# RFC 24 − Connection−oriented Based Communication

Confidential, Draft

14 Pages

## Abstract

Services that want use Input/Output Streams for communicating with other services, gateways or domains can use the Connection−based Communication API for obtaining these Input/Output Streams.

# 0 Document Information

## 0.1 Table of Contents

## 0.2 Status

This document specifies a Connection−based Communication Service 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 Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997..

`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 | 07/02/01<br>07/10/01 | Initial draft by Johan Vos, johan.vos@acunia.com<br>Second draft by Jean−Philippe Cattoor, jean−philippe.cattoor@acunia.com |
|  | 7/30/01 | Third draft by Johan Vos, johan.vos@acunia.com |

# 1 Introduction

In many cases, services running on an OSGi gateway need provide a streaming, synchronous channel to exchange data in a reliable way in a context of client, server. A service programmer should be able to use a standardized service in order to communicate in a synchronous way. However, there are different communication mechanisms, each with their own application areas. This document addresses the need of a standardized connection−oriented communication API.

# 2 Motivation and Rationale

The standardized connection API abstract the synchronous reliable communication and intend to relieve its clients of the transport medium used (TCP/WAP...).

The result is:

* Smaller and simpler services

* All services can benefit from advanced features available.

Many implementation of that service might be available on the platform ; providing different facilities such as security (authentication, integrity protection, encryption) or QoS. Anyhow, it is not insured they will be available by default.

Those implementations will provide configuration parameters for requesting security, QoS? It is up to the service provider to download a connection service that integrate security and QoS and up to the configuration administration to make those features available on the platform.

The Connection−based Communication API looks similar to the Socket and ServerSocket classes in the java.io package. However, the latter classes can not be registered with the Framework as OSGi services. This API allows for multiple implementations running on the same platform and leveraging the capabilities of the OSGi Framework (eg registration and lookup). Furthermore, the Connection−based Communication API uses the FederatedAddress and FederatedAddressPermission as discussed in RFC 22.

# 3 Technical Discussion

The connection API is designed to provide a simple, transport independent socket service from which security might be part of. This section discusses the requirements placed on the connection API, motivates the design decisions made and some of the implications that they have.

The standard protocol supported by default will be TCP/IP. This protocol has been chosen for interoperability between different connection service providers.

## 3.1 Overview of the API Specification

This section gives a brief overview of the connection−oriented API. The full details of the API is presented in Section 4.

The connection API contains three interfaces:

- ConnectionService: an interface for creating Connection, ServerConnection,
- Connection: an interface representing a connection between two endpoints,
- ServerConnection: an interface listening for incoming connections on a specified endpoint.

A Connection is a connection between two endpoints. Regular InputStream and OutputStream can be obtained from this Connection.

Basically, the philosophy follows the one of java.net.socket except that the creating of ServerConnection and Connection is only allowed via the factory. Writing or reading bytes is done via the write and read methods available respectively on OutputStream and InputStream classes that can be obtained on the Connection class.

Services that expect entering connections have to create ServerConnection while on the other side services that wants to connect to "server" application have to create Connection.

## 3.2 Design Decisions and their Implications

### 3.2.1 Simplicity and Minimalism

The overall design philosophy behind the connection service is simplicity.

It also follows the same design principles as messaging except for the Whiteboard approach that is not appropriate for synchronous connections where InputStream and OutputStream classes are preferred to listeners. Also it is not possible to provide a unique service for synchronous communication since no borders, no messages exists at the transport layer.

In consequence to the following of the same principles as the messaging AP, the API design is a very simple and minimal API for synchronous peer−to−peer communication. This design philosophy has influenced all other design decisions to a rather large extent.

### 3.2.2 Addressing

Addressing is done using the FederatedAddress, described in [3]. The whole purpose of the FederatedAddress is to enable communication between OSGi services, which is exactly what this RFC addresses.

### 3.2.3 Secure Communication and other Advanced Delivery Features

The API does not contain any methods for controlling how the stream is transported or delivered, i.e., whether it is encrypted, digitally signed, compressed, or with what level of urgency (priority) it is treated. The connection API such as the messaging API have been designed with security and other advanced delivery features in mind but they have not been explicitly integrated in the API in order to keep them simple.

A connection service can nonetheless support secure communication and other advanced delivery and transport features but the client application cannot use the connection API to control any of the parameters involved in the establishment of that connection. Instead, these must be controlled using a separate API or through configuration of the connection service.

Leaving control of security and other advanced delivery features to the configuration of the connection−based service has several advantages:

- It gives the Gateway Operator or other entity that manages the configuration of the Service Gateway explicit control over which communication is encrypted, digitally signed, compressed, etc. That entity thus has the possibility to setup the configuration so that all bytes sent to and from ports used by sensitive services, e.g., billing services, are digitally signed and encrypted, that all communication routed using a certain low bandwidth transport is compressed, that all communication sent to the back ends alarm handling system is routed with a certain priority, etc.

- It relieves the burden of bundle programmers since they do not need to decide whether to send data using certain delivery features. At the same time, it does not deprive bundles of the possibility to send data on different ports that are configured with different transport and delivery features. For instance, a bundle can use two different ports for its communication, one for high priority communication and one for normal priority. Depending on the functionality and configuration of the connection−based service, communications on these ports may be treated equally or with different priority.

### 3.2.4 Configuration

As described above, configuration will control the connection−oriented implementations available on OSGI gateways for example secure, optimized (QoS) connection−oriented implementation.

In order to do so a ConnectionFactory has to be implemented together with the connection−oriented service implementation. This ConnectionFactory will have to implement the ManagedServiceFactory and it is up to that class to be able to create different ConnectionServices depending on the Configuration Objects existing for that factory (cf. Configuration Management RFC).

In order for a service to retrieve the right Connection service, it will have to use the method getServiceReferences() on the bundle context where the filter is set to for example (secure=authenticated).

Supported Configuration objects will not be standardized here. It is left open if it needs to be standardized.

Anyway, there is a common need for defining configuration properties like "security={none, authenticated, encrypted, integrity, secure}", "priority={1,2,...}" or "transport={reliable,unreliable}",...

### 3.2.5 Security and Control of Usage

The FederatedAddressPermission class defined in the naming API enables connection−based services to perform fine−grained control over how a given bundle uses it. The security policy that assigns FederatedAddressPermissions to bundles provides an explicit control over the ports used for receiving byte array and also the destination address and port used when bytes are sent. The security issues addressed with FederatedAddressPermission are:

- Eavesdropping. A bundle will not receive any bytes on a port for which it does not have the FederatedAddressPermission to listen on.

- Spoofing. A bundle cannot send bytes specifying a source port for which it does not have the FederatedAddressPermission to send from.

- Unauthorized use of resources. A bundle cannot send bytes to a destination host or port unless it has the FederatedAddressPermission to send to that destination host and port.

# 4 API Specification

## 4.1 org.osgi.service.comm.connection
## Interface ConnectionService

public interface **ConnectionService**

The ConnectionService allows the creation of Connections and ServerConnections. A Connection can be compared with a `java.net.Socket` while a ServerConnection can be compared with a `java.net.ServerSocket`. However, the addressing of Connections is different from the addressing of Sockets. Furthermore, a ConnectionService is registered with the OSGi Framework, and multiple implementations are allowed to co−exist on the same platform.

## Method Summary

| | |
|---|---|
| `Connection` | **`createConnection`**`(org.osgi.service.comm.FederatedAddress destination)`<br>Create and open a connection to a (remote) endpoint. |
| `ServerConnection` | **`createServerConnection`**`(org.osgi.service.comm.FederatedAddress source)`<br>Create a ServerConnection at the port retrieved from this FederatedAddress. |

## Method Detail

### 4.1.1 createConnection

```
public                                                          Connection
createConnection(org.osgi.service.comm.FederatedAddressdestination)
                                 throws java.io.IOException
```

Create and open a connection to a (remote) endpoint.

**Parameters:**

`destination` − the address of the endpoint. A ServerConnection should already be opened at this endpoint.

**Returns:**

an opened Connection Object

**Throws:**

java.io.IOException − − if an I/O errors occurs during the connection to the destination.

java.lang.SecurityException − − if a security manager that supports permissions and the caller does not have the required `FederatedAddressPermission` with action `connect`

### 4.1.2 createServerConnection

```
public                                                        ServerConnection
createServerConnection(org.osgi.service.comm.FederatedAddresssource)
                                        throws java.io.IOException
```

Create a ServerConnection at the port retrieved from this FederatedAddress.

**Parameters:**

`source` − the address of the required ServerConnection. If the Service Gateway Domain is not specified, it is assumed that the ServerConnection should be established at the `localdoman`. If the Service Gateway Host is not specified, it is assumed that the ServerConnection should be established at the `localhost`.

**Returns:**

an opened ServerConnection Object

**Throws:**

java.io.IOException − − if a property "port" is not set in the service referenced.

java.lang.SecurityException − − if a security manager that supports permissions and the caller does not have the required `FederatedAddressPermission` with action `accept`.

## 4.2 org.osgi.service.comm.connection Interface Connection

public interface **Connection**

A Connection is a connection between two endpoints. Regular InputStream and OutputStream can be obtained from this Connection.

| **Method Summary** | |
|---|---|
| `void` | `close()`<br>Closes this Connection. |
| `org.osgi.service.comm.FederatedAddress` | `getDestinationAddress()`<br>Returns the destination federated address to which the connection is bound. |
| `java.io.InputStream` | `getInputStream()`<br>Returns the input stream for this Connection. |
| `java.io.OutputStream` | `getOutputStream()`<br>Returns the output stream for this Connection. |

| | |
|---|---|
| `org.osgi.service.comm.FederatedAddress` | `getSourceAddress()`<br>Returns the source federated address to which the connection is bound. |

# Method Detail

### 4.2.1 getInputStream

```
public java.io.InputStream getInputStream()
                                   throws java.io.IOException
```

Returns the input stream for this Connection.

**Throws:**

java.io.IOException − if an I/O error occurs when obtaining the input stream.

---

### 4.2.2 getOutputStream

```
public java.io.OutputStream getOutputStream()
                                     throws java.io.IOException
```

Returns the output stream for this Connection.

**Throws:**

java.io.IOException − if an I/O error occurs when obtaining the output stream.

---

### 4.2.3 getSourceAddress

```
public org.osgi.service.comm.FederatedAddress getSourceAddress()
                                                        throws java.io.IOException
```

Returns the source federated address to which the connection is bound.

**Returns:**

the source federated address to which the connection is bound.

---

### 4.2.4 getDestinationAddress

```
public org.osgi.service.comm.FederatedAddress getDestinationAddress()
                                                                          throws
java.io.IOException
```

Returns the destination federated address to which the connection is bound.

**Returns:**

the destination federated address to which the connection is bound.

### 4.2.5 close

```
public void close()
           throws java.io.IOException
```

Closes this Connection.

**Throws:**

java.io.IOException − if an I/O error occurs when closing this Connection.

## 4.3 org.osgi.service.comm.connection
## Interface ServerConnection

public interface **ServerConnection**

A ServerConnection can be compared with a ServerSocket. It listens for incoming connections on a specified endpoint. A ServerConnection is created by a ConnectionService.

## Method Summary

| | |
|---|---|
| Connection | **accept**()<br>Listens for incoming connections towards this endpoint and accepts it. |
| void | **close**()<br>Closes this ServerConnection. |
| org.osgi.service.comm.FederatedAddress | **getLocalAddress**()<br>Return the Federated Address on which it is listening for connections. |

## Method Detail

### 4.3.1 accept

```
public Connection accept()
                  throws java.io.IOException
```

Listens for incoming connections towards this endpoint and accepts it. This methods blocks until a Connection is detected.

**Throws:**

java.io.IOException − if an I/O error occurs while accepting a Connection

### 4.3.2 getLocalAddress

```
public org.osgi.service.comm.FederatedAddress getLocalAddress()
```

Return the Federated Address on which it is listening for connections.

**Returns:**

FederatedAddress on which it is listening on.

### 4.3.3 close

```
public void close()
           throws java.io.IOException
```

Closes this ServerConnection.

**Throws:**

java.io.IOException − if an I/O error occurs while closing the ServerConnection

# 5 Document Support

## 5.1 References

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

## 5.2 Author’s Address

| | |
|---|---|
| Name | Johan Vos |
| Company | ACUNIA |
| Address | Vanden Tymplestraat 35, 3000 Leuven, Belgium |
| Voice | +32 16 310020 |
| e−mail | johan.vos@acunia.com |

| | |
|---|---|
| Name | Jean−Philippe Cattoor |
| Company | ACUNIA |
| Address | Vanden Tymplestraat 35, 3000 Leuven, Belgium |
| Voice | +32 16 310020 |
| e−mail | Jean−philippe.cattoor@acunia.com |

## 5.3 Acronyms and Abbreviations

## 5.4 End of Document