
# RFC 44 OSGi Services for Content and Connection Handlers

Confidential, Draft
RFC 44

6 Pages

## Abstract

This RFC defines a mode of operation for using the OSGi Service Registry as a way of adding handlers for URLStreams and Content. This avoids the problem of allowing Bundles, which may be updated and removed, the ability to register URLStreamHandler and ContentHandler factories, but still provides the desired functionality. A side benefit is that the functionality is provided to all bundles, not just a single bundle that registers a factory.

# 0 Document Information

## 0.1 Table of Contents



## 0.2 Status

This document specifies **RFC 44: OSGi Services for Content and Connection Handlers** for the Open Services Gateway Initiative, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

## 0.3 Acknowledgement

This RFC is made possible by the developers of the java.net API.

## 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 [1].

`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 | 2/28/02 | *Initial Revision* |
| 1.00B | 4/8/02 | Changed property names. Proxies for absent handlers will always throw exceptions. Standard ranking for multiple services. Cannot override built−in handlers. |
| 1.00C | 5/16/02 | Referenced JAF. Changed to always throw an exception if there is no service providing the handler. Added system property to locate built−in handlers using the algorithm used in the Sun JVM. Added reference to RFC 43. Cited EE RFCs. |

# 1 Introduction

Java provides the `java.net.URL` class which is used by the OSGi framework as well as many of the bundles that run on the framework. The major benefits of using the URL class is the ease of which a URL string is translated into a protocol request and the response is rendered as the proper content. Historically the URL class is mostly used with the HTTP protocol and text or image contents.

The URL class was designed to be extensible so that new protocols and content types could be added to the URL class using handler factories. These new handlers allow existing applications to use new protocols and contents as easily as the built−in ones.

This RFC proposes a way for new handlers to be dynamically registered in the OSGi service registry and exposed to applications through the Java URL class. OSGi applications are encouraged to use the API outlined in RFC 43: Recommended Communications API [2] instead of the URL class since its behavior is more tuned to the OSGi environment.

# 2 Motivation and Rationale

There are two technical problems with just using the existing Java facilities for extending the handlers used by the `java.net.URL` class. The first of these is that only one handler factory can be registered, and registration is a one−time event. `Java.net.URL.setURLStreamHandlerFactory` is used to register a `URLStreamHandlerFactory` that will be used to construct protocol handlers for the new protocols to be added

to URL. `Java.net.URLConnection.setContentHandlerFactory` is used to register a new `ContentHandlerFactory` that will construct new content handlers to process new content types. Both these factories can only be registered once. There is not a way to unregister a factory or register a replacement factory.

The other technical problem arises from the implementation of the URL class. When a protocol or content type is used that it has not seen before it will request a handler from the appropriate factory and then put that handler in a hash table keyed by its protocol or type. This means that once a handler has been constructed for the URL class it cannot be removed or replaced by a new handler.

Both of these problems reflect directly on the OSGi operating model which allows a bundle to go through different life−cycle stages which involve exposing services, removing services, updating code, replacing services provided by one bundle with services from another, etc. We would like this same kind of behavior with the URL class.

It should be noted that JavaBeans Activation Framework (JAF) [3] also has a facility for adding handlers, however, those handlers do not get propagated to the `java.net.URL` class and also suffer from the problems noted above.

# 3 Technical Discussion

To obtain the desired behavior we must register factories that will not go away. The most obvious place to put such a factory is in the OSGi framework. The other place could be in a bundle that the framework knows not to remove or stop. Either way the code that registers the factories cannot be removed from the JVM unless the JVM is restarted. In addition the framework or unremovable bundle must also have concrete proxy classes that extend `java.net.URLStreamHandler` and `java.net.ContentHandler`.

When the two factories are registered, the service registry must be watched for the registration of `java.net.URLStreamHandler`'s and `java.net.ContentHandler`'s. Each of the services will have a property named `url.handler.protocol` and `url.content.mimetype` respectively to specify which protocols and content types can be handled by the registered services. The value of the `url.handler.protocol` property will be an array of protocol names (Strings). The value of the `url.content.mimetype` property will be an array of mime types names (Strings) in the form `type/subtype`.

## 3.1 Constructing a handler

When the factory is called to construct a handler, it will see if a registered service can handle the requested protocol or content type. If such a service exists, it will construct a concrete proxy object with the desired protocol or content type string and return it. The concrete proxy object will then forward any method invocations to the service whose service property indicates that it can handle the given protocol or content type.

If two services can handle the same protocol or content type, the proxy will choose the service to be used based upon service ranking as described in section 2.9.6 of Release 2 of the OSGi Service Platform. Even when the service becomes unregistered, methods on the proxy class will still get invoked. If there is another bundle providing a service to handle the protocol or content type, the proxy will start using the new service.

## 3.2 Absent handlers

When a handler is registered in the Service Registry and a URL object is created that uses the handler, a proxy will be constructed and put into the hash table in the URL class. If the handler service is unregistered and there is no other handler that can service the protocol or content type, the proxy must handle subsequent requests from the URL class on its own until another service is registered for that protocol or content type. When this happens the proxy class MUST throw an `java.net.MalformedURLException` if possible or throw `java.lang.IllegalStateException` otherwise..

Authors of handlers SHOULD also ensure that object they return from their handler methods behave correctly when the bundle they belong to gets stopped. Since it is possible that the classloader for a stopped bundle may not have access to the class files for that bundle once it is stopped, the authors SHOULD throw the above exceptions as appropriate.

## 3.3 Built−in handlers

Java implementations have some handlers built−in and ways to add handlers on the classpath. If the factory does not return a handler for a built−in handler, it will not get called later when a service registers a handler for a protocol or content type that has already been provided using a built−in handler. Thus it is not always guaranteed that a registered handler will be used.

Built−in handlers SHOULD take priority over handlers from the service registry. This will result in much more deterministic behavior when running on a particular platform. The built−in handlers from the OSGi Execution Environments [5][4] MUST always be used. To facilitate the discovery of built−in handlers the algorithm described in the following paragraph MUST be used.

If the system property `osgi.url.handler.packages` or `osgi.url.content.packages` are defined they will be used to located built−in handlers. Each property is a comma separated list of package names. The package names serve as package prefixes that are prepended to the converted handler name to identify a class to handle the protocol. A converted handler name for a protocol is formed by appending the string ".Handler" to the protocol name. A converted handler name for a content handler is formed by changing the '/' in the mime−type to a '.' and any non−alphanumeric characters to '_'.

# 4 Security Considerations

The ability to add protocol and content handlers makes it possible to directly affect the behavior of a core JVM class. The URL class is widely used by network applications and may be used by the OSGi framework itself. Thus, the ability to register handlers should not be given out lightly. The registration of services is governed by the `org.osgi.ServicePermission`, which can restrict access to the registration and use of a service of a given type. The two types mentioned in this RFC are `java.net.URLStreamHandler` and `java.net.ContentHandler`. Only trusted bundles should be allowed to register these services. Since these services will be made available, using the mechanisms outlined in this RFC, to other bundles through the `java.net.URL` class, it is advisable to restrict the use of these services (the GET permission) to the bundle that registers the handler factories. This avoids circumventing the permission checks done by the `java.net.URL` by using the handler services directly.

# 5 Document Support

## 5.1 References

1: Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, 1997
2: Johan Vos and Peter Kriens, RFC 43: Recommended Communications API, 2002
3: Bart Calder, Bill Shannon, JavaBeans Activation Framework Specification Version 1.0a, 1999
4: David Bowen and Kirk Chen, RFC 26 − An Execution Environment (J2ME CDC/FP), 2001
5: Rick DeNatale and BJ Hargrave, RFC 39 − Small Execution Environment, 2001

## 5.2 Author's Address



| | |
|---|---|
| Name | Benjamin Reed |
| Company | IBM |
| Address | 650 Harry Road, San Jose, CA 95120 |
| Voice | 408 927−1811 |
| e−mail | breed@almaden.ibm.com |



## 5.3 Acronyms and Abbreviations

## 5.4 End of Document