Recommended Communications API

Confidential, Draft
RFC 0043

20 Pages

Abstract

This document compares a number of communication APIs for Java. It recommends the J2ME javax.microedition.io API because it is the smallest abstraction of a streaming and messaging API while still allowing full flexibility.

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies **...** for the Open Services Gateway Initiative, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

0.3 Acknowledgement

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in **Error! Reference source not found.**.

`Source code is shown in this typeface.`

0.5 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	jan 17 2002	Peter Kriens, Peter.Kriens@aQute.se
Choice of javax.microedition.io	4/18/02	Johan Vos johan.vos@acunia.com

Changes after Leuven meeting	08/29/02	Johan Vos johan.vos@acunia.com

1 Introduction

This document describes a number of messaging design alternatives and selectes javax.microedition.io as the base API for OSGi communications API. This API does not support extensions and is therefore extended with an ConnectionFactory mechanism that allows different bundles to extend the schemes.

2 Motivation and Rationale

OSGi environments, in almost all cases, require connectivity to other computers. In Java, there are numerous ways to implement this connectivity. However, different OSGi bundles are deployed on the same environment. It is therefore necessary to have a recommended communication API because otherwise the demands on the deployed environment would be overwhelming.

3 Technical Discussion

Current the following Java communication APIs exist that may be applicable as a recommended OSGi communication API:

1.J2ME, javax.microedition.io package
2.Java Messaging
3.Java Telephony API
4.OSGi communication APIs

3.1 J2ME, javax.microedition.io

The J2ME specification efforts introduced a package for communicating with back end systems. The requirements for this package are very similar to the OSGi requirements:

1.Small footprint
2.Allows many different implementations simultaneously
3.Simple to use
4.Simple configuration

The architecture of the package is oriented around a Connector class. Static methods in this class allow access to communication implementers through a URI. This URI contains a scheme and properties related to that scheme. For example:

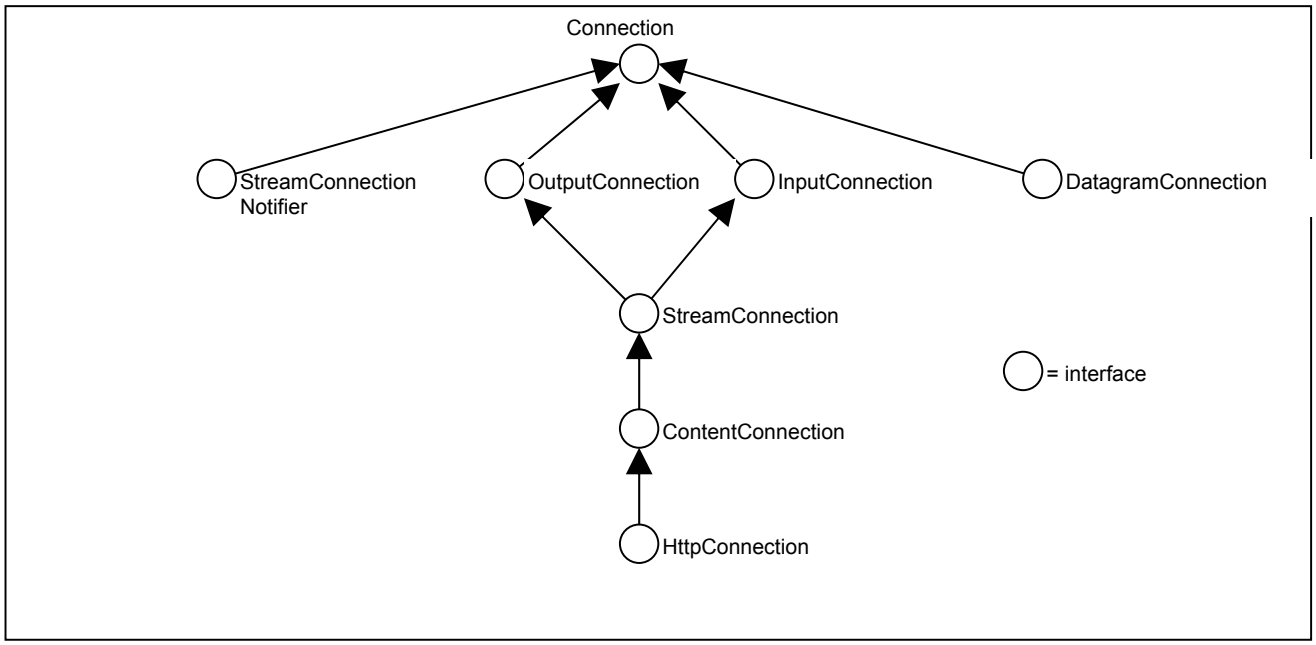
```
sms://+46705950899;expiry=24h;reply=yes;type=9
datagram://:53
socket://www.acme.com:5302
comm://COM1;baudrate=9600;databits=9
file:c:/autoexec.bat
```

The javax.microedition.io API does not prescribe any schemes (the name before the first ':'). It is up to the implementor of this package to provide implementations for different schemes. The Connector class will dispatch a request to some implementation class. In J2ME it is not defined how this dispatching is done.

When the Connector.open() method is called it returns a Connection object. The interfaces that this object implements define the type of the communication session. The following interfaces may be implemented:

1.HttpConnection - A ContentConnection with specific HTTP support.
2.DatagramConnection - A connection that can be used to send and receive datagrams.
3.OutputConnection - A connection that can be used for streaming output.
4.InputConnection - A connection that can be used for streaming input.
5.StreamConnection - A connection that is both input and output.
6.StreamConnectionNotifier - Can be used to wait for incoming stream connection requests.
7.ContentConnection - A StreamConnection that provides information about the type, encoding and length of the information.

Bundles using this approach must indicate to the operator what kind of connection they expect. The operator must then configure the bundle with a scheme and appropriate options that matches the bundles expectation. Well written bundles can adapt to any of the types of Connection interfaces they get returned. E.g. a good bundle should support both Stream as well as Datagram connections in the direction that matches the bundles needs (input/output).
The following code example shows a bundle that sends an alarm message:




```
  public class Alarm {
    String     uri;
    public Alarm(String uri) { this.uri = uri; }

    private void send(byte[] msg) {
      while ( true ) try {
        Connection  connection = Connector.open( uri );
         DataOutputStream    dout = null;
        if ( connection instanceof OutputConnection ) {
          dout = ((OutputConnection) connection).getDataOutputStream();
          dout.write( msg );
        }
        else if ( connection instanceof DatagramConnection ) {
          DatagramConnection dgc = (DatagramConnection) connection;
          Datagram datagram = dgc.newDatagram( msg, 0, msg.length );
          dgc.send( datagram );
        } else {
            error( "no configuration for alarm" );
            return;
        }
        connection.close();
      } catch( Exception e ) {  }
    }
}
```

3.2 JMS, Java Messaging Service

The Java Messaging Service was developed as part of Java Enterprise Edition (J2EE). The API support two messaging models: Publish/Subscribe and Point to Point.

Publish and subscribe is a model where a publisher is sending messages to a potentially large number of subscribers. Point to point is oriented toward persistent or transactional queues.

JMS consists of a number of interfaces and classes that can be implemented by different vendors. The API is extensive and heavily oriented towards transactional queues. It is therefore that JMS has become a part of Java 1.4 Enterprise Edition. A large number of companies have created implementations.

To use JMS, a bundle must obtain a ConnectionFactory. This object is usually configured by the administrator and it is advised to use JNDI for this purpose. A ConnectionFactory is either a TopicConnectionFactory (for publish/subscribe) or a QueueConnectionFactory (for point to point).

The factory is used to create a TopicConnection or QueueConnection respectively. In certain cases this requires he caller to authenticate itself.

This connection object allows the bundle to create sessions. Sessions are, unlike the connections, not synchronized for multiple threads. Any guarantees of delivery order are thus only valid for sessions. A session is used to create QueueSender/QueueReceiver or TopicPublisher/TopicSubscriber objects.

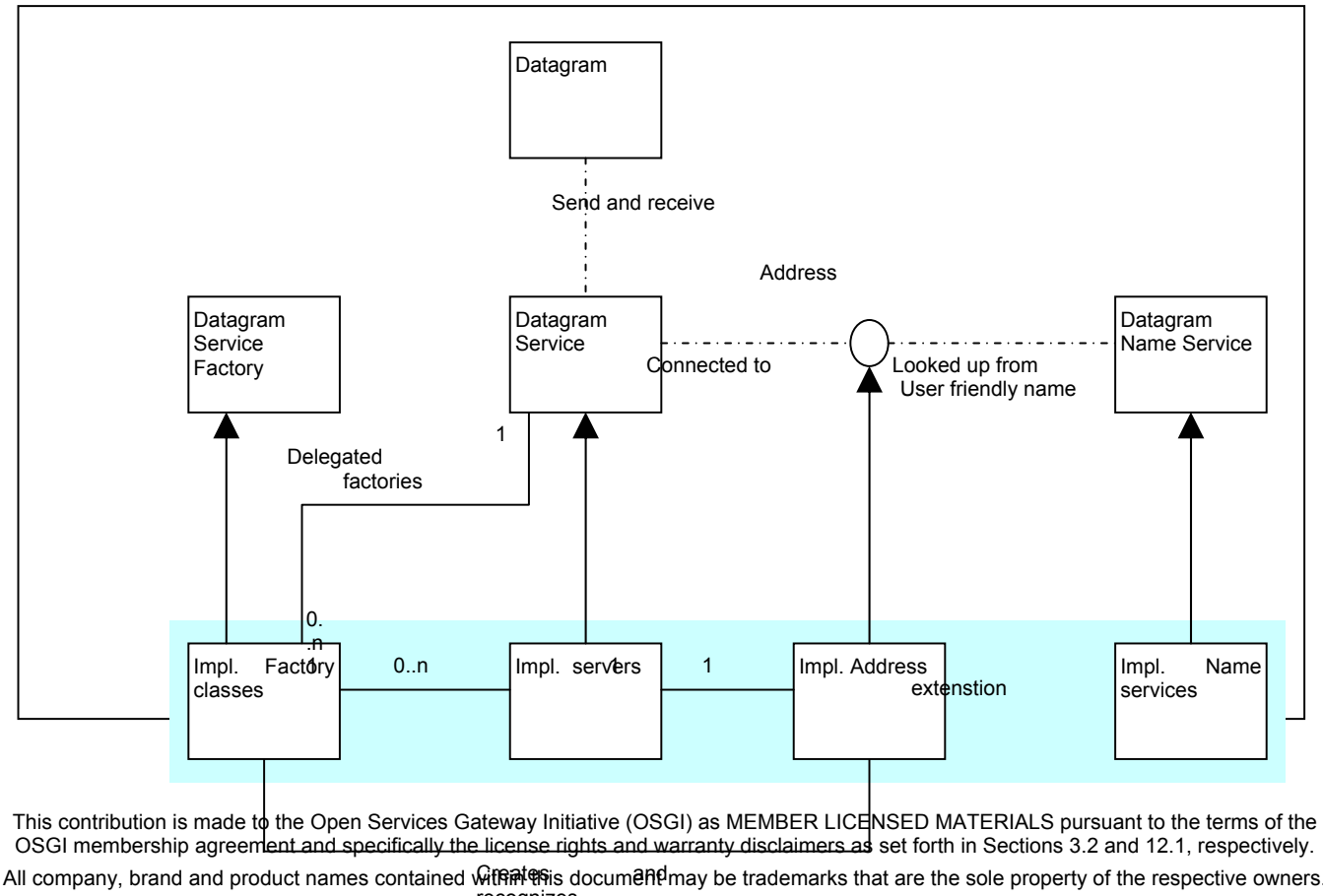
These objects are used to send/receive messages. The QueueReceiver and TopicSubscriber implement the MessageConsumer interface. This interface allows the direct reception of messages or delivery via a listener pattern. The QueueSender and TopicPublisher have methods to send messages.

JMS offers, for convenience, a number of message types: MapMessage, ObjectMessage, BytesMessage, StreamMessage and TextMessage.

The JMS is fully prepared to handle transactions.

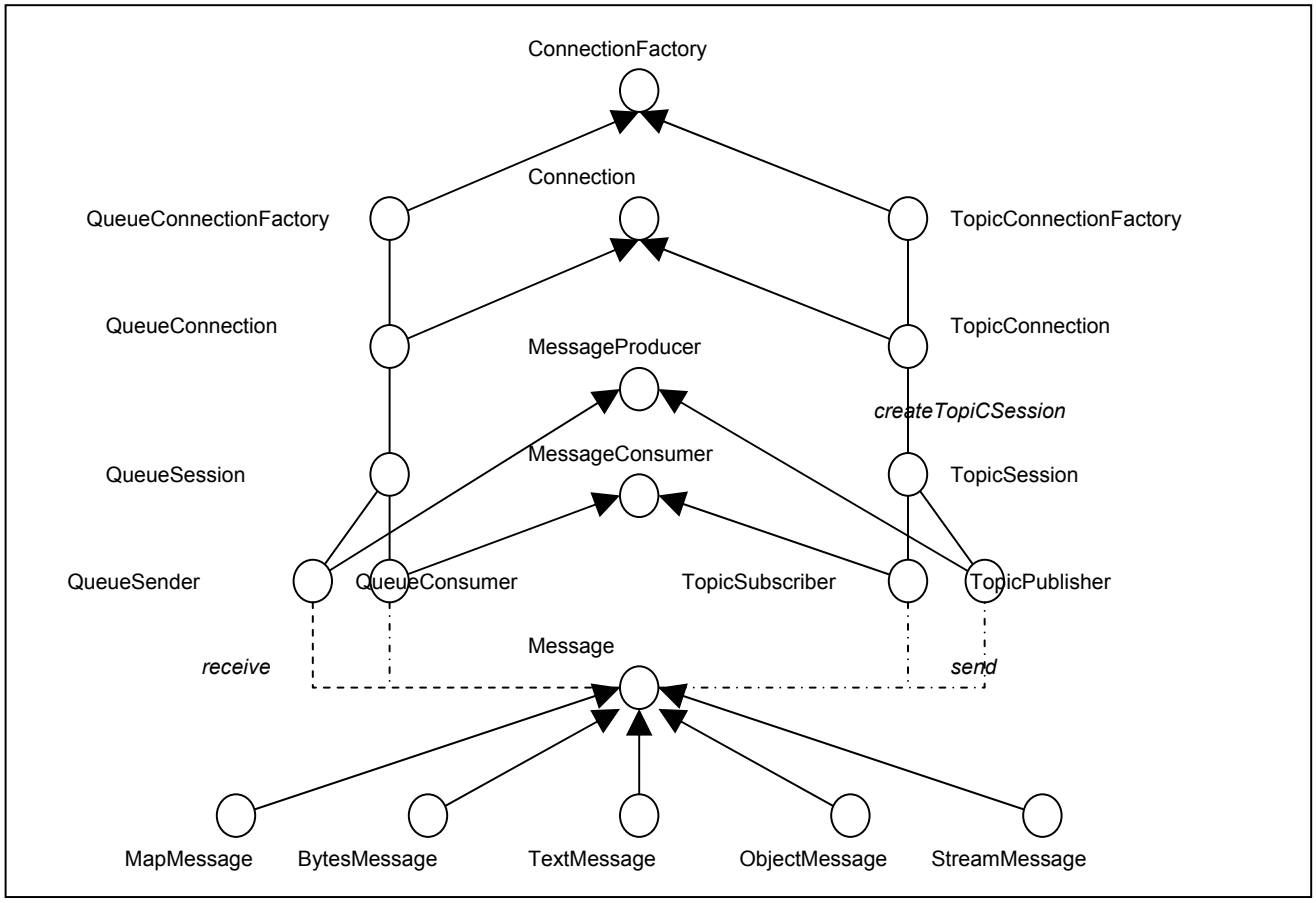
3.3 Java Phone API, javax.net.datagram

The Java Telephony API was developed specifically for running in high end telephones and PDAs. A part of the

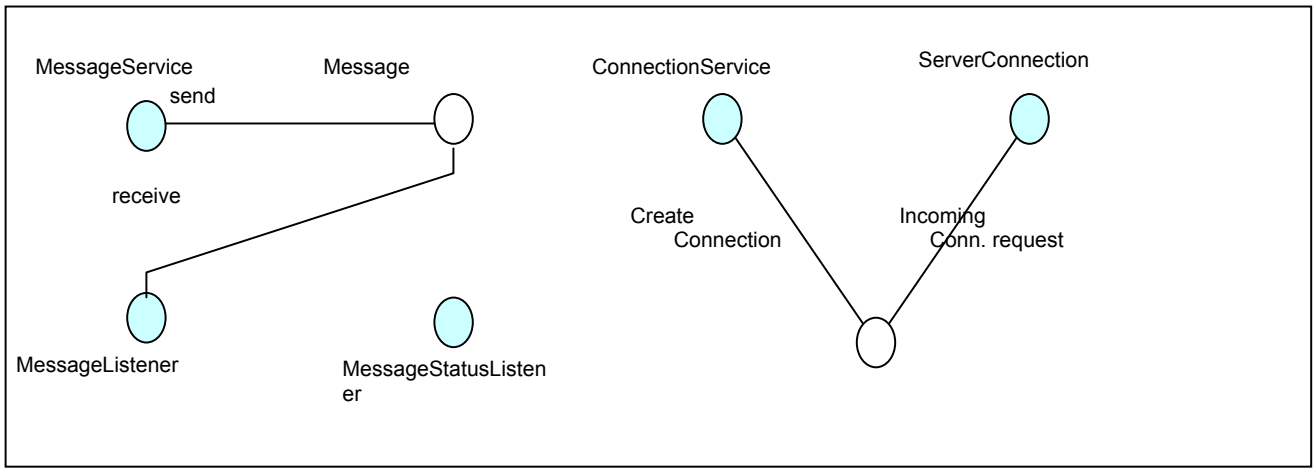


specifications is a datagram service. This datagram service is a simple abstraction of sending a message with bytes (datagram) to some destination or receiving such a datagram from some source. The source and destination are identified with an Address interface. New types of addresses can be introduced by adding DatagramServiceFactory objects via static methods on this base class. The DatagramService class has static methods to parse String based address. It asks all registered DatagramServiceFactory objects one by one if it recognizes the string as a valid address. The first factory that recognizes turns the string in an address object, implementing the Address interface. This Address object can then be used to fetch a DatagramService bound to that address via a static method on this class. This DatagramService can be used to send or receive Datagram objects.

Also provided is a DatagramNameService. Providers can register objects that convert "user friendly" names to actual Address objects.



3.4 OSGi developed Communication APIs



The VEG in the OSGi has developed 2 communication APIs.

The first API is a streaming API that allows bundles to initiate and receive stream connections. The API used the FederatedName class for addressing. It uses the whiteboard approach to receive incoming connections. Outgoing connections were based a server, registered with the Framework, that could create Connection objects.

The second API was a messaging API. It used the whiteboard approach for listening to incoming messages with a MessageListener object. Outgoing messages required getting the message server and sending a new message. There is a Message interface but this needed to be implemented by the senders. The MessageStatusListener interface could be used to listen to progress of the transport of the message (or lack thereof). However, there are no guarantees.

3.5 Comparison

Feature description	JMS	Javaphone	OSGi	J2MEMicroedition
Focus	Enterprise systems, transactional queues and broadcast	High end phones and PDAs. Examples show SMS etc.	Servers that are not always connected	Midrange and higher end phones (CLDC/CDCMIDP). Examples show comm ports, TCP/IP, SMS etc.
Driver	SUN, IBM, BEA	Symbian	-	Motorola, Sun
Reliability guarantees	Program	No	Configuration	Configuration
Expiration	Program	No	No	Configuration
Max Message size visible	No	No	No	Yes

(De)Fragmentation	Yes	Provider	Provider	Provider
Address	Configured by operator in server	Provider (String)	OSGi Namespace	Provider (String)
Options	Headers in message	No	No	In URI
Size of interface package	~30K	~5K	~7K	~10Kb
Estimated min. impl size	~40K - ~100K	Simple SMS: 8K	Ref. Impl. ~90K	Simple SMS: 8K
Supports different providers simultaneously	No	Yes	No	Yes
Configuration points	1. Address on server side 2. Options configured by program	1. Only Address in String (no option mechanisms)	1. Address from bundle 2. Options related to port, but not specified how they are configured	1. Address and options in URI
Estimated nr of implementations	>25	Symbian, ?	Reference impl, Acunia, ?	Sun,
Associated Profile	MIDP when used without XA classes	Personal Java	OSGi Profile (should support MIDP)	MIDP
Supports SMS	Only proprietary	Yes	Yes	Yes
What adaptions can the bundle do	Most features can be found out from the API			
Supports Receive	Yes	Yes	Yes	Yes
Supports Broadcast	Yes (publish/subscribe)	Provider	No	Provider
Supports Streams	No	No	Yes	Yes
Supports OSGi namespace	No	Provider	Yes	Provider
Supports different addressing schemes	No	Yes	No	Yes
Message Types	Bytes, Map, Text, Object, Stream	Datagram (bytes)	Bytes (via Message interface)	Datagram (bytes + DataXStream)
Notifications	Listener model or blocking	Blocking	Listener	Blocking
Defined provider extension mechanism	JNDI	Registration	Framework registry	No
Connection types	Program selects TopicPublisher, TopicSubscriber, QueueSender, QueueReceiver	Always Datagram	Program selects Connection, ServerConnection or Message	Rich hierachy of connection types: HttpConnection, StreamConnection, InputConnection, OutputConnection, DatagramConnection, ContentConnection
Security	Provider specific but	Provider	OSGi Namespace	Provider

	usually extensive		permissions	
Protocol	Provider	Provider	Provider	Provider
HTTP	No	Provider	No	Provider
Routing	Based on central server model that dispatches messages	Provider	Intended OSGi Namespace	Provider
Supports files	No	Provider	No	Provider
Timeouts	Program	No	No	Program
Supports communication ports (serial/parallel)	No	Provider	No	Provider

3.6 Recommendation

This RFC recommends the adoption of the javax.microedition.io API because it fulfills the requirements as laid out in the RFCs best. It allows for small implementations, has the most extensive support for connection types, has minimal configuration requirements and is well supported by the industry.

The javax.microedition.io API does, however, no have an extension mechanism. The Connector class defines how providers are found but there is no associated registration mechanism. It is therefore necessary to provide an OSGi specific mechanism to register and unregister providers.

3.6.1 Provider registration scheme for javax.microedition.io

The ConnectorService class dispatches a URI to a provider. In javax.microedition.io the provider is undefined. Therefore, a service is created that should keep track of the registrations of ConnectionFactory services. The bundle that implements org.osgi.io.ConnectorService must track the service registry for these type of services. If such a service is found, it must be made available through the ConnectorService class. The ConnectionFactory object must therefore supply the scheme(s) that if can handle.

Bundles that need to communicate must call one of the ConnectorService.open methods. The URI argument in the open call will usually come from configuration data and is set by the operator. The associated implementation must then get the scheme from the URI. This scheme is then used to find a ConnectionFactory. If such a provider is found, the URI is forwarded to it where it is parsed further. The ConnectionFactory should then return a Connection object that is returned to the caller.

The caller can then cast the object to the Connection type that it expects and start using it.

If a ConnectionFactory is unregistered, the implementation of the javax.microedition.io service must assure that all Connection objects are closed. Care should be taken that bundles using the ConnectorService service could hang on to objects that were created by provider bundles that are stopped.

Refer to the org.osgi.service.io package for the Javadoc for this service.

3.7 Javadoc

3.8 org.osgi.service.io
Interface ConnectorService

public interface **ConnectorService**

Implementations of this ConnectorService service have to listen to service registrations of `org.osgi.service.io.ConnectionFactory`. When a ConnectionFactory service is registered with the Framework, its `getScheme()` method is called by the implementation of Connector service and the ConnectionFactory service is reqistered as the provider for that particular scheme. Requests for connections using a particular scheme are first delegated to the existing environment. That is, if a handler for scheme `foo` exist in an underlying implementation of `javax.microedition.io.Connector`, this handler is used for handling the request. Other requests are delegated to the particular ConnectionFactory service that registered the scheme.
If more than one ConnectionFactory is registered for a particular scheme, the one with the highest `service.ranking` property is used.
This interface provides the same methods as the `javax.microedition.io.Connector` class. It can not be imposed, however, that all implementations of `javax.microedition.io.Connector` have the above mentioned behavior for handling new registrations of ConnectionFactory services.

The `Connection` instances that are returned by the `open()` methods on this interface are implementations of (subclasses of) `javax.microedition.io.Connection`. A detailed description of the different Connection interfaces can be found at http://jcp.org/aboutJava/communityprocess/review/jsr118

Field Summary

`static int`	**READ** Access mode
`static int`	**READ_WRITE** Access mode
`static int`	**WRITE** Access mode

Method Summary

`javax.microedition.io.Connection`	`open(java.lang.String name)` Creates a `Connection` object to the given address and opens it.
`javax.microedition.io.Connection`	`open(java.lang.String name, int mode)` Creates a `Connection` object to the given address and opens it in the specified mode (`READ`, `WRITE`, `READ_WRITE`).

javax.microedition.io.Connection	open(java.lang.String name, int mode, boolean timeout) Creates a Connection object to the given address and opens it in the specified mode (READ, WRITE, READ_WRITE).
java.io.DataInputStream	openDataInputStream(java.lang.String name) Create a Connection object to the given destination and opens a DataInputStream object ton this Connection object.
java.io.DataOutputStream	openDataOutputStream(java.lang.String name) Create a Connection object to the given destination and opens a DataOutputStream object on this Connection object.
java.io.InputStream	openInputStream(java.lang.String name) Create a Connection object to the given destination and opens an InputStream object on this Connection object.
java.io.OutputStream	openOutputStream(java.lang.String name) Create a Connection object to the given destination and opens an OutputStream object on this Connection object.

Field Detail

3.8.1 READ

```
public static final int READ
```

Access mode

3.8.2 WRITE

```
public static final int WRITE
```

Access mode

3.8.3 READ_WRITE

```
public static final int READ_WRITE
```

Access mode

Method Detail

3.8.4 open

```
public javax.microedition.io.Connection open(java.lang.String name)
                                      throws java.io.IOException,

javax.microedition.io.ConnectionNotFoundException
```

Creates a `Connection` object to the given address and opens it.

Parameters:

`name` - The address of the destination, in URI format.

Returns:

A new `Connection` object to the required destination

Throws:

`java.lang.IllegalArgumentException` - if a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` - if the `Connection` object can not be made or if no handler for the requested scheme can be found.

`java.io.IOException` - if some other kind of I/O error occurs.

3.8.5 open

```
public javax.microedition.io.Connection open(java.lang.String name,
                                             int mode)
                                      throws java.io.IOException
```

Creates a `Connection` object to the given address and opens it in the specified mode (`READ`, `WRITE`, `READ_WRITE`). Note that some modes do not make sense in some connection typs (e.g. reading from a printer).

Parameters:

`name` - The address of the destination, in URI format.

`mode` - The access mode for the `Connection` object.

Returns:

A new `Connection` object to the required destination

Throws:

`java.lang.IllegalArgumentException` - if a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` - if the `Connection` object can not be made or if no handler for the requested scheme can be found.

`java.io.IOException` - if some other kind of I/O error occurs.

3.8.6 open

```
public javax.microedition.io.Connection open(java.lang.String name,
                                             int mode,
                                             boolean timeout)
                                      throws java.io.IOException
```

Creates a `Connection` object to the given address and opens it in the specified mode (`READ`, `WRITE`, `READ_WRITE`). Note that some modes do not make sense in some `Connection` objects (e.g. reading from a printer).

Parameters:

`name` - The address of the destination, in URI format.

`mode` - The access mode for the `Connection` object.

`timeout` - When the requestor wants to get an `Exception` thrown when a `Connection` object can not be made within a given time, this flag has to be true

Returns:

A new `Connection` object to the required destination

Throws:

`java.lang.IllegalArgumentException` - if a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` - if the `Connection` object can not be made or if no handler for the requested scheme can be found.

`java.io.IOException` - if some other kind of I/O error occurs.

3.8.7 openInputStream

```
public java.io.InputStream openInputStream(java.lang.String name)
                                    throws java.io.IOException
```

Create a `Connection` object to the given destination and opens an `InputStream` object on this `Connection` object.

Parameters:

`name` - The address of the destination, in URI format.

Returns:

An `InputStream` object

Throws:

`java.lang.IllegalArgumentException` - if a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` - if the `Connection` object can not be made or if no handler for the requested scheme can be found.

`java.io.IOException` - if some other kind of I/O error occurs.

3.8.8 openDataInputStream

```
public java.io.DataInputStream openDataInputStream(java.lang.String name)
                                                    throws java.io.IOException
```

Create a `Connection` object to the given destination and opens a `DataInputStream` object ton this `Connection` object.

Parameters:

`name` - The address of the destination, in URI format.

Returns:

A `DataInputStream` object.

Throws:

`java.lang.IllegalArgumentException` - if a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` - if the `Connection` object can not be made or if no handler for the requested scheme can be found.

`java.io.IOException` - if some other kind of I/O error occurs.

3.8.9 openOutputStream

```
public java.io.OutputStream openOutputStream(java.lang.String name)
                                             throws java.io.IOException
```

Create a `Connection` object to the given destination and opens an `OutputStream` object on this `Connection` object.

Parameters:

`name` - The address of the destination, in URIL format.

Returns:

An `OutputStream` object.

Throws:

`java.lang.IllegalArgumentException` - if a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` - if the `Connection` object can not be made or if no handler for the requested scheme can be found.

`java.io.IOException` - if some other kind of I/O error occurs.

3.8.10 openDataOutputStream

```
public java.io.DataOutputStream openDataOutputStream(java.lang.Stringname)
                                                     throws java.io.IOException
```

Create a `Connection` object to the given destination and opens a `DataOutputStream` object on this `Connection` object.

Parameters:

`name` - The address of the destination, in URI format.

Returns:

A `DataOutputStream` object.

Throws:

`java.lang.IllegalArgumentException` - if a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` - if the `Connection` object can not be made or if no handler for the requested scheme can be found.

`java.io.IOException` - if some other kind of I/O error occurs.

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: INNER | Error! Hyperlink reference not DETAIL: Error! Hyperlink reference not

3.9 org.osgi.service.io
Interface ConnectionFactory

public interface **ConnectionFactory**

A service interface that is used to extend the schemes in `javax.microedition.io` An ConnectionFactory service must be detected by implementations of the `javax.microedition.io` package. The methods in this interface allow the implementation to provide new schemes.

Field Summary

`static java.lang.String`	**IO_SCHEME** A service registration property that reflects the scheme that this ConnectionFactory service implements.

Method Summary

javax.microedition.io.Connection	createConnection(java.lang.String uri, int mode, boolean timeouts) Create a new Connection object.

Field Detail

3.9.1 IO_SCHEME

```
public static final java.lang.String IO_SCHEME
```

A service registration property that reflects the scheme that this ConnectionFactory service implements. The type must be `String` and and the value is "io.scheme".

Method Detail

3.9.2 createConnection

```
public javax.microedition.io.Connection createConnection(java.lang.String uri,
                                                         int mode,
                                                         boolean timeouts)
                                                  throws java.io.IOException
```

Create a new `Connection` object. The implementation of the `Connector` class must parse the URI into the host, port and options. These values are placed in the options argument. The key is the name of the option This method may be called from different threads simultaneously.

Parameters:

`uri` - The full URI that is passed to the `Connector.open` method

`mode` - The mode parameter of the `Connector.open` method

`timeouts` - The timeouts parameters of the `Connector.open` method

Returns:

A new `Connection` object.

Throws:

`java.io.IOException` - When the connection could not be created

3.8. License issues

Since javax.microedition.io is not a part of the minimal execution environment of OSGi, it cannot be assumed to be present on the platform. The provider of an implementation of org.osgi.service.io is encouraged to package the javax.microedition.io interfaces within his bundle.

4 Security Considerations

ConnectionFactory services will perform system critical functions for many different applications. This implies that ServicePermission[ConnectionFactory,REGISTER] must be limited to those bundles that are trusted to supply this service. ServicePermission[ConnectionFactory,GET] must be limited to only one bundle: The implementation of the javax.microedition.io package.

5 Document Support

5.1 References

[1].Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2].Javax.microedition. http://java.sun.com/j2me/docs/pdf/cldcapi.pdf

[3].Java Messaging Service. http://java.sun.com/products/jms/index.html

[4].Java Telephony API. http://java.sun.com/products/javaphone/download.html

[5].OSGi messaging Service. RFC 25 Messaging Service. http://membercvs.osgi.org/rfcs/rfc0025

5.2 Author's Address

Name	Johan Vos
Company	Acunia
Address	
Voice	+32 16 310020
e-mail	
Name	Peter Kriens
Company	aQute
Address	Finnasandsvägen 22

Voice	+46 300 39800
e-mail	Peter.Kriens@aQute.se

5.3 Acronyms and Abbreviations

5.4 End of Document