RFC 182 - REST Interface for OSGi

Draft

38 Pages

Abstract

This RFC describes a REST interface for managing OSGi framework instances, e.g., in cloud computing setups.

0 Document Information

0.1 Table of Contents

0.2 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 11.1.

`Source code is shown in this typeface.`

0.3 Revision History

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

Revision	Date	Comments
Initial	01/20/12	Initial draft
2nd draft	09/08/12	Update for the Basel F2F
3rd draft	09/20/12	Update based on the discussions from the Basel F2F
4th draft	01/28/13	Update based on the Orlando F2F discussions
5th draft	02/25/13	Update based on the Austin F2F discussions
6th draft	06/07/13	Update for the Palo Alto F2F
7h draft	09/10/13	Update for the Hursley F2F

1 Introduction

Cloud computing is a continuing trend in the IT industry. As discussed in RFP 133 [3]., OSGi appears to be an ideal base for building scalable and dependable applications for the cloud. One of the possible scenarios for OSGi to be successfully applied to cloud computing is to use it in a Platform as a Service spirit. Users write their bundles and can deploy them to their own OSGi instance running in the cloud. This, however, requires the platform provider to expose the OSGi management API to the end user and make them available through a network protocol.

One of the popular approaches in cloud computing to remote communication is the use of RESTful web services. This document discusses the technical background of REST, the applicability to managing OSGi frameworks through a REST-style API, and proposes a concrete protocol.

2 Application Domain

Cloud Computing is a major IT trend; possibly qualifying as a real paradigm shift whose eventual impact may exceed that of the move to client server computing in the 1980's. *Cloud Computing* service are already extensively used to provide cost effective third party hosting for some categories of traditional application; differing only in the granularity of utilization and subsequent payment from the previous utility compute service providers. However, for applications architected in the appropriate manner, *Cloud Computing* also offers the promise of massive 'just-in-time' resource elasticity

The JCP is preparing for Cloud support in future versions of JavaSE and JavaEE. JSR 342 (http://jcp.org/en/jsr/detail?id=342), the JavaEE 7 JSR, makes cloud its main theme, mentioning PaaS support, multi-tenancy and elasticity. To properly support multitenancy within the VM some form of modularity and isolation is required and OSGi modularity is available for JavaSE today.

Managing OSGi frameworks has been an important problem for traditional deployments on desktop machines, mobile devices or servers. The one side of the problem is monitoring a running OSGi framework instance, e.g., to ensure correctness of a deployment. In public cloud environments, this capability is essential since they are typically metered in some form and unwanted behavior can lead to monetary losses. The other side of framework management is the deployment. Being able to bootstrap the framework with a set of bundles has been shown to be important and lead to the inclusion of the launch API. In a PaaS setup, the management interface can be the only way of interacting with the framework instance and a prerequisite for deploying application modules.

Given the particular importance of management for cloud setups, it is clear that some form is needed and since the management interface is client-facing it needs to be standardized for interoperability between OSGi cloud offerings. Furthermore, existing protocols do not work well with wide-area networks and their firewall restrictions. Therefore, this RFC discusses the use of REST, a protocol akin to HTTP and widely used in interaction with clouds.

3 Problem Description

Representational State Transfer (REST) [4]. is the architectural style of the world wide web. It can be described as a set of constraints that govern the interactions between the main components of the Internet. Recently, REST style interaction has gained popularity as a architecture for web services (RESTful web services), mainly to overcome the perceived complexity and verbosity of SOAP-based web services [5]..

3.1 REST Design

Client-Server is a separation of concern between the entity responsible for the user-interaction (client) and the other entity (server) responsible for data storage. For instance, in the original world wide web the browser is the client rendering and presenting the content delivered by one or more web servers.
As a result, web content becomes more portable and content providers more scalable.

Stateless State is entirely kept at the client side. Therefore, every request must contain all state required for the server to accomplish the transaction and deliver content. The main rationale behind this design constraint is to again improve the scalability since in a pure stateless design the server resources are not burdened with maintaining any client state. Another perceived advantage is that the failure models of stateless interactions is simpler and fault tolerance easier to achieve.

Cacheable Content marked as cacheable can be temporarily stored and used to immediately answer future equivalent requests and improve efficiency and reduce network utilization and access latencies. Due to the end-to-end principle, caches can be placed where necessary, e.g., at the client (forward-proxy), or at the server side (backward-proxy).

Layered Layering introduces natural boundaries to coupling since every layer only accesses the services provided by the lower layer and provides services to the next higher layer.

Uniform Interface Generality of component interfaces provides a natural decoupling of implementation and interface. REST furthermore encourages the separation of identifiable resources (addressing) and their representation (content delivery).

3.2 REST Elements

Resources and Resource Identifiers: A resource is abstract piece of information that can be addressed by a resource identifier. The mapping of a resource to a concrete set of entities can vary over time.

Representation: A representation is a sequence of bytes plus associated meta-data that describe the state of a resource. The data format of a representation is called the *media-type*. Every concrete representation of a resource is just one of arbitrarily many possible representations. The selection of a concrete representation of a resource can be made according to the media types supported by both the client and the server.

3.3 The semantics of selected HTTP methods

GET retrieves the representation addressed by the request URI. The operation is expected to be idempotent.
POST add the enclosed entity as a new subordinate of the resource identified by the request URI.
PUT store the enclosed entity under the request URI. The operation is expected to be idempotent.
DELETE delete the resource identified by the request URI.

3.4 REST vs. OSGi

Scalability is unlikely to become a significant issue for an OSGi framework as a REST server.
The number of people that concurrently want to manage the same OSGi framework is not expected to be high.
What is an issue though is portability.
Users likely want to use a diversity of programming and scripting languages or even embed the management functionality into web pages.
Furthermore, the use of an open-standard protocol like HTTP that is able to easily penetrate firewalls as well as the frequent use of open data formats like XML or JSON as a base for the media types makes REST-style APIs appealing to users.
In this regard, a REST-style API can help to support the adoption of OSGi in the cloud.
Arguably, cloud users are used to some flavor of REST-style APIs so that not having one could in turn hinder acceptance of OSGi in the cloud.
One can, however, generally question the wisdom in treating an OSGi framework as a set of resources.
REST is centered around a data model that is hypertext-driven and many existing HTTP-based APIs that call themselves RESTful have been criticized for not complying with this principle [6]..
As a consequence, some operations that are straightforward in an object model like the traditional OSGi APIs (e.g., starting a bundle) require some re-thinking when mapping them to a resource-centric API. Starting a bundle now becomes an update to the state resource of a particular bundle resource.
One can particularly question if there is any way at all to make an OSGi framework present itself in a way that does not create coupling between client and server, one of the important goals of REST. After all, the proposed API is about managing OSGi frameworks and that in fact requires knowledge about OSGi and the operation it supports.
The pessimistic conclusion might be that it is neither wise nor fully possible to use pure REST for the management of an OSGi framework.
So, in other words: "*Whereof one cannot speak therof one must be silent*" [7].?
This proposal takes a pragmatic approach: REST-style APIs (may they be compliant or not) are successfully used for similar purposes. This proposal is intended to be a starting point. A discussion about the compliance with the REST design has been initiated.

4 Requirements

- MAN0001: The solution SHOULD define APIs to interact with a variety of different cloud platforms to manage the resource pool by, e.g., adding new nodes or removing nodes, where nodes represent OSGi Frameworks.
- MAN0002*:* The solution SHOULD define APIs that allow querying of deployed Systems (Composite Applications), System Elements (Subsystems), Bundles and Services on an OSGi Framework in the Cloud.
- MAN0003: The deployment of bundles SHOULD facilitate consistent behavior across multiple framework instances. It SHOULD be possible to receive feedback on the deployment status.
- MAN0004: – The solution MUST provide APIs to discover available OSGi Frameworks in a Resource Domain.

5 Technical Solution

The following API embraces three important management tasks:

- **introspecting** a running OSGi framework, the bundles installed, and the services available.
- **changing the state** of the framework by installing new bundles or changing the state of an existing bundle
- **dealing with the startlevel** of the framework and the target startlevels of bundles.

5.1 Resources

The framework and its state is mapped to a set of different resources. Each resource is accessible through a resource identifier, as summarized in 5.3.

5.1.1 The Framework Startlevel Resource

The startlevel resource represents the active start level of the framework. It supports the following requests:

Method	Description
[GET]	get the startlevel. Returns a bundle startlevel representation (5.2.5). *Status codes:* *200 (OK)* — *success* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*
[PUT]	set the startlevel. Expects the body of the message to be a bundle startlevel representation (5.2.5). *Status codes:* *204 (NO CONTENT)* — *success* *415 (UNSUPPORTED MEDIA TYPE)* — *the request has a media type that is not supported* *500 (INTERNAL SERVER ERROR)* — *the operation has thrown an exception, e.g., IllegalArgumentException when the requested start level is less or equal to zero.*

5.1.2 The Bundles Resource

The bundles resource represents the set of all bundles installed on the framework. The resource supports the following requests:

Method	Description
[GET]	request the list of installed bundles in the form of the bundle list representation (5.2.1).

	Status codes: *200 (OK)* — *success* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*
[POST]	install a bundle addressed by a URI. The body of the message is expected to be a plain URI string with mime type text/plain. The URL will be used as the location of the bundle. Returns the (local) URI of the newly installed bundle. *Status codes:* *200 (OK)* — *success* *400 (BAD REQUEST)* — *if the URL could not be parsed or the bundle could not be retrieved from the given URL* *500 (INTERNAL SERVER ERROR)* — *the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.*
[POST]	install the bundle that is uploaded as the body of this request. Media-type SHOULD be vnd.osgi.bundle. Implementations are free to accept additional mime types other than text/plain such as `application/zip` or `application/x-jar`. Implementations MUST generate a random location string that is passed to the installBundle (core 10.1.8.18) method to avoid unintended collisions between unrelated bundles. *Status codes:* *200 (OK)* — *success* *400 (BAD REQUEST)* — *if the request body is not a valid OSGi bundle* *500 (INTERNAL SERVER ERROR)* — *the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.*

The bundles resource returns a list of the URIs of all bundles installed on the framework. For clients interested in all bundles there is also the possibility to retrieve the bundle representation of each installed bundle with a single request through the bundles/representations resource:

Method	Description
[GET]	request the representations of all installed bundles in the form of a bundle representations list (5.2.1.2). *Status codes:* *200 (OK)* — *success* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*

5.1.3 The Bundle Resource

The bundle resource represents a single, distinct bundle in the system. Hence, it has to be qualified by a bundle id. The bundle resource supports the following requests:

<table>
<tr><th>Method</th><th>Description</th></tr>
<tr><td>[GET]</td><td>request the bundle representation (5.2).

Status codes:
<table>
<tr><td>200 (OK)</td><td>success</td></tr>
<tr><td>404 (NOT FOUND)</td><td>there is no bundle with this bundle id</td></tr>
<tr><td>406 (NOT ACCEPTABLE)</td><td>does not support any of the requested representations</td></tr>
</table></td></tr>
<tr><td>[PUT]</td><td>update the bundle with a new version. ~~Expects the body of the message to be a plain URI~~The body of the message is expected to be a plain URIString with mime type text/plain ~~-~~referencing the ~~new~~ bundle that is to be used for updating the existing bundle or an empty content to trigger bundle.update().

Status codes:
<table>
<tr><td>204 (NO CONTENT)</td><td>success</td></tr>
<tr><td>400 (BAD REQUEST)</td><td>if the URL~~I~~ could not be parsed or the bundle could not be retrieved from the given URL~~I~~</td></tr>
<tr><td>404 (NOT FOUND)</td><td>there is no bundle with this bundle id</td></tr>
<tr><td>500 (INTERNAL SERVER ERROR)</td><td>the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.</td></tr>
</table></td></tr>
<tr><td>[PUT]</td><td>Updates the content of the the bundle with the content in the body of this request. Media-type SHOULD be vnd.osgi.bundle. Implementations are free to accept additional mime types other than text/plain such as <code>application/zip</code> or <code>application/x-jar</code>.

Status codes:
<table>
<tr><td>200 (OK)</td><td>success</td></tr>
<tr><td>400 (BAD REQUEST)</td><td>if the request body is not a valid OSGi bundle</td></tr>
<tr><td>500 (INTERNAL SERVER ERROR)</td><td>the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.</td></tr>
</table></td></tr>
<tr><td>[DELETE]</td><td>uninstall the bundle.

Status codes:
<table>
<tr><td>204 (NO CONTENT)</td><td>bundle has been successfully uninstalled</td></tr>
<tr><td>404 (NOT FOUND)</td><td>there is no bundle with this bundle id</td></tr>
<tr><td>500 (INTERNAL SERVER ERROR)</td><td>the operation has thrown a bundle exception</td></tr>
</table></td></tr>
</table>

5.1.4 The Bundle State Resource

The bundle state resource represents the state of an installed bundle qualified through its bundle id. It supports the following requests:

Method	Description
[GET]	request the state of the bundle as a bundle status representation (5.2.2). *Status codes:* *200 (OK)* — *success* *404 (NOT FOUND)* — *there is no bundle with this bundle id* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*
[PUT]	attempts to start the bundle (setting the state to 32=started) or stop the bundle (setting the state to 4=resolved). Returns the actual new state in bundle state representation (5.2.2). *Status codes:* *200 (OK)* — *success* *404 (NOT FOUND)* — *there is no bundle with this bundle id* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations* *415 (UNSUPPORTED MEDIA TYPE)* — *the request has a media type that is not supported* *500 (INTERNAL SERVER ERROR)* — *when the operation has thrown a bundle exception. The body of the message MUST be a BundleException representation 5.2.8.*

5.1.5 The Bundle Header Resource

The bundle header resource represents the manifest header of a bundle.
The bundle header resource supports the following requests:

Method	Description
[GET]	get the bundle header representation (5.2.3). *Status codes:* *200 (OK)* — *success* *404 (NOT FOUND)* — *there is no bundle with this bundle id* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*

5.1.6 The Bundle Startlevel Resource

The startlevel resource represents the start level of a specific bundle. It supports the following requests:

Method	Description
[GET]	get the startlevel. Returns a startlevel representation (5.2.5). *Status codes:* *200 (OK)* — *success* *404 (NOT FOUND)* — *there is no bundle with this bundle id* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*
[PUT]	set the startlevel. Expects the body of the message to be a startlevel representation (5.2.5). *Status codes:* *204 (NO CONTENT)* — *success* *404 (NOT FOUND)* — *there is no bundle with this bundle id* *415 (UNSUPPORTED MEDIA TYPE)* — *the request has a media type that is not supported* *500 (INTERNAL SERVER ERROR)* — *the operation has thrown an exception, e.g., IllegalArgumentException when the requested start level is less or equal to zero or the bundle is the system bundle. If the caught exception was a BundleException, a BundleException representation 5.2.8. MUST be returned.*

Implementations MUST ignore the values for persistentlystarted and activationpolicyused in PUT request bodies.

5.1.7 The Services Resource

The services resource represents the set of all services available on the framework, optionally constrained by those matching a given filter expression. The resource supports the following requests:

Method	Description
[GET]	request the list of available services in the form of one of the service list representation (5.2.7.1). *Status codes:* *200 (OK)* — *success* *400 (BAD REQUEST)* — *the filter expression was not valid (only if a filter is used)* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*

The services resource returns a list of the URIs of all services available on the framework. For clients interested in all services there is also the possibility to retrieve the service representation of each available service with a single request through the services/representations resource:

Method	Description
[GET]	request the representations of all available services in the form of a service representations list (5.2.7.2). *Status codes:* *200 (OK)* — *success* *400 (BAD REQUEST)* — *the filter expression was not valid (only if a filter is used)* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*

5.1.8 The Service Resource

The service resource represents a single, distinct service in the system. Hence, it has to be qualified by a service id. The service resource supports the following requests:

Method	Description
[GET]	request the service representation (5.2.6). *Status codes:* *200 (OK)* — *success* *404 (NOT FOUND)* — *there is no service with this id* *406 (NOT ACCEPTABLE)* — *does not support any of the requested representations*

5.2 Representations

The Bundle Representation

JSON	Content-Type: **`application/org.osgi.bundle+json`**
	
	`{`
	`"id":0,`
	`"lastModified":1314999275542,`
	`"location":"System Bundle",`
	`"state":32,`
	`"symbolicName":"org.eclipse.osgi",`
	`"version":"3.7.0.v20110613"`
	`}`

XML	Content-Type: **`application/org.osgi.bundle+xml`**

	<bundle> <id>0</id> <lastModified>1314999275542</lastModified> <location>System Bundle</location> <state>32</state> <symbolicName>org.eclipse.osgi</symbolicName> <version>3.7.0.v20110613</version> </bundle>

5.2.1 Bundles Representations

5.2.1.1 The Bundle List Representation

JSON	Content-Type: **application/org.osgi.bundles+json**
	
	{
	[*bundleURI*, *bundleURI*, ..., *bundleURI*]
	}

XML	Content-Type: **application/org.osgi.bundles+xml** <bundles> <uri>*bundleURI*</uri> <uri>*bundleURI*</uri> ... <uri>*bundleURI*</uri> </bundles>

5.2.1.2 The Bundle Representations List Representation

JSON	Content-Type: **application/org.osgi.bundles.representations+json.** [*BUNDLE REPRESENTATION, BUNDLE REPRESENTATION, …, BUNDLE REPRESENTATION*]
XML	Content-Type: **application/org.osgi.bundles.representations+xml** <bundles> *BUNDLE REPRESENTATION* *BUNDLE REPRESENTATION* ... *BUNDLE REPRESENTATION* </bundles>

5.2.2 The Bundle State Representation

JSON	Content-Type: **application/org.osgi.bundlestate+json**

	{
	"state":32
	"options":1
	}

XML	Content-Type: **application/org.osgi.bundlestate+xml** <bundleState> <state>32</state> <options>1</options> </bundleState>

5.2.3 The Bundle Header Representation

JSON	Content-Type: **application/org.osgi.bundleheader+json**
	
	{
	key:*value*,
	key:*value*,
	...
	key:*value*
	}

XML	Content-Type: **application/org.osgi.bundleheader+xml** <bundleHeader> <entry key="*key*" value="*value*"/> <entry key="*key*" value="*value*"/> ... <entry key="*key*" value="*value*"/> <bundleHeader>

5.2.4 The Framework Startlevel Representation

JSON	Content-Type: **application/org.osgi.frameworkstartlevel+json**
	
	{
	"startLevel":6,
	"initialBundleStartLevel":4
	}

XML	Content-Type: **application/org.osgi.frameworkstartlevel+xml** <frameworkStartLevel> <startLevel>6</startLevel> <initialBundleStartLevel>4</initialBundleStartLevel> </frameworkStartLevel>

5.2.5 The Bundle Startlevel Representation

JSON	Content-Type: **application/org.osgi.bundlestartlevel+json**
	
	{
	"startLevel":6
	
	"activationPolicyUsed":true
	"persistentlyStarted":false
	}

XML	Content-Type: **application/org.osgi.bundlestartlevel+xml** <bundleStartLevel> <startLevel>6</startLevel> <activationPolicyUsed>true</actiovationPolicyUsed> <persistentlyStarted>false</persistentlyStarted> </bundleStartLevel>

5.2.6 The Service Representation

JSON	Content-Type: **application/org.osgi.service+json**
	
	{
	properties:
	{
	key:value,
	key:value,
	...
	key:value
	},
	"bundle":bundleURI,
	"usingBundles":[bundleURI, bundleURI, … bundleURI],
	}

XML	Content-Type: **application/org.osgi.service+xml** <service> <properties> <entry key="key" value="value"/> <entry key="key" value="value"/> ... <entry key="key" value="value"/> </properties> <bundle>bundleURI</bundle> <usingBundles> <bundle>bundleURI</bundle> <bundle>bundleURI</bundle> ... <bundle>bundleURI</bundle>

```
    </usingBundles>
</service>
```

5.2.7 Service Representation

5.2.7.1 The Service List Representation

JSON	Content-Type: **application/org.osgi.services+json**
	
	{
	[service*URI*, service*URI*, ..., service*URI*]
	}

XML	Content-Type: **application/org.osgi.services+xml** <services> <uri>service*URI*</uri> <uri>service*URI*</uri> ... <uri>service*URI*</uri> </services>

5.2.7.2 The Service Representations List Representation

JSON	Content-Type: **application/org.osgi.services.representations+json.** [SERVICE *REPRESENTATION*, SERVICE *REPRESENTATION*, …, SERVICE *REPRESENTATION*]
XML	Content-Type: **application/org.osgi.services.representations+xml** <services> SERVICE *REPRESENTATION* SERVICE *REPRESENTATION* ... SERVICE *REPRESENTATION* </services>

5.2.8 BundleException Representation

If the implementation catches a BundleException while retrieving the requested resource, a BundleException representation is returned. The message can be an arbitraty string and is intended to be interpreted by humans. The type code corresponds to the codes defined in 10.1.10 of the core specification and MUST be set by the implementation to the value of the getType() method (10.1.20) on the caught BundleException.

JSON	Content-Type: **application/org.osgi.bundleexception+json** { “typecode”: 5,

	"message": "BundleException: Bundle activation error" }
XML	Content-Type: **application/org.osgi.bundleexception+xml** <bundleexception> <typecode>*5*</typecode> <message>*BundleException: Bundle activation error*</message> </bundleexception>

5.3 Resource Identifier Overview

```
framework
framework/state
framework/startlevel
framework/bundles

framework/bundles/representations

framework/bundle/{bundleid}
framework/bundle/{bundleid}/state
framework/bundle/{bundleid}/startlevel
framework/bundle/{bundleid}/header
framework/services

framework/services/representations

framework/service/{serviceid}
```

framework/bundle/0/state is an alias for framework/state

The `bundles`, `bundles/representations`, `services`, and `services/representations` resources allow the use of a query parameter which specifies a filter to restrict the result set.

The filter expression follows the IETF RFC 1960 String Representation of LDAP Search Filter format [8]..

Filters on services are matched against the service attributes. The query parameter is of the form:

framework/services?filter=*ldap-filter*

Filters on bundles are matched against the attributes of capabilities in the respective namespaces. Filters on bundles have the form:

framework/bundles?*namespace1=ldap-filter1***&***namespace2=ldap-filter2***&...**

A missing namespace declaration implies the IdentityNamespace ("osgi.identity").

5.4 Content Type Matching

The solution offering different variants of representations (e.g., JSON or XML) MUST return the best matching variant based on the HTTP accept header. In addition, it MUST respect the file extensions defined for the different media types as specified in the respective IETF RFC (e.g., "*.xml*" as specified in IETF RFC 3032 and "*.json*" as

specified in IETF RFC 4627). iI a file extension is appended to the resource, the solution MUST return the variant mandated by the file extension if it supports this content type..

5.5 Versioning and Interoperability

All representations described in this document have version 1. Future versions of the representations MUST contain the version number in their content type (e.g., **`application/org.osgi.bundlesV2)`** to allow legacy clients to explicitly request an older version of the representation that they understand by setting their accept header.

Clients MUST understand all attributes described in this documents and MAY ignore any further attribute that the implementation of the REST interface might add.

5.6 API Extensions

Implementations MUST allow providers of REST interfaces for components other than the framework to extend the client-facing API. Since there are various possible technologies that can be used for implementing the REST API, e.g., Java Servlet, Restlet, JAX-RS, etc., the extension mechanism can only provide visibility of extensions to the client since any tighter integration of extensions would expose implementation details and therefore limit the choice of technologies that can be used for an implementation.

In order to be discoverable by the client, an extension MUST register itself as a org.osgi.rest.RestApiExtension service through a whiteboard pattern. It MUST set the org.osgi.rest.RestApiExtension.URI_PATH property to the URI under which it can be reached. If the URI is relative, it is interpreted as relative to the “extensions/” path. Extensions providing a REST API for OSGi specifications MUST set org.osgi.rest.RestApiExtension.NAME to the package name of the specification for which they provide the API. All other extensions MUST set the field to a package name but MAY choose a name other than names with “org.osgi*” as a prefix.

Whiteboard pattern registrations under the path “framework” or “extensions” MUST be ignored. Implementations MUST provide the Extensions Resource under the path “extensions”, at the same hierarchical level as the framework resource.

5.6.1 The Extensions Resource

The extensions resource enumerates all API extensions that are currently registered through the whiteboard pattern:

Method	Description
[GET]	request the list of registered extensions in the form of one of the extension list representation (5.6.2). *Status codes:* *200 (OK)* — *success*

5.6.2 The Extension List Representation

JSON	Content-Type: **`application/org.osgi.extensions+json`**

	{
	[{ "name" : "org.osgi.service.event", "path" : "extensions/eventadmin" }, …]
	}

XML	Content-Type: **application/org.osgi.extensions+xml** <extensions> <extension> <name>org.osgi.service.event</name> <path>extensions/eventadmin</path> </extension> </extensions>

6 Considered Alternatives

The only technology that currently exist under the OSGi umbrella and that is designed for performing the remote management of a running OSGi framework is JMX. Arguably, JMX could be used in the cloud since it generally does not prohibit the use of web service protocols but it also does not make particularly easy to use them. The main argument against JMX has to be that REST-style APIs are easier to deal with from a programmer's perspective and are much more adopted in the overall cloud ecosystem.

Another alternative to avoid creating new APIs would be to canonically map the existing JMX APIs to REST, e.g., by implementing a JMX-to-REST bridge. This, however, would suffer from a great conceptual mismatch since JMX is an object-centric architecture whereas REST is resource-centric. The resulting REST-style API would possibly violate each and every REST design principle.

Finally, RFC 140 (Residential Management Tree) from the Residential Expert Group (REG) deals with a similar problem but is targeted to the object-centric TR-069 protocol. There is some overlap of the conceptual model with the proposed REST API but RFC 140 requires a fine-granular micro-management approach while this RFC is focused on the management of OSGi frameworks in the large.

Some particular design decisions were discussed and alternative approaches explored.

First of all, there is duplication in the API with regard to the bundle state. This property of the bundle can either be retrieved through the bundle representation or through the bundle state resource. The reason for this design is that the bundle state is the only mutable property of the bundle. Therefore, it has been modeled as a separate resource that accepts updates through post.
As an alternative, one could make the entire bundle representation mutable and accepting post requests and every update other than on the state property would either be silently ignored or result in an error code to be returned.

Second, there has been discussions about modeling the lifecycle operations on bundles not through a resource at all but through new HTTP methods on the bundle resource. The IETF RFC 2068 explicitly allows other methods (extension methods) for HTTP requests. However, concerns were raised with regard to problems with firewalls or

other security mechanisms in the communication path that could reject any HTTP method that is not explicitly defined in the IETF RFC 2068.

In this context, it was even discussed to abandon the use of PUT and DELETE since some members reported customer to have firewall restrictions in place that only allow GET and POST requests. However, after doing a survey of existing cloud APIs, the result was that all of the APIs of major cloud players (we looked at Amazon {S3, CloudFront, Route 53, and Simple DB}, Rackspace, Microsoft Azure {BlobService, QueueService, and Table Service}; Google App Engine appengine-rest-server; VMWare vCloud) excepti for the Amazon SimpleDB API make use of PUT and DELETE, some also use HEAD in their API.

7 Javadoc

Interface RestApiExtension

org.osgi.service.rest

```
public interface RestApiExtension
```

Field Summary		Page
String	NAME	Error: Reference source not found
String	URI_PATH	Error: Reference source not found

Field Detail

URI_PATH

```
public static final String URI_PATH = "org.osgi.rest.uri.path"
```

NAME

```
public static final String NAME = "org.osgi.rest.name"
```

8 REST Client Javadoc

Interface RestClient

org.osgi.service.rest.client

All Known Implementing Classes:

RestClientImpl

```
public interface RestClient
```

Method Detail

getFrameworkStartLevel

```
org.osgi.dto.framework.startlevel.FrameworkStartLevelDTO getFrameworkStartLevel()
                                                                    throws Exception
```

Throws:

Exception

---

setFrameworkStartLevel

```
void setFrameworkStartLevel(org.osgi.dto.framework.startlevel.FrameworkStartLevelDTO startLev
el)
                     throws Exception
```

Throws:

Exception

---

getBundles

```
Collection<String> getBundles()
                        throws Exception
```

Throws:

Exception

---

getBundleRepresentations

```
Collection<org.osgi.dto.framework.BundleDTO> getBundleRepresentations()
                                                          throws Exception
```

Throws:

Exception

getBundle

```
org.osgi.dto.framework.BundleDTO getBundle(long id)
                                    throws Exception
```

Throws:
Exception

---

getBundle

```
org.osgi.dto.framework.BundleDTO getBundle(String bundlePath)
                                    throws Exception
```

Throws:
Exception

---

getBundleState

```
int getBundleState(long id)
            throws Exception
```

Throws:
Exception

---

getBundleState

```
int getBundleState(String bundlePath)
            throws Exception
```

Throws:
Exception

---

startBundle

```
void startBundle(long id)
          throws Exception
```

Throws:
Exception

---

startBundle

```
void startBundle(String bundlePath)
          throws Exception
```

Throws:

Exception

---

startBundle

```
void startBundle(long id,
                 int options)
          throws Exception
```

Throws:

Exception

---

startBundle

```
void startBundle(String bundlePath,
                 int options)
          throws Exception
```

Throws:

Exception

---

stopBundle

```
void stopBundle(long id)
         throws Exception
```

Throws:

Exception

---

stopBundle

```
void stopBundle(String bundlePath)
         throws Exception
```

Throws:

Exception

---

stopBundle

```
void stopBundle(long id,
                int options)
         throws Exception
```

Throws:
Exception

---

stopBundle

```
void stopBundle(String bundlePath,
                int options)
         throws Exception
```

Throws:
Exception

---

getBundleHeaders

```
org.osgi.dto.MapDTO<String,Object> getBundleHeaders(long id)
                                                throws Exception
```

Throws:
Exception

---

getBundleHeaders

```
org.osgi.dto.MapDTO<String,Object> getBundleHeaders(String bundlePath)
                                                throws Exception
```

Throws:
Exception

---

getBundleStartLevel

```
org.osgi.dto.framework.startlevel.BundleStartLevelDTO getBundleStartLevel(long id)
                                                                    throws Exception
```

Throws:
Exception

---

getBundleStartLevel

```
org.osgi.dto.framework.startlevel.BundleStartLevelDTO getBundleStartLevel(String bundlePath)
                                                                    throws Exception
```

Throws:
Exception

---

setBundleStartLevel

```
void setBundleStartLevel(long id,
                         org.osgi.dto.framework.startlevel.BundleStartLevelDTO startLevel)
                  throws Exception
```

Throws:
Exception

---

setBundleStartLevel

```
void setBundleStartLevel(String bundlePath,
                         org.osgi.dto.framework.startlevel.BundleStartLevelDTO startLevel)
                  throws Exception
```

Throws:
Exception

---

installBundle

```
String installBundle(URL url)
              throws Exception
```

Throws:
Exception

---

installBundle

```
String installBundle(InputStream in)
              throws Exception
```

Throws:
Exception

---

uninstallBundle

```
void uninstallBundle(long id)
              throws Exception
```

Throws:

Exception

---

uninstallBundle

```
void uninstallBundle(String bundlePath)
              throws Exception
```

Throws:

Exception

---

updateBundle

```
void updateBundle(long id)
           throws Exception
```

Throws:

Exception

---

updateBundle

```
void updateBundle(long id,
                  URL url)
           throws Exception
```

Throws:

Exception

---

updateBundle

```
void updateBundle(long id,
                  InputStream in)
           throws Exception
```

Throws:

Exception

---

getServices

```
Collection<String> getServices()
                        throws Exception
```

Throws:
Exception

---

getServices

```
Collection<String> getServices(String filter)
                        throws Exception
```

Throws:
Exception

---

getServiceRepresentations

```
Collection<org.osgi.dto.framework.ServiceReferenceDTO> getServiceRepresentations()
                                                                    throws Exception
```

Throws:
Exception

---

getServiceRepresentations

```
Collection<org.osgi.dto.framework.ServiceReferenceDTO> getServiceRepresentations(String filte
r)
                                                                    throws Exception
```

Throws:
Exception

---

getServiceReference

```
org.osgi.dto.framework.ServiceReferenceDTO getServiceReference(long id)
                                                        throws Exception
```

Throws:
Exception

---

getServiceReference

```
org.osgi.dto.framework.ServiceReferenceDTO getServiceReference(String servicePath)
                                                        throws Exception
```

Throws:

`Exception`

Interface RestClientFactory

org.osgi.service.rest.client

```
public interface RestClientFactory
```

Method Detail

createRestClient

```
RestClient createRestClient(URL url)
```

9 REST Client Javascript

9.1.1.1 OsgiRestClient(baseUrl)

The OSGi REST API client

Parameters:

Name	Type	Description
baseUrl		the base URL to the REST API.

Returns:
the OSGi REST API client object

9.1.2 Methods

9.1.2.1 getBundle(b)

get the Bundle representation of a specific bundle

Parameters:

Name	Type	Description
b		the bundle, either the numeric bundle ID or the bundle URI path.

Returns:
the Bundle representation as a JSON object.

9.1.2.2 getBundlesRepresentations()

get the bundle representations of all bundles
Returns:
a JSON array containing all Bundle representations

9.1.2.3 getBundles()

get the bundles
Returns:
the URI paths of the bundles as a JSON array of strings.

9.1.2.4 getBundleStartLevel(b)

Get the bundle startlevel.

Parameters:

Name	Type	Description
b		the bundle, either the numeric bundle ID or the bundle URI path.

9.1.2.5 undleState(b)

Get the state of a bundle.

Parameters:

Name	Type	Description
b		the bundle, either the numeric bundle ID or the bundle URI path.

Returns:
the bundle state representation.

9.1.2.6 getFrameworkStartLevel()

get the framework start level in JSON FrameworkStartLevel representation.

9.1.2.7 getServiceRepresentations()

Get the representations of all services

Returns:
a JSON array containing all representations.

9.1.2.8 getServices()

Get all services.

Returns:
a JSON array of the URI paths of all services.

9.1.2.9 getServices(s)

Get the representation of a service.

Parameters:

Name	Type	Description
`s`		the service, either the numeric service ID or the service URI path.

Returns:
the service representation.

9.1.2.10installBundle(uri)

Install a new bundle.

Parameters:

Name	Type	Description
`uri`		the URI of the bundle to be installed

Returns:
the URI path of the newly installed bundle

9.1.2.11setBundleStartLevel(b, the)

Set the startlevel of a bundle.

Parameters:

Name	Type	Description
`b`		the bundle, either the numeric bundle ID or the bundle URI path.
`the`		target startlevel representation.

Returns:
the updated startlevel representation.

9.1.2.12setBundleState(b, state)

Set the state of a bundle to start or stop it.

Parameters:

Name	Type	Description
`b`		the bundle, either the numeric bundle ID or the bundle URI path.
`state`		the target state.

Returns:
the updated state of the bundle.

9.1.2.13setFrameworkStartLevel(the)

set the framework start level

Parameters:

Name	Type	Description
`the`		new framework startlevel in JSON representation.

Returns:
the updated framework startlevel in JSON representation.

9.1.2.14startBundle(b)

Start a bundle.

Parameters:

Name	Type	Description
`b`		the bundle, either the numeric bundle ID or the bundle URI path.

Returns:
the updated bundle state representation.

9.1.2.15stopBundle(b)

Stop a bundle.

Parameters:

Name	Type	Description
`b`		the bundle, either the numeric bundle ID or the bundle URI path.

Returns:
the updated bundle state representation.

9.1.2.16uninstallBundle(b)

Uninstall a bundle.

Parameters:

Name	Type	Description
`b`		the bundle, either the numeric bundle ID or the bundle URI path.

9.1.2.17updateBundle(b, uri)

Update a bundle.

Parameters:

Name	Type	Description
`b`		the bundle, either the numeric bundle ID or the bundle URI path.
`uri`		the URI from which to update the bundle.

10 Security Considerations

Like any externally visible management interface, the REST interface exposes privileged operations and hence requires access control. Since REST builds upon the HTTP(s) protocol, authentication mechanisms and encryption can be applied the same way as usually done for web servers: they can be layered below the REST protocol. E.g., confidentiality of the transmitted commands can be ensured by using HTTPS as the underlying transport. Authentication can be added by requiring, e.g., basic authentication prior to accepting a REST command.

The REST interface should only be implemented by a trusted bundle.

mplementations of this specification require all admin permissions and all service permissions.

11 Document Support

11.1 References

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

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. RFP 133: Cloud Computing

[4]. Fielding, R. T., Architectural Styles and the Design of Network-based Software Architectures, PhD Thesis, UC Irvine, 2000

[5]. W3C Consortium, SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2004

[6]. Fielding, R. T., REST APIs must be hypertext-driven, http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[7]. Wittgenstein, L., Tractatus Logico-Philosophicus, ISBN 0-415-05186-X

[8]. Howes, T., A String Representation of LDAP Search Filters, RFC 1960, June 1996

11.2 Author's Address

Name	Jan S. Rellermeyer
Company	IBM Research
Address	11501 Burnet Rd. Austin, TX 78758
e-mail	rellermeyer_at_us.ibm.com

11.3 Acronyms and Abbreviations

REST – Representational State Transfer

HTTP – Hypertext Transfer Protocol

11.4 End of Document