jms" URI scheme variants. Variant names beginning with "vnd." are reserved for vendor extensions. Such variants should follow a pattern of vnd.<vendorname>.<label>. The <vendorname> corresponds to the iana-vendor-tag production from [RFC6075], and vendor.<vendorname> must already be registered in the Application Configuration Access Protocol (ACAP) Vendor Subtree. The <label> is chosen by said vendor.

All variant names are to be registered on a first come, first served basis.

Variants must conform to the "jms-variant" production above. Since variants occur in URIs, they ought to be short, and MUST NOT be more than forty characters in length.

This document defines the "jndi", "queue", and "topic" variants initially included in the registry.

9.2.2. "jms" URI Scheme Variant Registration Template

This template describes the fields that must be present to register a new variant for use in a JMS URI.

To: iana@iana.org
Subject: Registration of JMS URI variant name

JMS URI variant name: Variants must conform to the "jms-variant" production above. Since variants occur in URIs, they ought to be short, and MUST NOT be more than forty characters in length.

Description: A description of the purpose of the variant being registered.

Contact Information: Name(s) and email address(es) to contact for more information about this registration.

Description URL: If available, a URL for a document describing the details of how the variant works.

Comments: Any comments the requester thinks are relevant to this request.

Change Controller: Contact information for the person who controls further changes to this variant definition.

9.2.3. Change Control

Once a JMS URI variant registration has been published by IANA, the change controller can request a change to its definition. The change request follows the same procedure as the registration request.

The change controller of a JMS URI variant can pass responsibility for the JMS URI variant to another person or agency by informing IANA; this can be done without discussion or review.

JMS URI variant registrations MUST NOT be deleted; mechanisms that are no longer believed appropriate for use can be marked as obsolete in the Comment field.

In exceptional circumstances, the IESG can reassign responsibility for a JMS URI variant.

The IESG is considered to be the owner of all JMS URI variants that are on the IETF Standards Track.

10. Contributors

The authors gratefully acknowledge the contributions of:

Phil Adams
International Business Machines Corporation
EMail: phil_adams@us.ibm.com

Glen Daniels
WSO2
EMail: glen@wso2.com

Peter Easton
Progress Software
EMail: peaston@progress.com

Tim Frank
Software AG.
EMail: tim.frank@softwareag.com

Lei Jin
BEA Systems, Inc. until March 2007

Eric Johnson
TIBCO Software Inc.
EMail: eric@tibco.com

Vinod Kumar
BEA Systems, Inc. until May 2007

Amelia A. Lewis
TIBCO Software Inc.
EMail: alewis@tibco.com

Roland Merrick
International Business Machines Corporation until June 2009

Mark Phillips
International Business Machines Corporation
EMail: m8philli@uk.ibm.com

Derek Rokicki
Software AG.
EMail: derek.rokicki@softwareag.com

Stephen Todd
International Business Machines Corporation until April 2007

Dongbo Xiao
Oracle Corp.
EMail: dongbo.xiao@oracle.com

Prasad Yendluri
Software AG.
EMail: prasad.yendluri@softwareag.com

11. Acknowledgements

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

12. References

12.1. Normative References

[JLS] Sun Microsystems, Inc., "The Java Language Specification, Third Edition", January 2005, <http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html>.

[JMS] Hapner, M., Burridge, R., Sharma, R., Fialli, J., and K. Stout, "Java Message Service", April 2002, <http://java.sun.com/products/jms/>.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[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.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC6075] Cridland, D., "The Internet Assigned Number Authority (IANA) Application Configuration Access Protocol (ACAP) Vendor Subtrees Registry", RFC 6075, December 2010.

12.2. Informative References

[HOMOGRAPH] "IDN Homograph attack", 2011, <http://en.wikipedia.org/w/index.php?title=IDN_homograph_attack&oldid=416746950>.

[JAVA] Oracle Corporation, "Oracle Technology for Java Developers", 2011, <http://www.oracle.com/technetwork/java/index.html>.

[SCA-JMS] Holdsworth, S. and A. Karmarkar, "Service Component Architecture JMS Binding Specification Version 1.1", November 2010, <http://docs.oasis-open.org/opencsa/sca-bindings/sca-jmsbinding-1.1-spec.html>.

[SCA-TC] "OASIS Service Component Architecture / Bindings (SCA-Bindings) TC", <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sca-bindings>.

[SOAP-JMS] Adams, P., Easton, P., Johnson, E., Merrick, R., and M. Phillips, "SOAP over Java Message Service 1.0", October 2010, <http://www.w3.org/TR/2010/WD-soapjms-20101026/>.

Authors' Addresses

Mark Phillips
International Business Machines Corporation
Hursley House, Hursley Park
Winchester, Hampshire SO21 2JN
United Kingdom

EMail: m8philli@uk.ibm.com

Phil Adams
International Business Machines Corporation
11501 Burnet Rd.
Austin, TX 78758
United States

EMail: phil_adams@us.ibm.com

Derek Rokicki
Software AG.
11700 Plaza America Drive
Reston, VA 20190
United States

EMail: derek.rokicki@softwareag.com

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

EMail: eric@tibco.com