DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

DATE OF DISTRIBUTION: 19 October 2009

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable, worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future (“Future Specification”), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license.
under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.
Preface

This document is Early Draft 2 of the OSGi Service Platform Release 4 Residential Version 4.2 specifications. As an early draft, it contains non-final specification work and it is not organized in the format normally associated with final release OSGi specifications. This document contains copies of OSGi design documents which either propose to modify existing published OSGi specifications from the OSGi Service Platform Release 4 Version 4.2 specification documents or propose new specifications to potentially be incorporated in the final OSGi Service Platform Release 4 Residential Specification Version 4.2 documents.

Since this early draft is not a complete specification document, the reader is expected to be familiar with OSGi Technology and the currently published OSGi Service Platform Release 4 Version 4.2 specification documents. The reader should refer to http://www.osgi.org/About/Technology for more information on the OSGi Technology. There the reader can find a description of the OSGi Technology, as well as links to whitepapers and the OSGi Service Platform Release 4 Version 4.2 specification documents, which are all available for download.

In an effort to make this early draft available as quickly as possible, it contains OSGi design documents ("RFCs"). These documents have been declassified by the OSGi Alliance so that they may be made available in this early draft. This early draft contains a majority of the design documents the OSGi expert groups currently anticipate will be incorporated into the final specification documents.

Pursuant to the Distribution and Feedback License above, the OSGi expert groups welcome your feedback on this early draft. Feedback can be provided by opening a bug at https://www.osgi.org/bugzilla/enter_bug.cgi?product=OSGi%20Specification.

BJ Hargrave
Chief Technical Officer
OSGi Alliance
Residential Design Documents

OSGi Service Platform Release 4
Version 4.2 – Residential Early Draft 2

Revision 1.1
19 October 2009
RFC-140 Residential Management Tree

Draft

38 Pages

Abstract

Different industries are interested in the application of OSGi for their business, in which remote management is a key issue. Telecom operators (both fixed and mobile ones), server managers, and automotive manufactures, etc. need solutions to remotely manage their instances of OSGi frameworks. The main problem with that need is that is very difficult to solve with a single solution, taking into account that the management protocols are many and different for the different industries. To achieve this goal, a management object model should be defined to expose OSGi framework manageable information through a management agent.
# 1 Document Information

## 1.1 Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Document Information</td>
<td>2</td>
</tr>
<tr>
<td>1.1 Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Terminology and Document Conventions</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Revision History</td>
<td>3</td>
</tr>
<tr>
<td>2 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>3 Application Domain</td>
<td>5</td>
</tr>
<tr>
<td>4 Problem Description</td>
<td>6</td>
</tr>
<tr>
<td>4.1 Management agent making use of OSGi standardized Java interfaces</td>
<td>6</td>
</tr>
<tr>
<td>4.2 Mobile specification approach</td>
<td>7</td>
</tr>
<tr>
<td>4.2.1 Support for TR-069 notifications</td>
<td>7</td>
</tr>
<tr>
<td>4.2.2 Limitations in the number of services available in the DMT</td>
<td>8</td>
</tr>
<tr>
<td>4.2.3 Mapping TR-069 to the OMA-DM inspired DMT model</td>
<td>9</td>
</tr>
<tr>
<td>4.2.4 Conclusion</td>
<td>9</td>
</tr>
<tr>
<td>5 Requirements</td>
<td>9</td>
</tr>
<tr>
<td>6 Technical Solution</td>
<td>10</td>
</tr>
<tr>
<td>6.1 Legend</td>
<td>12</td>
</tr>
<tr>
<td>6.2 Framework Object</td>
<td>12</td>
</tr>
<tr>
<td>6.3 BundleState Object</td>
<td>19</td>
</tr>
<tr>
<td>6.4 PackageState Object</td>
<td>22</td>
</tr>
<tr>
<td>6.5 ServiceState Object</td>
<td>24</td>
</tr>
<tr>
<td>6.6 Filters Object</td>
<td>27</td>
</tr>
<tr>
<td>6.7 BundleResources Object</td>
<td>33</td>
</tr>
<tr>
<td>6.8 Configuration Object</td>
<td>34</td>
</tr>
<tr>
<td>6.9 Policy Object</td>
<td>34</td>
</tr>
<tr>
<td>6.10 Log Object</td>
<td>34</td>
</tr>
<tr>
<td>6.11 Limitations</td>
<td>35</td>
</tr>
<tr>
<td>7 Considered Alternatives</td>
<td>36</td>
</tr>
<tr>
<td>8 Security Considerations</td>
<td>37</td>
</tr>
<tr>
<td>9 Document Support</td>
<td>37</td>
</tr>
<tr>
<td>9.1 References</td>
<td>37</td>
</tr>
<tr>
<td>9.2 Author's Address</td>
<td>38</td>
</tr>
<tr>
<td>9.3 Acronyms and Abbreviations</td>
<td>38</td>
</tr>
</tbody>
</table>
1.2 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 9.

Source code is shown in this typeface.

1.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Sep. 12 2008</td>
<td>Initial Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Koya Mori, NTT Corporation, <a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.2</td>
<td>Oct. 9 2008</td>
<td>2nd Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Overall architecture is changed based on REG meeting and RFC-139 definition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Koya Mori, NTT Corporation, <a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ikuo Yamasaki, NTT Corporation, <a href="mailto:yamasaki.ikuo@lab.ntt.co.jp">yamasaki.ikuo@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.3</td>
<td>Nov. 10 2008</td>
<td>3rd Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Definition tables for all nodes in Management Objects are added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Koya Mori, NTT Corporation, <a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ikuo Yamasaki, NTT Corporation, <a href="mailto:yamasaki.ikuo@lab.ntt.co.jp">yamasaki.ikuo@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.4</td>
<td>Jan. 19 2009</td>
<td>4th Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Revising tree architecture based on the discussion at the REG meeting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Filter format description is added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Koya Mori, NTT Corporation, <a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ikuo Yamasaki, NTT Corporation, <a href="mailto:yamasaki.ikuo@lab.ntt.co.jp">yamasaki.ikuo@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>0.5</td>
<td>Feb. 12 2009</td>
<td>5th Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Revising Bundles Object based on discussion at the REG meeting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Koya Mori, NTT Corporation, <a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ikuo Yamasaki, NTT Corporation, <a href="mailto:yamasaki.ikuo@lab.ntt.co.jp">yamasaki.ikuo@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| 0.6      | Mar. 26 2009 | 6th Draft  
Revising Filters Object specification based on discussion and analysis.  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp |
| 0.7      | May. 14 2009 | 7th Draft  
Revising Resources sub-tree and Filters Object specification based on discussion at the REG meeting.  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp  
Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp |
| 0.8      | Jun. 04 2009 | 8th Draft  
Revising wording and ServiceState Object specification based on discussion at the REG meeting.  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp  
Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp |
| 1.0      | Sep. 18 2009 | 9th Draft  
Bundle StartLevel configuration is moved to the Framework Object.  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp  
Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp |
| 1.1      | Oct. 16 2009 | 10th Draft  
Lifecycle sub-tree of Bundle is moved to the Framework Object.  
Revising BundleState Object specification and BundleResources Object specification based on discussion at the REG meeting.  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp  
Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp |
2 Introduction

Traditionally, fixed telecommunication operators don’t have knowledge about what runs in the customer’s local area network (LAN). They provide connectivity and manage the wide area network (WAN) that provides this connectivity, but they do not know anything about the devices and networks behind the gateway (xDSL mainly) that interconnect WAN and LAN. Recently the need for management of customer networks and devices is increasing in order to make the deployment of new complex services at home (home automation, tele-health, VoIP, IPTV, surveillance, etc) feasible with reasonable costs. For example to avoid sending technicians to the customer premises for solving problems.

There are two main kinds of devices that need to be managed remotely in operator’s business: those which come from the fixed line business managed via the TR-069 protocol, that is standardized by the Broadband FORUM [6], and those which come from the mobile business managed by OMA DM, which is standardized by the Open Mobile Alliance. Up to now, the OSGi specifications cover the OMA DM ones, but lacks of specification about how an OSGi framework on a device could be managed by TR-069.

One key question is: why do we need two technologies to manage devices in a converged scenario in which the two kinds of devices are going to interact between them? The question has a better answer from the business point of view than from a technical point of view. Fixed and mobile businesses have evolved independently during the last years. Both technologies have acquired enough critical mass not only for having success in product implementation and roll-out, but also to be a de facto standard in their respective applications domains.

Despite the fact that the best solution would be a single protocol to manage all kind of devices, we are aware that at least for the short-medium term both technologies must coexist. For the future, the model shouldn’t be closed to add new solutions, for example a mixed model that unifies both worlds.

3 Application Domain

Driven by triple play service delivery in the home network, fixed line access service providers have the need to configure home devices to ensure the proper service delivery. Broadband Forum’s CPE WAN Management Protocol (CWMP, alias TR-069) enables them to do this. By using a remote management server (Auto Configuration Server, ACS), they are able to manage TR-069 enabled devices. TR-069 provides them with possibilities to configure parameters, be informed of parameter changes (notification), start diagnostic tools, update firmware, etc.

Similarly, for the mobile world, the OMA defined the OMA-Device Management specification for remote management of mobile devices. OMA-DM offers similar tools to the mobile service providers as TR-069 to fixed line service provider, but OMA-DM is of course tailored to the specifics of the mobile environment.

As OSGi technology offers a flexible software environment for all these different devices, the remote management of the platform is of interest for both fixed and mobile service providers. As such, it should be possible to integrate
the remote management of the OSGi platform, and the applications running on top of it, in the existing management infrastructure.

The DMT Admin service with its mobile management tree in the Mobile specification for OSGi R4 standardizes the remote management of an OSGi platform. As it is largely inspired by OMA-DM, it needs to be evaluated for multi protocol support.

4 Problem Description

In a scenario in which service providers offer a growing number of services, to use specific solutions for the management of those services is not the most suitable option. To speed up the deployment of these services, such as triple play, home automation or tele-health, it is essential to offer general management solutions that allow for the management of a large number of services and the flexible life-cycle management of applications.

These devices usually are already managed by a standard protocol, so it makes sense that an OSGi framework, which hosts the services, running on a device could be managed in the same way as the other resources of the device. Of course, the remote management should be fully integrated in the existing remote management solutions of the service provider to avoid duplicating management infrastructure and to increase performance on the devices.

Currently, there are two options in OSGi for remote management:

- create a management agent bundle making use of the Java object interfaces,
- create a protocol adapter bundle that interacts with the DMT Admin service, as defined in the OSGi Mobile specification.

4.1 Management agent making use of OSGi standardized Java interfaces

Currently, for the management of a bundle, the OSGi specifications define different Java objects with which a management application can interact. Using this approach, a management agent can implement extensive management of the OSGi framework, as well as any service standardized. Mapping the Java interfaces to the specific remote management protocol and data model tree is up to the management agent.

For runtime interaction with a bundle, a bundle can register a service in the service registry. However, this service interface is not standardized. Also, mapping the service interface to a general management model is not standardized. A current approach is to implement a proprietary service interface on all bundles to be managed. By tailoring this interface so that it easily maps to the management protocol primitives, it is simple for the management agent to map remote management commands to the bundle’s service interface. The disadvantage is the proprietary service interface, so that 3rd party bundles might not be compliant.

As a conclusion we can say that this current approach allows for extensive remote management of any aspect of the OSGi platform, but lacks a standardized service interface definition for bundles to implement.
4.2 Mobile specification approach

The Mobile Expert Group has provided its own solution based on the OMA [3] Device Management [4] specification to provide a remote management solution. The OSGi Mobile specification contains two chapters related to remote management:

- chapter 3: detailing the mobile management tree
- chapter 117: detailing the DMT Admin service, bundle plugin interface specification, the notification service

The Device Management Tree model of OMA-DM was chosen as meta-data model and operational model. However, it was intended to be mappable to other protocols.

An analysis of mapping the Mobile specification DMT model to TR-069, however, shows that the current DMT model approach (as defined in the OSGi R4 Mobile Specification) introduces some issues. For example:

- Limitations for active or passive notifications on any parameter in the object tree
- A limited number of services have been mapped to the DMT model
- The complexity of mapping a new protocol to the OMA-inspired DMT model, which could imply performance issues on limited devices.

4.2.1 Support for TR-069 notifications

TR-069 offers the feature of active and passive notifications. By setting a parameter’s notification attribute, a remote manager requests to be notified with the parameter’s new value at the time the value changes (active notification) or at the next periodic inform (passive notification). Notification can be configured on any parameter of the TR-069 object tree. This approach enables the remote manager to be informed not only of changes in status variables of the platform, but also of configuration changes performed by a local manager, e.g. through a local Web interface.

The Mobile specification offers a few features that could help to implement TR-069 notification support:

- The DMT Admin service sends events using the Event Admin service when operations have been performed on nodes (nodes added, removed or copied; node values changed etc.)
- The OSGi Notification service defines a way to alert a remote management server. Protocol adapters on their turn have to implement a RemoteAlertSender interface (and register it) for use by the notification service. Notifications are sent by calling sendNotification on the notification service:
  - The Monitor Admin service: A bundle can register a Monitorable service, to be used by the Monitor Admin service. By registering a Monitorable service, the bundle exposes access to a number of status variables. Notification can be implemented by the StatusVariable provider. If it does, it will call the update method on the Monitor Listener. The Monitor Admin service then generates an event on the Event Admin service. The Monitor service is currently also represented in the DMT tree.

Two problems arise when trying to map the current approach to TR-069:

- TR-069 defines that notification is applicable to any parameter in the object tree.
  
  Currently, the DMT Admin service only send events for operations on DMT nodes that were performed using the DMT Admin API. For example: if configuration changes are performed by
using the Configuration Admin service API, no events will be sent. Most of the current implementations do not perform all changes via the DMT Admin service. Therefore, the events sent by the DMT Admin service are an only subset and thus not very reliable as single source of events (and thus as single source of TR-069 notifications).

The OSGi Monitor service only supports notification of changes on Status Variables, exposed through a Monitorable service, and enabled by the bundle to support on-change notification (i.e. dynamic Status Variables).

- Requesting notification is not fully under the control of the remote manager. In the case of a bundle using the notification service, there is no standardized way to configure the bundle to send alerts when the value of one of the implemented DMT nodes changes. In the case of the monitor service, the sending of events can be controlled, but is limited to dynamic Status Variables.

The current DMT Admin service has no attributes properties on its nodes to be used to configure notification behavior, such as active notification and passive notification defined in TR-069. Therefore, a remote manager cannot control the notification behavior of DMT nodes in a standardized way.

To conclude, the current options, as provided in the Mobile specification, limit notification of parameter changes to StatusVariables, explicitly enabled for monitoring. There is no standardized approach available to monitor changes on any node in the DMT.

### 4.2.2 Limitations in the number of services available in the DMT

The OSGi R4 Mobile specification mapped a number of services to the DMT tree. However, these services are limited to the services listed in the Mobile Specification. Other interesting services, as listed in the OSGi R4 Service Compendium are not yet mapped (standardized) in the DMT tree:

The DMT tree defined in the Mobile Specification contains objects for the following services:

- Configuration Admin service
- Log service
- Monitor Admin service
- Application Admin service
- Conditional Permission Admin service
- Deployment service

A number of areas that could be of interest to a remote manager are currently missing in the DMT tree:

- Startlevel management
- Bundle management: managing individual bundles as opposed to deployment packages (inventory, life-cycle management, exported services, …)
- Service management: getting a remote view on services registered in the service registry
- Permission Admin management
4.2.3 Mapping TR-069 to the OMA-DM inspired DMT model

Within the OSGi Mobile specification, the choice has been made to model the DMT after OMA-DM.

As a result, creating an OMA-DM protocol adapter is quite straightforward. Although no major hurdles have been identified in creating a TR-069 protocol adapter, it is less straightforward:

- The TR-069 RPC primitives have to be translated to the DMT Admin service interface methods (which are OMA-DM RPC inspired).
- The TR-069 tree has to be mapped to the DMT tree. Translating object model specific features like DMT meta nodes, or TR-069 attributes is not straightforward. It might require specific extensions to the DMT, e.g. to support TR-069 attributes, or a single node in the DMT might result in multiple objects in a TR-069 data model, etc.
- The TR-069 data types have to be mapped to the DMT Admin data types. However, TR-069 data types, such as “unsignedint” and “dateTime” (ISO 8601), cannot be translated appropriately into DMT Admin data types defined in the current specification. Translating these data types might result a limitation of the available value range and a complex object that consists of multiple nodes, respectively.

4.3 Conclusion

The OSGi Mobile specification delivers a standardized data model (the DMT), and standardized interface (on the DMT Admin service) to enable remote management through a protocol adapter. However, the current specification lacks management objects for a number of interesting areas. Also, there is some support lacking for TR-069 notifications. Furthermore, since the DMT model is OMA-DM inspired, implementing a TR-069 protocol adapter is not straightforward, although not impossible.

5 Requirements

REQUIREMENT[1]: A management tree, which is mappable to multiple remote management protocols, MUST be standardized. The solution MUST be mappable at least to OMA DM and TR-069 protocols. The model MUST be open to add new protocols in the future, like a possible common solution to substitute OMA-DM and TR-069.

REQUIREMENT[2]: A bundle MAY implement a non-standard sub-tree of the management tree. The solution MUST support the management of this type of sub-trees. As such, it MUST define the interface to be implemented by the bundle to support this management.

REQUIREMENT[3]: The management tree SHOULD cover bundle life cycle management and service monitoring.
6 Technical Solution

This RFC defines the Residential Management Tree which is handled via DMT Admin service as it is available on an OSGi Residential Platform. The protocol used between the remote server and the device is not specified, but it is expected that the TR-069 protocol will be the management protocol used to manipulate this tree.

Although the top level nodes of this tree depend on the user policy, this RFC supposes that TR-106 structural requirements defined by the Broadband Forum [6] will be adopted by the Residential Management Tree architecture. The top level nodes of a tree adhering to TR-106 are depicted in Figure 1 (See TR-106 Amendment1[7]). The partial tree, enclosed by the dashed line, is specified by TR-106, which is expected to be used with the Residential Management Tree in many circumstances in residential service domain. This RFC does not define any restrictions on the architecture enclosed in the dashed line, and the ancestor nodes of /Device/Services/OSGi node can be arbitrarily defined by users. Therefore, the parent node of the OSGi node is referred to as “$” in the following sections.

The OSGi Residential Management Tree is a relative tree. This tree consists of a number of distinct parts as shown in Figure 1. Each of the sub-trees in the figure is explained in the following sections. Users of this tree may add a user-defined sub-tree under the $/OSGi/<instance_id> node. Moreover, there is no restriction on the use of sub-trees defined in Mobile Management Tree [8] as a user-defined sub-tree in the Residential Management Tree.
$/OSGi/<instance_id> node is used to represent an instance of the OSGi framework on a device. In most situations there is only one instance of OSGi, so only $/OSGi/1 is available. However, in some cases such as a remotely managed proxy of OSGi implemented devices in the network, there may be several OSGi instances in the Residential Management Tree. In this case, the local OSGi framework on which the Residential Management Tree is implemented should be identified, so that the DMT Admin implementation can access the indicated node path via appropriate way. Therefore, $/OSGi/Local node must be the alias of $/OSGi/<instance_id>, which presents information of the local OSGi Framework. The pairs between each OSGi Framework and the corresponding <instance_id> must be kept persistently beyond restart of the local OSGi Framework and reconnection of remote OSGi Framework.

The Residential Management Tree adopts the complemental data model with the Mobile Management Tree defined in the OSGi Mobile Specification. On the one hand, some top level nodes, Configuration, Policy and Log, have definitions similar to the Mobile Specification. On the other, the Residential Management Tree has some original sub-trees: Framework, BundleState, PackageState, ServiceState, Filters, and BundleResources.

The Framework sub-tree is to manipulate the OSGi framework on which this management tree is implemented and to control the life cycle of installed bundles instead of the Deployment sub-tree defined in the Mobile Specification. The BundleState sub-tree is used to derive information of individual bundles. The PackageState and ServiceState sub-trees provide information of available Packages and Services on the OSGi framework, respectively. The Filters sub-tree is used to filter information contained in a tree. The BundleResources sub-tree is used to derive resources in a bundle jar file.

Basically, the structure of the sub-trees in the Residential Management Tree corresponds to the one defined in “RFC-139 JMX Control of OSGi”, which defines four interfaces for OSGi framework core APIs; FrameworkMBean,
BundleStateMBean, PackageStateMBean and ServiceStateMBean. Although the fundamental architecture adheres to the API of RFC-139, some features such as command execution and structural data exchange are adjusted to realize functionality in a hierarchical object tree, because the Java interfaces are difficult to map completely to the Residential Management Tree which adheres to the DMT Admin model. One of the biggest differences is that returned values of OSGi Core API's methods cannot be returned in DMT Admin interfaces.

6.1 Legend

All nodes of the Residential Management Tree are described in a table format. This table format defines the following meta information:

- **Add** – An x indicates that the implementation must support the creation of the given node by the management system.

- **Get** – An x indicates that the implementation must support retrieval of the properties of the given node (including the value).

- **Replace** - An x indicates that the implementation must support setting the value of the given node. Support for changing other properties is optional. Note, that this column does not correspond to the node attribute changing, which can be provided by an implementation even if the node value cannot be changed, for example in case it supports setting the Title property.

- **Delete** - An x indicates that the implementation must support deletion of the given node by the management system.

- **Exec** - An x indicates that the implementation must support the execute operation for the given node.

- **Type** - The node type for an interior node, or the data type for a leaf node. The following data types are defined: str, int, float, date, time, bin, xml, bool, b64.

- **Cardinality** - The range of occurrences of the given node. * means infinite.

- **Scope** - The scope indicates the creation strategy. It can have the following values:
  - **P** - Permanent. A permanent node cannot be changed by the management system. It can, however, appear due to an internal device event, for example, the addition of a network interface.
  - **D** - Dynamic. A node that must be created by the management system. Such a creation can then automatically create other nodes.
  - **A** - Automatic. A node that is created automatically by a managed object if its parent node is created.

6.2 Framework Object

The *Framework* Object is a managed object that allows manipulation of the OSGi framework functions; StartLevel configuration, Bundle install, Framework Lifecycle control and Bundle lifecycle control. The *Framework* Object is an optional managed object, therefore this object does not have to be implemented if the user does not need to manipulate the framework.

The tree structure of *Framework* Object can be accessed from the $/OSGi/<instance_id>/Framework sub-tree. Figure 2 shows the structure of the *Framework* Object sub-tree.
The Framework Object consists of 5 parts; StartLevel, InstallBundle, FrameworkLifecycle, BundleControl and Ext. These sub-trees represent the individual functions that are manipulated through the Residential Management Tree.

The StartLevel sub-tree contains RequestedStartLevel, ActiveStartLevel and InitialBundleStartLevel. RequestedStartLevel and InitialBundleStartLevel are Start Level configurations of the OSGi framework. ActiveStartLevel is a read-only node, not writable, which represents the start level of the framework at the moment. The FrameworkLifecycle sub-tree controls the state of the OSGi framework itself, but the detail behaviors corresponding to these commands depend on the implementation of the OSGi framework. The InstallBundle sub-tree is used to install bundles into the OSGi framework. This sub-tree supports the simultaneous installation of multiple bundles. The BundleControl sub-tree controls a bundle start level and lifecycle of an individual bundle after installation. The Ext sub-tree extends the data model of the Framework Object.

The Framework Object must support transactions because all changes to the $/OSGi/<instance_id>/Framework tree must be done in an atomic session to keep the OSGi Framework consistent. Only atomic sessions can perform the required changes to the node values.

The following introduces the steps taken to install a bundle. At first, $/OSGi/<instance_id>/Framework/InstallBundle/<id> node must be created by ReadWriteDataSession#createInteriorNode() method, its children will be created automatically by the Framework Object. Next, the value of Location and URL must be set by ReadWriteDataSession#setNodeValue() method. After that, when TransactionalDataSession#commit() method is called, InstallBundle sub-tree should call BundleContext#installBundle(String, InputStream) method where the first argument is the Location and the second argument is the InputStream object retrieved from the specified URL.

[REMARK] If commit() method is called BEFORE any value of Location and URL node is set by setNodeValue() method, InstallBundle sub-tree must NOT call installBundle() method. If commit() method is called AFTER either
of them is set, `installBundle()` method must be called. Multiple bundles can be installed by just one commit() if multiple $/OSGi/<instance_id>/Framework/InstallBundle/<id> are created before one commit() is called.

The `BundleControl` sub-tree provides bundle start level and life cycle operations of an individual bundle after installation. Regarding bundle life cycle control, the operation is basically conducted by indicating the desired state of the bundle; Active, Resolved or Uninstalled in the $/OSGi/<instance_id>/Framework/BundleControl/<bundle_id>/Lifecycle/DesiredState node. This sub-tree supports lazy activation and transient start (or stop). When commit() is called, `Bundle#start(int option)` or `Bundle#stop(int option)` is called where the argument is the value set in the $/OSGi/<instance_id>/Framework/BundleControl/<bundle_id>/Lifecycle/Option node. The value of `Option` node must be retained beyond the data plug-in bundle reboot. Details of the values are shown in Table 6.1

[REMARK] A Protocol Adapter (or local manager) should set not only “Resolved” or “Active” in DesiredState but also `Option` node in same transactional session. The reason is that if a protocol adapter (or a local manager) sets only the `DesiredState` node, not the `Option` node, a value that the remote manager does not expect might be kept in the `Option` node. In that case, the operation resulting from this value is likely to differ from what the remote manager expects.

Note that `Update` is indicated directly as a command because this kind of operation does not represent the specific desired state of a bundle. The value set to `Update` node is the string of the new bundle's URL. The `OperationResult` node represents the result of the latest operation on this bundle.

To use the TR-069 protocol to control the Residential Management Tree, the node name of $/OSGi/<instance_id>/Framework/InstallBundle/<id> should be represented by a numeric character string, not a literal node name such as Location or URL. The reason is that TR-069 has only the RPC called “AddObject” to create a new node in a tree, and the RPC can take as an argument the path name of the collection of objects for which new nodes are to be created.

[REMARK] According to TR-069, a dynamic node path in the Residential Management Tree should be defined as numeric character. Therefore, the path name of the new node should be assigned as an instance number by incrementing the number; the management system cannot specify the identifier of the new node.

All nodes for the `Framework` Object sub-tree are explained in Table 6.1.

**Table 6.1 Framework sub-tree Nodes**

<table>
<thead>
<tr>
<th>URI</th>
<th>Add</th>
<th>Get</th>
<th>Replace</th>
<th>Delete</th>
<th>Exec</th>
<th>Type</th>
<th>Cardinality</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>P</td>
<td>Framework Root node.</td>
</tr>
<tr>
<td>Framework/StartLevel</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>P</td>
<td>Interior node that contains the values for StartLevel configuration. If the StartLevel service is unavailable, this node and its child nodes must not be created.</td>
</tr>
<tr>
<td>Framework/StartLevel/RequestedStartLevel</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>int</td>
<td>1</td>
<td>P</td>
<td>A leaf node used to configure the Framework's StartLevel. When this node value is replaced or the Bundles sub-tree starts, <code>StartLevel#setStartLevel</code> with the specified value must be called. This value must be kept</td>
</tr>
<tr>
<td>URI</td>
<td>Add</td>
<td>Get</td>
<td>Replace</td>
<td>Delete</td>
<td>Exec</td>
<td>Type</td>
<td>Cardinality</td>
<td>Scope</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>---------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>Framework/StartLevel/ActiveStartLevel</td>
<td>X</td>
<td></td>
<td>int 1</td>
<td>P</td>
<td>A leaf node that contains the Framework's current StartLevel. This node is read-only to get the Framework's StartLevel.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/StartLevel/InitialBundleStartLevel</td>
<td>X</td>
<td>X</td>
<td>int 1</td>
<td>P</td>
<td>A leaf node used to configure the initial bundle StartLevel. When this node value is replaced or the Bundles sub-tree starts, StartLevel#setInitialStartLevel with the specified value must be called.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/FrameworkLifecycle</td>
<td>X</td>
<td></td>
<td>node 1</td>
<td>P</td>
<td>Interior node that contains the values for Lifecycle control of the OSGi Framework.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/FrameworkLifecycle/Restart</td>
<td>X</td>
<td>X</td>
<td>bool 1</td>
<td>P</td>
<td>A leaf node used to restart the OSGi Framework. This node is writable to set the restart command. If this node value is replaced with 'TRUE', the Framework sub-tree must restart the OSGi Framework.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/FrameworkLifecycle/Shutdown</td>
<td>X</td>
<td>X</td>
<td>bool 1</td>
<td>P</td>
<td>A leaf node used to shutdown the OSGi Framework. This node is writable to set the shutdown command. If this node value is replaced with 'TRUE', the Framework sub-tree must shutdown the OSGi Framework.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/FrameworkLifecycle/Update</td>
<td>X</td>
<td>X</td>
<td>bool 1</td>
<td>P</td>
<td>A leaf node used to update the OSGi Framework. This node is writable to set the update command. If this node value is replaced with 'TRUE', the Framework sub-tree must update the OSGi Framework.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/InstallBundle</td>
<td>X</td>
<td></td>
<td>node 1</td>
<td>P</td>
<td>Interior node that contains the values for bundle installation.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/InstallBundle/&lt;id&gt;</td>
<td>X</td>
<td>X</td>
<td>node</td>
<td>D</td>
<td>Interior node that represents a bundle which will be installed to the OSGi Framework. This node is created for each bundle to be installed. When a transaction is committed, the Framework Object attempts to install the bundle specified under this node. See details of the installation operation described in the Location definition. This node must be stored persistently unless the Framework Object deletes it after installation succeeds.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Framework/InstallBundle/<id>/ | X | X | str 1 | A | A leaf node used to set the bundle location under
## Location

<table>
<thead>
<tr>
<th>URI</th>
<th>Add</th>
<th>Get</th>
<th>Replace</th>
<th>Delete</th>
<th>Exec</th>
<th>Type</th>
<th>Cardinality</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework/InstallBundle/&lt;id&gt;/URL</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td>A leaf node used to be set the bundle’s jar file URL from which the specified bundle is installed. This node is writable because the node is used to install a bundle. See details of the installation operation described in the Location definition.</td>
</tr>
<tr>
<td>Framework/InstallBundle/&lt;id&gt;/Error</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>0,1</td>
<td>A leaf node used to represent the reason for an installation failure. When the installation of a bundle fails, the Framework Object should create this node and set a value specifying the error reason.</td>
</tr>
<tr>
<td>Framework/BundleControl</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td>Interior node that contains the control functions for each bundle.</td>
</tr>
<tr>
<td>URI</td>
<td>Add</td>
<td>Get</td>
<td>Replace</td>
<td>Delete</td>
<td>Exec</td>
<td>Type</td>
<td>Cardinality</td>
<td>Scope</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----</td>
<td>-----</td>
<td>---------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Framework/BundleControl/ &lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 0..*</td>
<td>A</td>
<td>A interior node that represents a Bundles instance. This number must equal the bundle id, which Bundle#getBundleId() returns.</td>
</tr>
<tr>
<td>Framework/BundleControl/ &lt;bundle_id&gt;/BundleStartLevel</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>int 1</td>
<td>A</td>
<td>The configuration node for the StartLevel of the bundle. When the node value is gotten, this object must return the current StartLevel of the bundle. If this value is changed, StartLevel#setBundleStartLevel with the specified value must be called.</td>
</tr>
<tr>
<td>Framework/BundleControl/ &lt;bundle_id&gt;/Lifecycle</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>A</td>
<td>This is a parent node for commands related to the bundle’s life-cycle control. This sub-tree is described in detail later.</td>
</tr>
<tr>
<td>Framework/BundleControl/ &lt;bundle_id&gt;/Lifecycle/ DesiredState</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>int 1</td>
<td>A</td>
<td>A leaf node used to control the bundle’s life-cycle. When this node value is replaced with the state described below or the BundleControl sub-tree starts, the BundleControl sub-tree must change the bundle life-cycle to the specified bundle state. The BundleControl sub-tree must return an error, if the specified state cannot be understood. This state must be one of the following: 0 – Uninstalled 1 – Resolved 32 – Active 64 – Active</td>
</tr>
<tr>
<td>Framework/BundleControl/ &lt;bundle_id&gt;/Lifecycle/Update</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str 1</td>
<td>A</td>
<td>A leaf node used to update the bundle. This node is writable to set the update command. When this node value is replaced with an URL string, Bundle#update(InputStream) must be called where InputStream is the specified URL. If the specified URL is an empty string, Bundle#update() must be called so that the specified bundle is updated with the jar-file indicated by the BundleLocation. If the update fails, the Bundle object does not have to retry the operation. If DesiredState and Update are replaced during same transaction, BundleControl sub-tree must put update operation ahead of life cycle operation attributed by changing DesiredState Node value.</td>
</tr>
<tr>
<td>Framework/BundleControl/ &lt;bundle_id&gt;</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>int 1</td>
<td>A</td>
<td>A leaf node used to set start or stop option. A</td>
</tr>
</tbody>
</table>
### URI

<table>
<thead>
<tr>
<th>Add</th>
<th>Get</th>
<th>Replace</th>
<th>Delete</th>
<th>Exec</th>
<th>Type</th>
<th>Cardinality</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Protocol Adapter (PA) (or local manager) should set not only DesiredState but also this node in the same transactional session. After called commit() method, Lifecycle sub-tree must call Bundle#start(int option) or Bundle#stop(int option). The value of this node must be kept beyond the data plug-in bundle reboot. Option node contains integer format data as below. 0:default 1:START_TRANSIENT or STOP_TRANSIENT 2:START_ACTIVATION_POLICY 3:START_ACTIVATION_POLICY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Example 1) If a Protocol Adapter wants to start a Bundle transiently, the Protocol Adapter should set “Active” to DesiredState node and “1” to Option node. (Example 2)If a Protocol Adapter wants to start a bundle not transiently, the Protocol Adapter should set DesiredState node to “Active” and Option node to “0”. (Example 3)If a Protocol adapter wants to stop a bundle transiently, it should set DesiredState node to “Resolved” and this node to “1”.</td>
</tr>
</tbody>
</table>

### Framework/BundleControl/

| <bundle_id>/Lifecycle/ | X | str | 1 | A | This node holds the latest operation’s result which is conducted through the Framework/BundleControl/<bundle_id>/Lifecycle sub-tree, so that the Management System can derive the result of the bundle life-cycle operation. If the operation succeeds, the value string must be start with “Success: “. Otherwise it must start with “Fail” |

### Framework/Ext

| X | node | 1 | P | Interior node that can contain values for user extensions. |
6.3 BundleState Object

Figure 3 shows the BundleState sub-tree architecture. This sub-tree is used to obtain the information of a bundle from the OSGi framework. When a bundle is installed on the OSGi framework, a new node of 

$/OSGi/<instance_id>/BundleState/<bundle_id>

is automatically added and <bundle_id> is incremented. In other words, <bundle_id> is equivalent to Bundle#getBundleId(). The identifiers including SymbolicName and Version are used to identify a bundle on the OSGi framework. BundleType, an integer value, represents the type of the bundle. Manifest represents the Manifest header by String value. StartLevel shows the bundle's start level on this OSGi framework. State shows the bundle state; Installed, Resolved, Starting, Active, Stopping and Uninstalled. Location, a String value, represents the BundleLocation of the bundle. Fragments, Hosts, Required, Requiring and Signers sub-trees contain the corresponding bundle identifiers.

The BundleState Object must be kept after uninstallation of the bundle until the org.osgi.framework.Bundle object of the bundle will be deleted from the OSGi Framework. In other words, once created, the BundleState Object of a bundle must remain available while the OSGi Framework continues to run.

All nodes for the BundleState Object sub-tree are explained in Table 6.2.

Table 6.2 BundleState sub-tree Nodes

<table>
<thead>
<tr>
<th>URI</th>
<th>Add</th>
<th>Get</th>
<th>Replace</th>
<th>Delete</th>
<th>Exec</th>
<th>Type</th>
<th>Cardinality</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BundleState</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>A</td>
<td>This is a parent node for status information of the bundle. This sub-tree is detailed later.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 0..*</td>
<td>A</td>
<td>A node that represents a Bundles instance. This</td>
</tr>
<tr>
<td>URI</td>
<td>Add</td>
<td>Get</td>
<td>Replace</td>
<td>Delete</td>
<td>Exec</td>
<td>Type</td>
<td>Cardinality</td>
<td>Scope</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>---------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/SymbolicName</td>
<td>X</td>
<td>str</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>The Bundle-SymbolicName of the bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Version</td>
<td>X</td>
<td>str</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>The version of the bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/BundleType</td>
<td>X</td>
<td>int</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>A node indicating the type of the bundle. The node value must be equivalent to the value of PackageAdmin#getBundleType().</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Manifest</td>
<td>X</td>
<td>str</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>This node is a leaf node that contains Manifest headers as String. The node implementation must get Dictionary object by Bundle#getHeaders(). For converting the Dictionary object to String, this node must have the value adhering to the following format (LF means linefeed): &quot;&lt;key&gt;:&lt;value&gt;LF&lt;key&gt;:&lt;value&gt;LF...&quot; where order of appearance depends on its implementation.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Status</td>
<td>X</td>
<td>node</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>This node is the parent of the nodes that represent the status of the bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Status/Location</td>
<td>X</td>
<td>str</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>The BundleLocation of the bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Status/State</td>
<td>X</td>
<td>int</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>The state of the bundle as returned by Bundle#getState(). This state is one of the following: 0 – Not Available 2 – Installed 4 – Resolved 8 – Starting 16 – Stopping 32 – Active</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Status/StartLevel</td>
<td>X</td>
<td>int</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>The StartLevel of the bundle. Because this value is a read-only node in this sub-tree, user must use the Framework Object to configure the StartLevel.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Status/PersistentlyStarted</td>
<td>X</td>
<td>bool</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>The status of the bundle at the last shutdown of the OSGi Framework. If the bundle is Active when the OSGi Framework shutdowns, the value must be TRUE. Otherwise, the value must be FALSE.</td>
</tr>
<tr>
<td>URI</td>
<td>Add</td>
<td>Get</td>
<td>Replace</td>
<td>Delete</td>
<td>Exec</td>
<td>Type</td>
<td>Cardinality</td>
<td>Scope</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>---------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Status/LastModified</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>date 1</td>
<td>A</td>
<td>The latest time at which the bundle has been modified.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Host</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>A</td>
<td>This node is the parent of nodes that specify the bundle hosting this bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Host/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>0,1</td>
<td>A</td>
<td>A leaf node that represents the bundle ID of the host bundle. If there is no host bundle, this node must not exist. This node does not have any value, so the default value should be an empty string.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Fragments</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td>A</td>
<td>This node is the parent of nodes that specify the fragment bundles of this bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Fragments/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>0..*</td>
<td>A</td>
<td>A leaf node that represents the bundle ID of a fragment bundle. If there is no fragment bundle, this node must not exist. This node does not have any value, so the default value should be an empty string.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Required</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td>A</td>
<td>This node is the parent of nodes that specify the bundles requiring this bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Required/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>0..*</td>
<td>A</td>
<td>A leaf node that represents the bundle ID of a required bundle. If there is no required bundle, this node must not exist. This node does not have any value, so the default value should be an empty string.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Requiring</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td>A</td>
<td>This node is the parent of nodes that specify the bundles required by this bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Requiring/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>0..*</td>
<td>A</td>
<td>A leaf node that represents the bundle ID of requiring bundles. If there is no requiring bundle, this node must not exist. This node does not have any value, so the default value should be an empty string.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Signers</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td>A</td>
<td>This node is the parent of nodes that specify the signers of the bundle.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/Signers/&lt;id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td>A</td>
<td>A signer of the bundle. The node value is the semicolon-separated list of the Subjects (distinguished Name) of the X.509 Certificate.</td>
</tr>
</tbody>
</table>
URI

<table>
<thead>
<tr>
<th>Add</th>
<th>Get</th>
<th>Replace</th>
<th>Delete</th>
<th>Exec</th>
<th>Type</th>
<th>Cardinality</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>chain. The last item is the root.</td>
</tr>
<tr>
<td>BundleState/&lt;bundle_id&gt;/BundleStateExt</td>
<td></td>
<td>node 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
</tr>
</tbody>
</table>

### 6.4 PackageState Object

The PackageState Object is a managed object that allows the package information to be derived. This object can be used to retrieve package dependencies between bundles.

![Diagram of PackageState Object](image)

Figure 4 shows the overall architecture of the PackageState object. The 
$/OSGi/<instance_id>/PackageState/<id>$ node is created for an individual package existing on the OSGi framework. This node represents a package's information including bundle dependencies. The pairs between each package and the corresponding <id> must be kept persistently as long as the package is exported from the same bundle, which means the same bundle ID, beyond restart of the OSGi Framework.

The Name node contains a qualified package name, and the Version node contains the version number of the package. The ExportingBundle node shows a bundle identifier exporting the package. On the other hand, the ImportingBundles sub-tree shows the bundles importing the package. These nodes can be used to get information on packages shared between bundles.
The PackageState Object may support only readable session because it does not contain any writable node.

All nodes for the PackageState Object sub-tree are explained in Table 6.3.

Table 6.3 PackageState sub-tree Nodes

<table>
<thead>
<tr>
<th>URI</th>
<th>Add</th>
<th>Get</th>
<th>Replace</th>
<th>Delete</th>
<th>Exec</th>
<th>Type</th>
<th>Cardinality</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PackageState</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>P</td>
<td>PackageState Root node containing package information existing on the OSGi Framework. The children of this node must represent the actual package status when this sub-tree is accessed.</td>
</tr>
<tr>
<td>PackageState/&lt;id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 0..*</td>
<td>A</td>
<td>A node that represents the Package instance.</td>
</tr>
<tr>
<td>PackageState/&lt;id&gt;/Name</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str 1</td>
<td>A</td>
<td>The qualified name of the package.</td>
</tr>
<tr>
<td>PackageState/&lt;id&gt;/Version</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str 1</td>
<td>A</td>
<td>The version of the package.</td>
</tr>
<tr>
<td>PackageState/&lt;id&gt;/RemovalPending</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bool 1</td>
<td>A</td>
<td>A leaf node that represents the removal status of the package. If a bundle exporting the package has already been uninstalled or updated but the package is still used, this node must be TRUE.</td>
</tr>
<tr>
<td>PackageState/&lt;id&gt;/ExportingBundle</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>A</td>
<td>This node is the parent of node that specify the bundle exporting the package.</td>
</tr>
<tr>
<td>PackageState/&lt;id&gt;/ExportingBundle/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str 1</td>
<td>A</td>
<td>A leaf node that represents the bundle ID of the exporting bundle. This node does not have any value, so the default value should be an empty string.</td>
</tr>
<tr>
<td>PackageState/&lt;id&gt;/ImportingBundles</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>A</td>
<td>A node that contains ids of bundles importing the package.</td>
</tr>
<tr>
<td>PackageState/&lt;id&gt;/ImportingBundles/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str 0..*</td>
<td>A</td>
<td>A node that represents the bundle ID of importing bundles. If there is no importing bundle, this node must not exist. This node does not have any value, so the default value should be an empty string.</td>
</tr>
</tbody>
</table>
6.5 ServiceState Object

The ServiceState Object is a managed object that allows the service information to be derived. This object can be used to retrieve service dependencies between bundles.

![ServiceState Object Diagram]

Figure 5 shows the overall architecture of the ServiceState object. The $/OSGi/<instance_id>/ServiceState/<service_id>$ node is created for an individual service instance existing on the OSGi framework. This node represents a service's information including registering bundle and using bundles. Registering a service to the OSGi framework automatically adds a new Service instance under the $/OSGi/<instance_id>/ServiceState node with incrementation of <service_id>. In other words, <service_id> equals the service.id of the service instance in the OSGi framework.

The Properties node contains service properties of the service, which include the interface names implemented by the service. The Properties node must contain all service properties which consist of string, boolean or numeric data types including single-dimension arrays or vectors. However, non-serializable data types can be discarded from the Properties sub-tree, since these types of properties are difficult to be represented in object trees. The RegisteringBundle node shows the id of the registering bundle. On the other hand, the UsingBundles sub-tree shows bundles using the service. These nodes can be used to get information on the relationships between registering bundle and using bundles.
In order to get service interface names of a service, "/OSGi/<instance_id>/ServiceState/<service_id>/Properties/objectClass/Values/<n> node can be used.

The ServiceState Object may support only readable session because it does not contain any writable node.

All nodes for the ServiceState Object sub-tree are explained in Table 6.4.

<table>
<thead>
<tr>
<th>Table 6.4</th>
<th>ServiceState sub-tree Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>Add</td>
</tr>
<tr>
<td>ServiceState</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;/Type</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;/Cardinality</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;/Values</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;/Values/&lt;n&gt;</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/RegisteringBundle</td>
<td>X</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/RegisteringBundle/&lt;bundle_id&gt;</td>
<td>X</td>
</tr>
</tbody>
</table>
6.5.1 Service Property Dictionary nodes

The service property Dictionary consists of key-value pairs. The service property Dictionary is mapped to a sub-tree. The URI for a service property item is the following:

$/OSGi/<instance_id>/ServiceState/<service_id>/Properties/<key>

Key nodes are interior nodes. Their type, cardinality, and value are represented as separate nodes. These sub-nodes are:

- **Type** contains the Java type name like `java.lang.Float`, `String`, etc.

- **Cardinality** – Defines if the value is a scalar, an array, or a vector. It can take the following values:
  - scalar – For simple, unstructured values, like a string or a byte[].
  - array – When the value is a Java array (but not byte[]).
  - vector – When the value must be a Java Vector object.

- **Values** – The children of this node must be named with an integer that starts at zero for the first element, and increases by one with each additional element. If the Cardinality is scalar, there is one child node named “0”.

The actual value (Values child nodes) is mapped to a DmtData type if possible. If this mapping is not possible, the node must be a str node and the Java class of the given type must be able to parse the value in a constructor.
6.6 Filters Object

The Filters Object is a managed object that searches the nodes in a tree that correspond to the filter expression. This Filter Object is a generic mechanism for the whole management tree below the $/OSGi node. In other words, the Filter Object plugin must be able to filter all nodes in the tree except Filters Object itself and Log Object because the Log Object already has an original filtering mechanism.

The Filters Object can be used to group bundles, packages, services and other information in the tree. Since the filter string set in the Filter node has no restriction in terms of its usage policy, users can use this function in accordance with their needs.

The $/OSGi/Filters object is used to search nodes by filtering values or names of nodes located under the sub-tree specified by $/OSGi/Filters/<search_id>/TargetSubtree. At first, the user needs to create the $/OSGi/Filters/<search_id> node by incrementing the <search_id> number, which should be a numeric character string as demanded by the TR-069 protocol. Then the user sets the desired partial path in the $/OSGi/Filters/<search_id>/TargetSubtree node, that specifies the sub-tree required to provide information as the result, and sets an appropriate filter string in $/OSGi/Filters/<search_id>/Filter. When the user accesses under the $/OSGi/Filters/<search_id>/Result node, the sub-trees that match the filter are extracted from the sub-trees specified by $/OSGi/Filters/<search_id>/TargetSubtree and are copied as children sub-trees of the $/OSGi/Filters/<search_id>/Result node. Therefore, the sub-trees that match the filter string are aligned under the $/OSGi/Filters/<search_id>/Result node with the absolute path from the $ node.

The <search_id> node is a dynamic node which means that the path name of the new node should be assigned as an instance number by incrementing the largest existing number as demanded by TR-069. Therefore, the node name should be defined as numeric character in the Residential Management Tree.

All nodes for the Filters Object sub-tree are explained in Table 6.5.
<table>
<thead>
<tr>
<th>URI</th>
<th>Add</th>
<th>Get</th>
<th>Replace</th>
<th>Delete</th>
<th>Exec</th>
<th>Type</th>
<th>Cardinality</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filters</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>P</td>
<td>The root node of Filters Object.</td>
</tr>
<tr>
<td>Filters/&lt;search_id&gt;</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 0..*</td>
<td>D</td>
<td>Represents a filter search request. This node is created to set the filter expression and to get the sub-trees, which match with the filter expression. The &lt;search_id&gt; is assigned in ascending order to Filters instances when they are set. This is a unique ID of Filters instance and must be kept persistently.</td>
</tr>
<tr>
<td>Filters/&lt;search_id&gt;/TargetSubtree</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td>A</td>
<td>Partial path to the sub-tree under which Filter is going to be matched. This must be the absolute path of the top node name of the sub-tree. The default value of this node is an empty string, which means no filtering must be done. This value must be kept persistently.</td>
</tr>
<tr>
<td>Filters/&lt;search_id&gt;/Filter</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td>A</td>
<td>Contains the filtering expression. The filter must be given in the OSGi Filter format. See Filter and Target Expression section. An empty string indicates that no filtering must be done. An empty string is the default value for this node. This value must be kept persistently.</td>
</tr>
<tr>
<td>Filters/&lt;search_id&gt;/Result</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td>A</td>
<td>Matched sub-trees are stored under the Result node. Any children of this node must represent the actual status of the sub-tree whenever the children nodes in this sub-tree are accessed. The result must be only those sub-trees that satisfy the specified Filter node. And the filtered sub-trees must include all nodes located under the sub-tree specified by the TargetSubtree node. See Filter and Target Expression section.</td>
</tr>
</tbody>
</table>

### 6.6.1 Filter and Target Expression

The expression of $/OSGi/Filters/<search_id>/TargetSubtree must be a partial path from the “$” node to the top node of the target sub-tree to be filtered and must end with “/”, which implies that a partial path must indicate an interior node. The characters “*” and “-” must be allowed to be used in the path as a wild-card; “*” indicates a wildcarded node for 1 level, while “-” indicates multiple levels in a node path respectively. Multiple wild-card can be used in a node path. If an incorrect string, such as a string including an incorrect format path, is specified in $/OSGi/Filters/<search_id>/TargetSubtree, the Filter Object should throw a DmtException to the caller and should keep the TargetSubtree node value as an empty string.
The filter expression of \$/OSGi/Filters/<search_id>/Filter should be a LDAP search filter (RFC 1960. See Filter for a description of the filter string syntax.). This expression is equivalent to the OSGi Filter format.

The key string in a filter must be a node name which exists in the sub-tree specified by \$/OSGi/Filters/<search_id>/TargetSubtree. Therefore it must not contain “/”. All nodes equivalent to the specified node name must be the targets for the filtering. If the indicated node is not included in the sub-tree, the Filters Object must not create sub-trees under the Result node. If there exists multiple nodes with the indicated node name under the specified TargetSubtree, the Filters Object must perform the filtering for all nodes that match the node name. A wild-card must not be allowed to be used in a key string. Both interior and leaf node can be specified as the key string of a filter.

The value string in a filter indicates either a leaf node value or a node name. If an interior node is specified as the key string, node names of children nodes of the specified interior node must be recognized as the target values to be matched. On the other hand, if a leaf node is specified, the value of the specified leaf node must be recognized as the target value. A wild-card can be used at the end of the value string to conduct prefix-searches in either cases.

In a filter expression, types of value should be ignored because the Filter Object must recognize the filter strings as String. Therefore, the filter matching operation demands that the filter value should be translated into the type appropriate for each leaf node.

For the service properties sub-trees in the ServiceState Object and the configuration properties sub-trees in the Configuration Object, such as \$/OSGi/<instance_id>/ServiceState/<service_id>/Properties and \$/OSGi/<instance_id>/Configuration/<pid>/Keys respectively, the filtering must be done in a different way from other sub-trees in the following three respects:

1. For those sub-trees, the value of \$/OSGi/Filters/<search_id>/Filter must be described in the same way as in the properties filtering implemented by BundleContext#getServiceReferences() method.

2. A key name of service property or configuration property may conflict names of other nodes under the specified TargetSubtree. Therefore, to match against service properties or configuration properties, a key must be prefixed with the commercial at sign ‘@’ (0u0040) in a filter expression. It means, a remaining key string without “@” indicates the property key to be searched, and the value string represents the desired value of the property. For example, “@objectClass” will refer to a service property with the name “objectClass”.

3. The key and value in the filter string are processed in a case sensitive manner unless the key name references a service property or configuration property, which are case insensitive.

Assume that the key string of a filter is prefixed with “@”. The Filter Object must recognize the nodes which has the name as same as the remaining key string without “@” under the service properties subtree or the configuration properties subtree, or the nodes which has the name as same as the key string including “@” under other subtrees.

When the \$/OSGi/Filters/<search_id>/Result sub-tree is accessed, the returned sub-tree must reflect the current situation; the detailed mechanism of the synchronization depends on the implementation. When the node is accessed, the filter matching sub-trees are created. The Filters Object searches nodes that satisfy the specified keys and values in \$/OSGi/Filters/<search_id>/Filter against the sub-trees specified by \$/OSGi/Filters/<search_id>/TargetSubtree, and must create a matched sub-tree under the \$/OSGi/Filters/<search_id>/Result node. The absolute node path from the $ node must be created with the actual node name of the wild-card appearing in the node path specified by \$/OSGi/Filters/<search_id>/TargetSubtree.
The sub-trees under the $/OSGi/Filters/<search_id>/Result must be read-only in order to keep consistency among data-plugins related to the filter search. The Filter Object must prevent attempts to access the sub-tree to change node value or properties, and must throw DmtException to the caller.

The following expressions are examples of the Filter and TargetSubtree:

[Case1] Simple TargetSubtree and Filter usage

- TargetSubtree: $/OSGi/1/Bundles/
- Filter: (SymbolicName=org.osgi.*)
  - Bundles whose Bundle-SymbolicName correspond to "org.osgi.*" are matched. The Result node contains corresponding sub-trees that have absolute node paths descending from $ node. A possible sub-tree under the Result node is described below ($ node should be changed to the actual node path depending on each execution environment):

```
Result
  .$  
  ./OSGi
    ./1
      .Bundles
        ./1
          .BundleState
            .SymbolicName = org.osgi.service.log
          ./Resources
          ./Lifecycle
        ./BundleExt
        ./3
          .BundleState
            .SymbolicName = org.osgi.service.cm
          ./Resources
          ./Lifecycle
          ./BundleExt
```

[Case2] Filter including an interior node name as a key value and TargetSubtree including "*" as a wild-card

- TargetSubtree: $/OSGi/*/PackageState/
- Filter: (&(ExportingBundle=5)(ImportingBundle=10))
  - Packages that have a leaf node named "5" as a child node of ExportingBundle and have a leaf node named "10" among children nodes of ImportingBundle are matched. The filter search is performed for all OSGi Frameworks included in the management tree, because the <instance_id> of the OSGi Framework is represented as a wild-card. The Result node contains corresponding sub-trees that have absolute node paths descending from $ node. A possible sub-tree under the Result node is described below ($ node should be changed to the actual node path depending on each execution environment):
[Case3] Filter for service properties filtering

- TargetSubtree: $/OSGi/1/ServiceState/
- Filter: (@application=automation)
  - Services that have properties including "application" as a key and "automation" as a value are matched. The Result node contains corresponding sub-trees that have relative node paths descending from the <service_id> node. A possible sub-tree under the Result node is described below ($ node should be changed to the actual node path depending on each execution environment):

```
Result
  $.
  OSGi
  .1
  PackageState
  .27
  .Name
  .Version
  .RemovalPending
  .ExportingBundle
  .5
  .ImportingBundle
  .10
  .13
  PackageState
  .11
  .Name
  .Version
  .RemovalPending
  .ExportingBundle
  .5
  .ImportingBundle
  .10

Properties
.application
  .Type = java.lang.String
  .Cardinality = scalar
  .Values
    .1 = automation
      ...
  .RegisteringBundle
    ...
  .UsingBundle
    ...
```
[Case4] TargetSubtree including "-" as multiple level wild-card

- TargetSubtree: $/OSGi/-/BundleState/
- Filter: (BundleType=1)

  Bundles whose BundleType correspond to 1, which is the constant of BUNDLE_TYPE_FRAGMENT, are matched. The filter search is performed for all OSGi Frameworks included in the management tree, because the <instance_id> of the OSGi Framework is included in the wild-card. The Result node contains corresponding sub-trees that have absolute node paths descending from $ node. A possible sub-tree under the Result node is described below ($ node should be changed to the actual node path depending on each execution environment):

```
Result
  $  
  .OSGi
  .1  
    Bundles
    .12  
     .BundleState
      .BundleType = 1
      :  
      .Resources
      :  
      .Lifecycle
      :  
      .BundleExt
  .2  
    Bundles
    .15  
     .BundleState
      .BundleType = 1
      :  
      .Resources
      :  
      .Lifecycle
      :  
      .BundleExt
```
6.7 BundleResources Object

The BundleResources sub-tree is optional and it is used to derive resources in the bundle jar file. The BundleResources sub-tree consists of interior and leaf nodes that correspond to actual file-paths and files in the jar file respectively, so that a remote manager can derive a bundle's resources by a simple operation. For example, suppose there is /META-INF/Manifest.mf file in a jar file, 

$\text{OSGi/}<\text{instance_id}>/\text{BundleResources/}<\text{bundle_id}>/\text{META-INF/Manifest.mf}$

nodes are automatically created by the BundleResources Object, which implements the BundleResource sub-tree. Therefore, a remote manager simply gets the node value of $\text{OSGi/}<\text{instance_id}>/\text{BundleResources/}<\text{bundle_id}>/\text{META-INF/Manifest.mf}$.

The <directory> node is automatically created and represents a directory included in a jar file. The <bundle_id> node must indicate the root directory of the jar file. On the other hand, <file_name> node is also created automatically as a leaf node, but represents a file name in a jar file. The <file_name> node contains the content of the file as node value, which is encoded in base 64 format. If the size of the content exceeds a limit which depends on the BundleResources Object implementation, the BundleResources Object should abort the reading of file content and should throw a DmtException.

Because the BundleResources sub-tree is created automatically based on the actual file architecture of the indicated jar file, this sub-tree must be read-only. A remote manager cannot add, replace, delete and exec nodes in this sub-tree.

Note: Assume that adopted remote management protocol implements a method that retrieves data of all nodes under the specified node recursively (such as "GetParameterValues" RPC of TR-069 protocol). When a remote manager indicates $\text{OSGi/}<\text{instance_id}>/\text{BundleResources/}<\text{bundle_id}>$ node or ancestor nodes of it, heavy data transaction between the remote manager and client would occur because the BundleResources sub-tree contains all file of the bundle jar-file. Therefore, when using such kind of method in an operation, a remote manager should carefully specify the node in terms of performance.

[REMARK] A considered alternative of the BundleResources sub-tree architecture is shown in section 7.

All nodes for the Resources Object sub-tree are explained in Table 6.6.
Table 6.6 Resources sub-tree Nodes

<table>
<thead>
<tr>
<th>URI</th>
<th>Add</th>
<th>Get</th>
<th>Replace</th>
<th>Delete</th>
<th>Exec</th>
<th>Type</th>
<th>Cardinality</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BundleResources</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This is a parent node for resources included in the bundle. This sub-tree is detailed later.</td>
</tr>
<tr>
<td>BundleResources/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A node that represents a Bundles instance. This number must equal the bundle id, which Bundle#getBundleId() returns.</td>
</tr>
<tr>
<td>BundleResources/&lt;bundle_id&gt;/&lt;directory&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Represents a directory name. This node is automatically created to show a file-path in a bundle jar-file.</td>
</tr>
<tr>
<td>BundleResources/&lt;bundle_id&gt;/&lt;file_name&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The content of the file specified in the node path. This leaf node is automatically created for an individual file in a bundle jar-file.</td>
</tr>
</tbody>
</table>

6.8 Configuration Object

The Configuration Object manages the Configuration Admin service via the DMT Admin service. The features of this object are the same as those defined in the Mobile Management Tree except for the position of the top node: here the top node can be accessed by $/OSGi/<instance_id>/Configuration while the top node in Mobile Management Tree can be accessed at $/Configuration.

6.9 Policy Object

The Policy Object manages the Conditional Permission Admin service and the Permission Admin service via the DMT Admin service. The features of this object are the same as those defined in the Mobile Management Tree except for the position of the top node: here the top node can be accessed by $/OSGi/<instance_id>/Policy while the top node in the Mobile Management Tree can be accessed at $/Policy.

6.10 Log Object

The Log Object manages the Log service via the DMT Admin service. The features of this object are the same as those defined in the Mobile Management Tree except for the position of the top node: here the top node can be accessed by $/OSGi/<instance_id>/Log here while the top node in the Mobile Management Tree can be accessed at $/Log.
6.11 Limitations
The Residential Management Tree has several limitations due to the specification of a remote management protocol (TR-069) utilized with the DMT Admin service. The following section describes these limitations.

6.11.1 Maximum and Minimum values of numeric parameters
Due to the TR-069 specification, only the data types Integer or Unsigned Integer are available in the Residential Management Tree. Therefore, the maximum value of an unsigned integer should be less than 4294967295, and the minimum value should be greater than or equal to -2147483648. Consequently, the Bundle ID and the service.id which are originally a Long value in the OSGi specification should be represented as integer value in the Residential Management Tree.

6.11.2 Life-cycle control of the Framework
The life-cycle control of the OSGi Framework depends on the implementation of the framework on which the Residential Management Tree can run. This RFC doesn't specify the results of calling $/OSGi/<instance_id>/Framework/FrameworkLifecycle/*.

6.11.3 Service Properties Expression
The following list summarizes the restrictions and rules that should be obeyed in order to maintain a valid ServiceState Object.

- Complex data types, multiple-dimension arrays or vectors might be discarded from the Properties sub-tree, since these types of properties are difficult to be represented in the object tree if the data isn't serializable.
7 Considered Alternatives

7.1 BundleResources sub-tree architecture

There is considerable alternative for designing the BundleResources sub-tree to retrieve resources inside the bundle jar file. The following architecture is another design of the BundleResources sub-tree, which was eventually discarded as the result of discussion.

The `<resource_id>` node is a dynamic node, which means the path name of the new node is automatically assigned as an instance number by the Bundles Object. In advance, the management system needs to create the `<resource_id>` node. Then it has to indicate the file path by setting the Path node parameters and can retrieve the contents of the specified file as the value of the Content node.

This architecture has pros and cons compared to the proposed architecture in Section 6.8.

First, this architecture enables a scalable implementation because the remote manager is able to decide which resources should be retrieved from bundles arbitrarily. The remote manager, therefore, can avoid heavy data traffic between remote manager and client, when an ancestor node of the Resources sub-tree is specified by the GetParameterValues RPC defined in the TR-069.

Secondly, the typical usage of the Resources sub-tree would be for diagnostics scenarios; when some problems occur, a remote operator who needs to find the cause of the problem retrieves the content of the bundle JAR file by using this sub-tree.

But this architecture limits the diagnostics ability of a remote manager. The remote manager has to check many files by repetitively creating `<resources_id>` nodes until the cause of problem is detected. This situation prevents an effective diagnostics through the Resources sub-tree.

On the other hand, the proposed architecture in Section 6.8 provides a better diagnostics ability than this but the performance decreases in terms of data transactions when retrieving data recursively.
Consequently, the architecture proposed in Section 6.8 is chosen for the Resources sub-tree due to the effectiveness of the diagnostics ability, even though there is a risk of a performance drawback when a remote manager indicates $/OSGi/<instance_id>/BundleResources/<bundle_id>$ node or ancestor nodes.

8 Security Considerations

All security requirements follow the DMT Admin specification.

9 Document Support

9.1 References


[3]. OMA, Open Mobile Alliance. The mission of the Open Mobile Alliance is to facilitate global user adoption of mobile data services by specifying market driven mobile service enablers that ensure service interoperability across devices, geographies, service providers, operators, and networks, while allowing businesses to compete through innovation and differentiation. http://www.openmobilealliance.org/

[4]. OMA Device Management specification v1.2. The goal of the Device Management Working Group is to specify protocols and mechanisms that achieve management of mobile devices including the necessary configuration to access services and management of the software on mobile devices. http://www.openmobilealliance.org/release_program/dm_v1_2C.html


[6]. The Broadband Forum is a global consortium of nearly 200 leading industry players covering telecommunications, equipment, computing, networking and service provider companies. Established in 1994, originally as the ADSL Forum and later the DSL Forum, the Broadband Forum continues its drive for a global mass market for broadband, to deliver the benefits of this technology to end users around the world over existing copper telephone wire infrastructures. http://www.broadband-forum.org/about/forumhistory.php

[7]. Data model template for TR-069 enabled devices, TR-106 amendment 1, November 2006
9.2 Author's Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Koya Mori</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81-46-859-3446</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Ikuo Yamasaki</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81-46-859-8537</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:yamasaki.ikuo@lab.ntt.co.jp">yamasaki.ikuo@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Shigekuni Kondo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81-46-859-3444</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

9.3 Acronyms and Abbreviations

9.4 End of Document
Different industries are interested in applying OSGi to advance their businesses, in which remote management is a key issue. In the residential area, the TR-069 is the one of the de-facto standard protocol for remote management. The best way to realize remote management based on the TR-069 on OSGi is utilizing DMT Admin service, which has been defined in the OSGi Alliance for the mobile device management. In this case, TR-069 is implemented as a protocol adapter of the DMT Admin. The DMT Admin service, however, has the interfaces inspired by OMA-DM, which is the de-facto standard protocol for mobile area. Although these protocols have the similar objectives and functionality, there are several architectural differences especially on a notification mechanism. Therefore, the DMT Admin service should be extended to accommodate various protocols such as TR-069 to manage OSGi framework remotely. This RFC defines a new node property and methods manipulating it to handle notification mechanism with the DMT Admin service.
# 1 Document Information

## 1.1 Table of Contents

1 Document Information ................................................................................................................. 2
   1.1 Table of Contents ................................................................................................................ 2
   1.2 Terminology and Document Conventions ............................................................................ 3
   1.3 Revision History .................................................................................................................. 3

2 Introduction .................................................................................................................................. 3

3 Application Domain ..................................................................................................................... 4

4 Problem Description .................................................................................................................... 5
   4.1 Management agent making use of OSGi standardized Java interfaces ......................... 5
   4.2 Mobile specification approach ............................................................................................. 5
   4.2.1 Support for TR-069 notifications ....................................................................................... 6
   4.2.2 Limitations in the number of services available in the DMT .............................................. 7
   4.2.3 Mapping TR-069 to the OMA-DM inspired DMT model .................................................... 7
   4.3 Conclusion .......................................................................................................................... 8

5 Requirements ............................................................................................................................... 8

6 Technical Solution ....................................................................................................................... 9
   6.1 Extension of notification mechanism ................................................................................... 9
   6.1.1 Node property ................................................................................................................... 9
   6.1.2 Notification configuration methods ................................................................................... 9
   6.2 Addition of data types .......................................................................................................... 10
   6.3 Limitation of notification ....................................................................................................... 10
   6.4 Javadoc ................................................................................................................................... 10

7 Considered Alternatives .............................................................................................................. 18

8 Security Considerations .............................................................................................................. 19

9 Document Support ....................................................................................................................... 19
   9.1 References .......................................................................................................................... 19
   9.2 Author's Address .................................................................................................................. 20
   9.3 Acronyms and Abbreviations ............................................................................................... 20
   9.4 End of Document .................................................................................................................. 21
1.2 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 9.1.

Source code is shown in this typeface.

1.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Sep. 16 2008</td>
<td>Initial Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Koya Mori, NTT Corporation, <a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>2nd</td>
<td>Jan. 20 2009</td>
<td>2nd Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generic setter/getter methods are added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limitation of notification is described.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Koya Mori, NTT Corporation, <a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td>1.0.0</td>
<td>Oct. 16 2009</td>
<td>Public Draft</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Javadoc and Considered Alternatives are added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Notification property definition is changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Koya Mori, NTT Corporation, <a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ikuo Yamasaki, NTT Corporation, <a href="mailto:yamasaki.ikuo@lab.ntt.co.jp">yamasaki.ikuo@lab.ntt.co.jp</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shigekuni Kondo, NTT Corporation, <a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

2 Introduction

Traditionally, fixed telecommunication operators don’t have knowledge about what runs in the customer’s local area network (LAN). They provide connectivity and manage the wide area network (WAN) that provides this connectivity, but they do not know anything about the devices and networks behind the gateway (xDSL mainly) that interconnect WAN and LAN. Recently the need for management of customer networks and devices is increasing in order to make the deployment of new complex services at home (home automation, tele-health,
VoIP, IPTV, surveillance, etc) feasible with reasonable costs. For example to avoid sending technicians to the customer premises for solving problems.

There are two main kinds of devices that need to be managed in operator’s business: those which come from the fixed business managed through TR-069 and standardized by the Broadband FORUM, and those which come from the mobile business, managed through OMA-DM and standardized by the Open Mobile Alliance. The DMT Admin specification of OSGi covers the OMA-DM ones. In addition, the design of DMT Admin Service specification potentially allows adoption of other remote management protocols other than the OMA-DM.

The best way to realize remote management based on TR-069 on OSGi is utilizing the DMT Admin service. In this case, TR-069 is implemented as a protocol adapter of the DMT Admin. However, there are several architectural differences between the TR-069 and the OMA-DM, although these two protocols have the similar objectives and functionality.

### 3 Application Domain

Driven by triple play service delivery in the home network, fixed line access service providers have the need to configure home devices to ensure the proper service delivery. Broadband Forum’s CPE WAN Management Protocol (CWMP, alias TR-069) enables them to do this. By using a remote management server (Auto Configuration Server, ACS), they are able to manage TR-069 enabled devices. TR-069 provides them with possibilities to configure parameters, be informed of parameter changes (notification), start diagnostic tools, update firmware, etc.

Similarly, for the mobile world, the OMA defined the OMA-Device Management specification for remote management of mobile devices. OMA-DM offers similar tools to the mobile service providers as TR-069 to fixed line service provider, but OMA-DM is of course tailored to the specifics of the mobile environment.

As OSGi technology offers a flexible software environment for all these different devices, the remote management of the platform is of interest for both fixed and mobile service providers. As such, it should be possible to integrate the remote management of the OSGi platform, and the applications running on top of it, in the existing management infrastructure.

The DMT Admin service with its mobile management tree in the Mobile specification for OSGi R4 standardizes the remote management of an OSGi platform. As it is largely inspired by OMA-DM, it needs to be evaluated for multi protocol support.
4 Problem Description

In a scenario in which service providers offer a growing number of services, to use specific solutions for the management of those services is not the most suitable option. To speed up the deployment of these services, such as triple play, home automation or tele-health, it is essential to offer general management solutions that allow for the management of a large number of services and the flexible life-cycle management of applications.

These devices usually are already managed by a standard protocol, so it makes sense that an OSGi framework, which hosts the services, running on a device could be managed in the same way as the other resources of the device. Of course, the remote management should be fully integrated in the existing remote management solutions of the service provider to avoid duplicating management infrastructure and to increase performance on the devices.

Currently, there are two options in OSGi for remote management:

- create a management agent bundle making use of the Java object interfaces,
- create a protocol adapter bundle that interacts with the DMT Admin service, as defined in the OSGi Mobile specification.

4.1 Management agent making use of OSGi standardized Java interfaces

Currently, for the management of a bundle, the OSGi specifications define different Java objects with which a management application can interact. Using this approach, a management agent can implement extensive management of the OSGi framework, as well as any service standardized. Mapping the Java interfaces to the specific remote management protocol and data model tree is up to the management agent.

For runtime interaction with a bundle, a bundle can register a service in the service registry. However, this service interface is not standardized. Also, mapping the service interface to a general management model is not standardized. A current approach is to implement a proprietary service interface on all bundles to be managed. By tailoring this interface so that it easily maps to the management protocol primitives, it is simple for the management agent to map remote management commands to the bundle's service interface. The disadvantage is the proprietary service interface, so that 3rd party bundles might not be compliant.

As a conclusion we can say that this current approach allows for extensive remote management of any aspect of the OSGi platform, but lacks a standardized service interface definition for bundles to implement.

4.2 Mobile specification approach

The Mobile Expert Group has provided its own solution based on the OMA [3] Device Management [4] specification to provide a remote management solution. The OSGi Mobile specification contains two chapters related to remote management:

- chapter 3: detailing the mobile management tree
- chapter 117: detailing: the DMT Admin service, bundle plugin interface specification, the notification service
The Device Management Tree model of OMA-DM was chosen as meta-data model and operational model. However, it was intended to be mappable to other protocols.

An analysis of mapping the Mobile specification DMT model to TR-069, however, shows that the current DMT model approach (as defined in the OSGi R4 Mobile Specification) introduces some issues. For example:

- Limitations for active or passive notifications on any parameter in the object tree
- A limited number of services have been mapped to the DMT model
- The complexity of mapping a new protocol to the OMA-inspired DMT model, which could imply performance issues on limited devices.

### 4.2.1 Support for TR-069 notifications

TR-069 offers the feature of active and passive notifications. By setting a parameter’s notification attribute, a remote manager requests to be notified with the parameter’s new value at the time the value changes (active notification) or at the next periodic inform (passive notification). Notification can be configured on any parameter of the TR-069 object tree. This approach enables the remote manager to be informed not only of changes in status variables of the platform, but also of configuration changes performed by a local manager, e.g. through a local Web interface.

The Mobile specification offers a few features that could help to implement TR-069 notification support:

- The DMT Admin service sends events using the Event Admin service when operations have been performed on nodes (nodes added, removed or copied; node values changed etc.)
- The OSGi Notification service defines a way to alert a remote management server. Protocol adaptors on their turn have to implement a RemoteAlertSender interface (and register it) for use by the notification service. Notifications are sent by calling sendNotification on the notification service:
- The Monitor Admin service: A bundle can register a Monitorable service, to be used by the Monitor Admin service. By registering a Monitorable service, the bundle exposes access to a number of status variables. Notification can be implemented by the StatusVariable provider. If it does, it will call the update method on the Monitor Listener. The Monitor Admin service then generates an event on the Event Admin service. The Monitor service is currently also represented in the DMT tree.

Two problems arise when trying to map the current approach to TR-069:

- TR-069 defines that notification is applicable to any parameter in the object tree.

Currently, the DMT Admin service only send events for operations on DMT nodes that were performed using the DMT Admin API. For example: if configuration changes are performed by using the Configuration Admin service API, no events will be sent. Most of the current implementations do not perform all changes via the DMT Admin service. Therefore, the events sent by the DMT Admin service are an only subset and thus not very reliable as single source of events (and thus as single source of TR-069 notifications).

The OSGi Monitor service only supports notification of changes on Status Variables, exposed through a Monitorable service, and enabled by the bundle to support on-change notification (i.e. dynamic Status Variables).
Requesting notification is not fully under the control of the remote manager. In the case of a bundle using the notification service, there is no standardized way to configure the bundle to send alerts when the value of one of the implemented DMT nodes changes. In the case of the monitor service, the sending of events can be controlled, but is limited to dynamic Status Variables.

The current DMT Admin service has no attributes properties on its nodes to be used to configure notification behavior, such as active notification and passive notification defined in TR-069. Therefore, a remote manager cannot control the notification behavior of DMT nodes in a standardized way.

To conclude, the current options, as provided in the Mobile specification, limit notification of parameter changes to StatusVariables, explicitly enabled for monitoring. There is no standardized approach available to monitor changes on any node in the DMT.

### 4.2.2 Limitations in the number of services available in the DMT

The OSGi R4 Mobile specification mapped a number of services to the DMT tree. However, these services are limited to the services listed in the Mobile Specification. Other interesting services, as listed in the OSGi R4 Service Compendium are not yet mapped (standardized) in the DMT tree:

The DMT tree defined in the Mobile Specification contains objects for the following services:

- Configuration Admin service
- Log service
- Monitor Admin service
- Application Admin service
- Conditional Permission Admin service
- Deployment service

A number of areas that could be of interest to a remote manager are currently missing in the DMT tree:

- Startlevel management
- Bundle management: managing individual bundles as opposed to deployment packages (inventory, life-cycle management, exported services, …)
- Service management: getting a remote view on services registered in the service registry
- Permission Admin management
- Home Gateway Core Function management: handling Home Gateway core functions, such as firewall configuration and port forwarding control, from bundles running on an OSGi framework

### 4.2.3 Mapping TR-069 to the OMA-DM inspired DMT model

Within the OSGi Mobile specification, the choice has been made to model the DMT after OMA-DM.
As a result, creating an OMA-DM protocol adapter is quite straightforward. Although no major hurdles have been identified in creating a TR-069 protocol adapter, it is less straightforward:

- The TR-069 RPC primitives have to be translated to the DMT Admin service interface methods (which are OMA-DM RPC inspired).
- The TR-069 tree has to be mapped to the DMT tree. Translating object model specific features like DMT meta nodes, or TR-069 attributes is not straightforward. It might require specific extensions to the DMT, e.g. to support TR-069 attributes, or a single node in the DMT might result in multiple objects in a TR-069 data model, etc.
- The TR-069 data types have to be mapped to the DMT Admin data types. However, TR-069 data types, such as “unsignedint” and “dateTime” (ISO 8601), cannot be translated appropriately into DMT Admin data types defined in the current specification. Translating these data types might result a limitation of the available value range and a complex object that consists of multiple nodes, respectively.

### 4.3 Conclusion

The OSGi Mobile specification delivers a standardized data model (the DMT), and standardized interface (on the DMT Admin service) to enable remote management through a protocol adapter. However, the current specification lacks management objects for a number of interesting areas. Also, there is some support lacking for TR-069 notifications. Furthermore, since the DMT model is OMA-DM inspired, implementing a TR-069 protocol adapter is not straightforward, although not impossible.

### 5 Requirements

REQUIREMENT[5]: Support for notification of parameter value changes is required for both framework and services sub-trees, as well as for bundles implementing a sub-tree of the management tree. Some lightweight mechanism MUST allow identifying which parameters have changed and haven’t already been notified.

REQUIREMENT[6]: The remote management solution MUST provide some mechanism to notify the remote management entities, where the remote manager controls which information to receive.

REQUIREMENT[9]: The solution should be able to handle the data types of “unsignedint” and “dateTime” (ISO 8601) to accommodate TR-069 protocol.
6 Technical Solution

This RFC describes how DMT Admin service is extended to accommodate the TR-069 protocol. Since DMT Admin service intends to use the OMA-DM protocol as a remote management protocol, some key features such as notification mechanism and data types are missing which makes it difficult to support the TR-069 protocol.

This RFC describes the following changes to DMT Admin service.

- Extension of notification mechanism of DMT Admin service
- Addition of new data types to DMT Admin service

Only the above changes are necessary to permit DMT Admin service to support the TR-069 protocol, there is no need to modify the basic architecture.

6.1 Extension of notification mechanism

According to the TR-069 specification, all nodes in a management tree have to be configurable regardless of whether they issue notifications to the remote management server or should not. However, the definition of nodes made by the DMT Admin specification does not support notification management, and the DMT Admin service does not provide methods for manipulating those properties.

6.1.1 Node property

The new node property "Notification" must be added to the node properties definition of the DMT Admin service specification. This property is used to configure the notification behavior of the node.

- Notification – (String) The behavior of notification for the node. This property indicates the types of notification, which is issued to the remote manager to notify the change of a node's value. This property is optional depending on the node's implementation.

Because the behavior of notification depends on the underlying protocol, this RFC does not define any meaning to any value of this property except "0_Notification_Off". The value of "0 Notification Off" is defined as meaning that notification is off, and this definition must be commonly applied for all protocols. In other words, if the "0_Notification_Off" is specified as a "Notification" property of a node, a data plugin must not issue notification for the node value change. "0_Notification_Off" should be a default value of this property. The default value is applied when the data plugin does not have an initial notification attribute specified by the data model implemented, and no other notification attribute has been set through ReadWriteDataSession.setNodeNotification(String[], String).

Given that multiple protocols may be used in the same system, multiple definitions, each of which is associated with a different protocol, can be concurrently used for the notification property for different nodes. Therefore, the definition of the Notification value is recommended to be defined as protocol identifiable, because the data plugin needs to check whether it can handle the specified notification attribute or not.

6.1.2 Notification configuration methods

The Notification property described above should be configured by a protocol adapter through DMT Admin service. Therefore, the interfaces defined in DMT Admin service must be extended to provide new methods for
the configuration of Notification property on a node. Details of these methods are described in the Javadoc section. See also Section 6.4 Javadoc.

Following methods must be added to info.dmtree.DmtSession interface:

- public String getNodeNotification(String nodeUri) throws DmtException
  Get the Notification property of the indicated node.

- public void setNodeNotification(String nodeUri, String notification) throws DmtException
  Set the Notification property state of the indicated node. Definition of notification depends on the underlying protocol used with DMT Admin service.

Following method should be added to info.dmtree.spi.ReadableDataSession interface:

- public String getNodeNotification(String[] nodePath) throws DmtException
  Get the Notification property of a node. There might be no Notification property set for a node.

Following method should be added to info.dmtree.spi.ReadWriteDataSession interface:

- public void setNodeNotification(String[] nodePath, String notification) throws DmtException
  Set the Notification property of the indicated node.

### 6.2 Addition of data types

Two data types are missing in the DMT Admin specification and are needed to accommodate the TR-069 protocol. The following data types should be added to the format of the DmtData class.

- FORMAT_LONG – A long value. Since there is no suitable data type for unsignedInt in Java, this data type should be mapped as unsigned integer for use by the TR-069 protocol.

- FORMAT_DATETIME – A String object that is interpreted as the dateTime type defined in ISO 8601; it is used as the value of date and time in TR-069.

### 6.3 Limitation of notification

DMT Admin extension has a limitation on notification, which depends on the OSGi Framework and services. The several APIs of OSGi Framework and standard services do not support a notifying or eventing mechanism for value/state changing. For example, permission changes can not be notified to a bundle running on a framework. That is, nodes that have this characteristic cannot implement the notification mechanism even if their notification property is set to issue notifications. This limitation depends on features specified in the OSGi specification.

### 6.4 Javadoc

Several DMT Admin interfaces need to be modified to support the notification capability. The javadoc should be considered as a binding part of this specification. Note that only APIs that have been modified and added are included in this document.
6.5 info.dmtree

Class DmtData

java.lang.Object

info.dmtree.DmtData

public final class DmtData extends java.lang.Object

Field Summary

| static int FORMAT_DATETIME | The node holds a String object that is interpreted as a dateTime type defined in ISO 8601. |
| static int FORMAT_LONG   | The node holds a long value. |

Constructor Summary

DmtData(long ln)
Create a DmtData instance of long format and set its value.

DmtData(java.lang.String value, int format)
Create a DmtData instance of the specified format and set its value based on the given string.

Method Summary

| java.lang.String | getDateTime() | Gets the value of a node with dateTime format. |
| long             | getLong()     | Gets the value of a node with long format. |
| java.lang.String | toString()    | Gets the string representation of the DmtData. |

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Field Detail
6.5.1 FORMAT_LONG

public static final int FORMAT_LONG

The node holds a long value. This data type should be mapped as unsigned integer if needed.

See Also:
Constant Field Values

6.5.2 FORMAT_DATETIME

public static final int FORMAT_DATETIME

The node holds a String object that is interpreted as a dateTime type defined in ISO 8601.

See Also:
Constant Field Values

Constructor Detail

6.5.3 DmtData

public DmtData(java.lang.String value, int format)

Create a DmtData instance of the specified format and set its value based on the given string. Only the following string-based formats can be created using this constructor:

- FORMAT_STRING - value can be any string
- FORMAT_XML - value must contain an XML fragment (the validity is not checked by this constructor)
- FORMAT_DATE - value must be parseable to an ISO 8601 calendar date in complete representation, basic format (pattern CCYYMMDD)
- FORMAT_TIME - value must be parseable to an ISO 8601 time of day in either local time, complete representation, basic format (pattern hhmmss) or Coordinated Universal Time, basic format (pattern hhmmssZ)
- FORMAT_DATETIME - value must be parseable to an ISO 8601 calendar date-time in complete representation, basic format (pattern ccyy-mm-ddThh:mm:ssZ)

The null string argument is only valid if the format is string or XML.

Parameters:
value - the string, XML, date, time or dateTime value to set
format - the format of the DmtData instance to be created, must be one of the formats specified above

Throws:
java.lang.IllegalArgumentException - if format is not one of the allowed formats, or value is not a valid string for the given format
java.lang.NullPointerException - if a date or time is constructed and value is null

6.5.4 DmtData

public DmtData(long ln)

Create a DmtData instance of long format and set its value.

Parameters:
ln - the long value to set

Method Detail

6.5.5 getDateTime

public java.lang.String getDateTime()

Gets the value of a node with dateTime format. The returned dateTime string is formatted according to the ISO 8601 definition of a calendar date-time in complete representation, basic format (pattern ccyy-mm-ddThh:mm:ssZ).

Returns:
the dateTime value

Throws:
DmtIllegalStateException - if the format of the node is not dateTime

6.5.6 getLong

public long getLong()

Gets the value of a node with long format.

Returns:
the long value

Throws:
DmtIllegalStateException - if the format of the node is not long

6.5.7 toString

public java.lang.String toString()

Gets the string representation of the DmtData. This method works for all formats.
For string format data - including `FORMAT_RAW_STRING` - the string value itself is returned, while for XML, date, time, dateTime, integer, long, float, boolean and node formats the string form of the value is returned. Binary - including `FORMAT_RAW_BINARY` - and base64 data is represented by two-digit hexadecimal numbers for each byte separated by spaces. The NULL_VALUE data has the string form of "null". Data of string or XML format containing the Java `null` value is represented by an empty string.

Overrides:

`toString` in class `java.lang.Object`

Returns:

the string representation of this `DmtData` instance

### 6.6 info.dmtree

### Interface DmtSession

```java
public interface DmtSession
```

#### Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static java.lang.String</code></td>
<td><code>NOTIFICATION_OFF</code></td>
<td>The notification is off, and a data plugin must not issue a notification for the node value change.</td>
</tr>
</tbody>
</table>

#### Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getNodeNotification(String nodeUri)</code></td>
<td>Get the notification attribute of a node.</td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setNodeNotification(String nodeUri, String notification)</code></td>
<td>Set the notification property of a node.</td>
</tr>
</tbody>
</table>

#### Field Detail

### 6.6.1 NOTIFICATION_OFF

```java
static final java.lang.String NOTIFICATION_OFF
```

The notification is off, and a data plugin must not issue a notification for the node value change.
Method Detail

6.6.2 getNodeNotification

```java
String getNodeNotification(String nodeUri)
throws DmtException
```

Get the notification attribute of a node. There might be no notification property set for a node.

**Parameters:**
- `nodeUri` - the URI of the node

**Returns:**
- the notification attribute of the node, or null if the node has no notification attribute

**Throws:**
- `DmtException` - with the following possible error codes:
  - `URI_TOO_LONG` if `nodeUri` or a segment of it is too long, or if it has too many segments
  - `INVALID_URI` if `nodeUri` is null or syntactically invalid
  - `NODE_NOT_FOUND` if `nodeUri` points to a non-existing node
  - `PERMISSION_DENIED` if the session is associated with a principal and the ACL of the node does not allow the Get operation for the associated principal
  - `METADATA_MISMATCH` if node information cannot be retrieved according to the meta-data (it does not have `MetaNode.CMD_GET` access type)
  - `FEATURE_NOT_SUPPORTED` if the Notification property is not supported by the DmtAdmin implementation or the underlying plugin
  - `DATA_STORE_FAILURE` if an error occurred while accessing the data store
  - `COMMAND_FAILED` if the URI is not within the current session's subtree, or if some unspecified error is encountered while attempting to complete the command

- `DmtIllegalStateException` - if the session is already closed or invalidated

- `java.lang.SecurityException` - if the caller does not have the necessary permissions to execute the underlying management operation, or, in case of local sessions, if the caller does not have `DmtPermission` for the node with the Get action present

6.6.3 setNodeNotification

```java
void setNodeNotification(String nodeUri, String notification)
throws DmtException
```

Set the notification property of a node.

**Parameters:**
- `nodeUri` - the URI of the node
- `notification` - the notification attribute of the node

**Throws:**
- `DmtException` - with the following possible error codes:
  - `URI_TOO_LONG` if `nodeUri` or a segment of it is too long, or if it has too many segments
  - `INVALID_URI` if `nodeUri` is null or syntactically invalid
• **NODE NOT_FOUND** if `nodeUri` points to a non-existing node
• **PERMISSION_DENIED** if the session is associated with a principal and the ACL of the node does not allow the `Get` operation for the associated principal
• **COMMAND NOT ALLOWED** in non-atomic sessions if the underlying plugin is read-only or does not support non-atomic writing
• **METADATA_MISMATCH** if node information cannot be retrieved according to the meta-data (it does not have `MetaNode.CMD_GET` access type)
• **FEATURE_NOT_SUPPORTED** if the Notification property is not supported by the DmtAdmin implementation or the underlying plugin
• **TRANSACTION_ERROR** in an atomic session if the underlying plugin is read-only or does not support atomic writing
• **DATA_STORE_FAILURE** if an error occurred while accessing the data store
• **COMMAND_FAILED** if the URI is not within the current session's subtree, or if some unspecified error is encountered while attempting to complete the command

DmtIllegalStateException - if the session is already closed or invalidated
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation, or, in case of local sessions, if the caller does not have DmtPermission for the node with the Get action present

### 6.7 info.dmtree.spi

#### Interface ReadableDataSession

All Known Subinterfaces:
- ReadWriteDataSession, TransactionalDataSession

```java
public interface ReadableDataSession
```

### Method Summary

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getNodeNotification(java.lang.String[] nodePath)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the notification attribute of a node.</td>
<td></td>
</tr>
</tbody>
</table>

### Method Detail

#### 6.7.1 getNodeNotification

```java
java.lang.String getNodeNotification(java.lang.String[] nodePath)
```

throws DmtException

Get the notification attribute of a node. There might be no notification property set for a node.

**Parameters:**
- `nodePath` - the absolute path of the node
Returns:
the notification attribute of the node

Throws:
DmtException - with the following possible error codes:
• NODE_NOT_FOUND if nodePath points to a non-existing node
• METADATA_MISMATCH if the information could not be retrieved because of meta-data restrictions
• FEATURE_NOT_SUPPORTED if the Notification property is not supported by the DmtAdmin implementation or the underlying plugin
• DATA_STORE_FAILURE if an error occurred while accessing the data store
• COMMAND_FAILED if some unspecified error is encountered while attempting to complete the command

java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

6.8 info.dmtree.spi
Interface ReadWriteDataSession
All Superinterfaces:
ReadableDataSession

All Known Subinterfaces:
TransactionalDataSession

public interface ReadWriteDataSession extends ReadableDataSession

Method Summary
void setNodeNotification(java.lang.String[] nodePath, java.lang.String notification)
Set the notification property of a node.

Method Detail
6.8.1 setNodeNotification
void setNodeNotification(java.lang.String[] nodePath, java.lang.String notification)
throws DmtException
Set the notification property of a node.

Parameters:
nodePath - the absolute path of the node
notification - the notification attribute of the node

Throws:
DmtException - with the following possible error codes:

- **NODE_NOT_FOUND** if nodePath points to a non-existing node
- **META_DATA_MISMATCH** if the information could not be retrieved because of meta-data restrictions
- **FEATURE_NOT_SUPPORTED** if the Notification property is not supported by the DmtAdmin implementation or the underlying plugin
- **DATA_STORE_FAILURE** if an error occurred while accessing the data store
- **COMMAND_FAILED** if some unspecified error is encountered while attempting to complete the command

Java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

See Also:

DmtSession.setNodeNotification(String, String)

---

### 6.9 Constant Field Values

#### 6.9.1 Contents

- **info.dmtree.*

<table>
<thead>
<tr>
<th>info.dmtree.DmtData</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final int FORMAT_DATETIME 16384</td>
</tr>
<tr>
<td>public static final int FORMAT_LONG 8192</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>info.dmtree.DmtSession</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final java.lang.String NOTIFICATION_OFF &quot;0_Notification_Off&quot;</td>
</tr>
</tbody>
</table>

---

### 7 Considered Alternatives

In terms of the differences in notification behavior between protocols, one alternative is to implement the notification mechanism inside a protocol adapter, which implements the protocol-specific logic. Data plugins should issue notification events whenever the nodes' values change internally; this means DMT Admin does not need to be extended to take account of notification attribute handling.

However, there are two main problems with this alternative.

1. If the node value is changed frequently, such as occurs with packet statistics, a huge number of events will be issued through the Event Admin. This will degrade the performance of the system.

2. If the data-model implemented by a data plugin contains pre-defined notification attributes, this alternative fails because the protocol adapter has no knowledge of the data-model.
Given the above consideration of these issues, the basic concept of the notification mechanism should fulfill the following requirements.

- The notification attributes should be kept and treated by data plugins. The data plugins should decide whether a node value change needs to fire a notification and should reduce the number of firing a notification.

- The notification mechanism should be able to handle pre-defined notification attributes in data-model. Such attributes, which are not specified by the remote manager via the protocol adapter, should be configured by data plugins.

The technical solution described in this RFC is considered to be the suitable approach to implementing the notification mechanism.

8 Security Considerations

All security requirements follow the DMT Admin specification.

9 Document Support

9.1 References


[3]. OMA, Open Mobile Alliance. The mission of the Open Mobile Alliance is to facilitate global user adoption of mobile data services by specifying market driven mobile service enablers that ensure service interoperability across devices, geographies, service providers, operators, and networks, while allowing businesses to compete through innovation and differentiation. http://www.openmobilealliance.org/

[4]. OMA Device Management specification v1.2. The goal of the Device Management Working Group is to specify protocols and mechanisms that achieve management of mobile devices including the necessary configuration to access services and management of the software on mobile devices. http://www.openmobilealliance.org/release_program/dm_v1_2C.html

[6]. The Broadband Forum is a global consortium of nearly 200 leading industry players covering telecommunications, equipment, computing, networking and service provider companies. Established in 1994, originally as the ADSL Forum and later the DSL Forum, the Broadband Forum continues its drive for a global mass market for broadband, to deliver the benefits of this technology to end users around the world over existing copper telephone wire infrastructures. [http://www.broadband-forum.org/about/forumhistory.php](http://www.broadband-forum.org/about/forumhistory.php)

[7]. Data model template for TR-069 enabled devices, TR-106 amendment 1, November 2006

### 9.2 Author's Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Koya Mori</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81-46-859-3446</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Ikuo Yamasaki</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81-46-859-8537</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:yamasaki.ikuo@lab.ntt.co.jp">yamasaki.ikuo@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Shigekuni Kondo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81-46-859-3444</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:kondo.shigekuni@lab.ntt.co.jp">kondo.shigekuni@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

### 9.3 Acronyms and Abbreviations
9.4 End of Document
This RFC specifies the architecture and mapping rules for an OSGi based admin service to access and manage the various aspects and underlying services of the Home Gateway Device. An implementation of the admin service enables OSGi services running on an Internet Gateway Device to access the core level functions of a home gateway. An important part of this RFC is the specification of mapping rules between the Broadband Forum’s TR documents, such as TR-098, and the DMT Admin tree.
## 0 Document Information

### 0.1 Table of Contents

- **0 Document Information** ................................................................................................................. 2
  - 0.1 Table of Contents ................................................................................................................ 2
  - 0.2 Terminology and Document Conventions ............................................................................... 3
  - 0.3 Revision History ................................................................................................................... 3
- **1 Introduction** ................................................................................................................................. 4
- **2 Application Domain** .................................................................................................................... 5
- **3 Problem Description** .................................................................................................................... 5
- **4 Requirements** ............................................................................................................................... 7
  - 4.1 Functional Requirements .................................................................................................... 7
  - 4.2 Non-Functional Requirements ............................................................................................. 7
- **5 Technical Solution** ....................................................................................................................... 8
  - 5.1 Scope .................................................................................................................................. 8
  - 5.2 General Considerations ......................................................................................................... 8
  - 5.3 Roles and Functional Blocks ............................................................................................... 10
  - 5.1 DMT Subtree Admin ............................................................................................................ 11
    - 5.1.1 DMT Admin restrictions .............................................................................................. 11
    - 5.1.2 Managing and Mapping DMT Admin subtrees ........................................................... 12
    - 5.1.3 Mapping Rules Examples ........................................................................................... 15
    - 5.1.4 Additional expected behaviour when using DMT Subtree Admin .............................. 16
    - 5.1.5 Raising Events ............................................................................................................ 17
    - 5.1.6 Event Mapping Examples ........................................................................................... 19
  - 5.2 Mapping for an Internet Gateway Device ........................................................................... 19
    - 5.2.1 Registering IGD related Data Plugins ........................................................................ 19
    - 5.2.2 Remote Management Events and Notifications ......................................................... 20
  - 5.3 Mapping rules to DMT Admin Management Tree .......................................................... 20
    - 5.3.1 Mapping from Broadband Forum TR-069 and TR-098 to DMT .................................. 20
    - 5.3.2 Mapping from UPnP IGD 1.0 ...................................................................................... 21
    - 5.3.3 Examples .................................................................................................................... 22
- **6 Considered Alternatives** .............................................................................................................. 22
  - 6.1 Not using DMT Admin ......................................................................................................... 22
- **7 Security Considerations** .............................................................................................................. 22
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 8.1.

Source code is shown in this typeface.

0.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial 0.5</td>
<td>Mai 29th 2009</td>
<td>Initial version for review. Andreas Kraft</td>
</tr>
<tr>
<td>0.6</td>
<td>June 3rd 2009</td>
<td>Included changes according to the audio conference on June 2nd 2009 and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the first review of the document. Andreas Kraft</td>
</tr>
<tr>
<td>0.7</td>
<td>June 4th 2009</td>
<td>Included changes according to comments in the document and e-mails.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified session handling. Clarified event handling. Changed security</td>
</tr>
<tr>
<td></td>
<td></td>
<td>considerations. Andreas Kraft</td>
</tr>
<tr>
<td>0.9</td>
<td>June 5th 2009</td>
<td>Updated illustration 5. Fixed typos. Rephrased description of the use of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session.id property. Andreas Kraft</td>
</tr>
<tr>
<td>0.10</td>
<td>October 10th 2009</td>
<td>Updated RFC accordingly to REG discussions and experiences gained</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with the reference implementation. Use the term &quot;IGD&quot; consistently in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the document. Moved the description for the DMT Subtree Admin in a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>separate section and made it more generic. Renamed properties to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reflect their plural nature. Added dmtAction property. Added an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>illustration for overlapping subtrees.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Andreas Kraft</td>
</tr>
</tbody>
</table>
1 Introduction

This RFC specifies the general architecture for an OSGi admin service to access and manage the various aspects and underlying services of the Internet Gateway Device (IGD). This service enables OSGi services running on an IGD to access the core level functions of an IGD, for example NAT and the firewall. There are various benefits which would be gained from such a service:

- **Increase compatibility**
  Make a retail home gateway compatible for a variety of operators to support full retail business; install operator-specific software on a retail home gateway (e.g. Web-based Management UI, VoIP Termination (B2BUA), VoD Termination (RTSP)).

- **Accelerate Differentiation**
  Install standards based as well as proprietary applications on an IGD (e.g. TR-069 remote management agents, UPnP IGD service, TR-064 LAN-side CPE management).

- **Leverage WAN Services**
  Connect home appliances to WAN-centric services (e.g. VoIP phones and other devices that need to be accessed from the WAN).

- **Good User Experience**
  Provide customers with branded user interfaces accommodated to their needs and experiences, as well as easily support plug&play devices in the LAN.

The following high-level use cases cover only a small number of application that are enabled by this service.

- **The customer is managing certain aspects of an IGD, such as his credentials, Wi-Fi SSID, NAT/PAT forwarding.**
  An OSGi-service running on the router provides an HTML-based user interface for the customer where he can change and manage various settings of the IGD. The OSGi-service validates the user's input and makes the necessary changes via the IGD Service.

- **The IGD is managed remotely via a TR-069 remote management agent.**
  The IGD vendor does not provide a TR-069 remote management service on his own. Instead, an OSGi-based TR-069 service is installed by the ISP. The TR-069 service manages the IGD via the IGD Service. In case an ISP supports an other management protocol than TR-069, that ISP installs an OSGi bundle that implements that management protocol.

- **An IGD vendor implements the UPnP Forum's IGD specification as an OSGi service.**
  So far, IGD vendors implement the UPnP IGD services as part of their firmware. That means that they have to implement a minimal UPnP stack. In an OSGi-enabled residential gateway that functionality could be moved to the OSGi framework. That would make this service more manageable and adaptable to changes in the protocol and environment. Another aspect is that only one UPnP stack needs to be installed on the IGD.

- **A SIP B2B User Agent manages the port forwarding of the IGD.**
  Services, such as a SIP B2B User Agent, could access functions of the IGD which need to be managed in order to provide certain services. In this use case, temporarily opening the WAN firewall and forwarding of...
certain ports to other devices in the LAN is an essential part of a SIP service that is running as an OSGi service on the IGD.

2 Application Domain

The main application domain for a Home Gateway Admin service is the home gateway that acts as a manageable device between the home network (LAN) and the public Internet (WAN). Even a low-priced router model includes a lot of basic network functions that need at least some simple administrative care, either by the network operator, the home user, or even by connected third-party equipment.

The network operator wants to offer an easy to use client interface for the basic operations the user might need to perform in order to install the device. This includes, for example, the provisioning of credentials or setting the identification for a local Wi-Fi network. Sometimes an automated setup process helps the home user to install a new device without any interaction at all.

An experienced home user can change some of the more arcane settings of the home gateway. For example, for security reasons he wants to change the Wi-Fi identification, enable or disable services, or he needs to change some other settings. For this he needs a rich user interface for the device-internal functions. Today, the home gateway hosts a web page or other user interface application that enables the home user to administer these settings.

The home user bought a VoIP-enabled phone. After he connects the phone to the local network and provided the necessary phone settings, he expects the phone to work properly. The phone itself “knows” how to connect to a telephony service in the Internet, but for receiving calls some adjustments in the home gateway have to be made in order to forward IP calls to the device in the LAN.

3 Problem Description

The UPnP Forum defines an network-side interface to the functionality of an Internet Gateway Device [3]. However, the UPnP Forum does not define a similar interface to the internal services of a home gateway, nor does the OSGi specification, yet. Another standards body that defined a specification to manage the management functions for a home gateway is the Broadband Forum [6]. The BBF’s Technical Recommendation TR-098 “Internet Gateway Device Data Model for TR-069” describes the Internet Gateway Device data model for the CPE WAN Management Protocol (TR-069) [5]. TR-069 defines the generic requirements of the management protocol methods which can be applied to any TR-069-enabled CPE.
Consider a residential service gateway that supports the UPnP Forum's IGD specification, and also runs OSGi technology to support networked residential services. A service running on the home gateway that wants to access and manage the gateway functions has no other choice as to call the IGD service interface via UPnP, even if both are running on the same hardware environment. Beside of the more complicated and error-prone service architecture and load on the LAN, a vendor of a home gateway needs to provide more resources to support access to the UPnP stack on the device, even if no other service on the OSGi part of the home gateway needs access to it.

A Home Gateway Admin, standardized by the OSGi Alliance, would provide well-defined means to the core functions of a home gateway, hiding complexity as well as vendor specific implementation details. A vendor, ISP, or other service provider could implement their IGD-enabled applications on top of the HG Admin architecture as a portable module, e.g. the HTML-based user interface to the router.

So far, no standardized OSGi-based HG-related architecture and means to access and manage IGD internally, and possibly other, functionality exists.

The following figure presents a rough sketch of an architecture that would utilize the OSGi Home Gateway Admin (HG Admin). The vendor specific core functions of the gateway are made available through a unified IGD Service interface, which enables various types of services, such as Management Agents, GUIs, and even the UPnP IGD service itself.

It is expected that an implementation of the Home Gateway Admin would usually contain native code parts.
4 Requirements

The following functional and non-functional requirements are given for the Home Gateway Device Service.

4.1 Functional Requirements

- An implementation of the Home Gateway Admin service MUST enable the management of the core functionality of an IGD.
- An implementation of the Home Gateway Admin service MUST provide means to map the functionalities of the Broadband Forum's TR-098 specification [6].
- An implementation of the Home Gateway Admin service MUST provide means to map the functionalities of the UPnP Forum's IGD 1.0 specification [3].
- An implementation of the Home Gateway Admin service SHOULD support the new functionalities of the upcoming Broadband Forum's TR-098 Amendment 2.
- An implementation of the Home Gateway Admin service SHOULD support the new functionalities of the upcoming UPnP Forum's IGD 2.0 specification.
- The Home Gateway Admin SHOULD support the management of further TR-069-managed services, such as TR-104 [7].
- The Home Gateway Admin MUST notify interested services when certain changes in the managed, e.g. the IGD, services happen. Examples:
  - connect, disconnect, and reconnect to WAN,
  - detect a new IP device, and
  - firewall intrusion detection.
- A service that has been notified by the Home Gateway Admin SHOULD be able to reject certain requests.
- The specification of the Home Gateway Admin MUST allow implementations to support and use IPv6.

4.2 Non-Functional Requirements

The following security considerations MUST be taken into account:

- Access to methods of the IGD's drivers MUST be controlled.
- The Home Gateway Admin SHOULD support the extensions to the UPnP IGD specification made by the Broadband Forum [6].
- The OSGi DMT Admin MUST be used in order to set and retrieve configuration values for managing the IGD.
• Events that originate in the core layer of the IGD must be forwarded to interested OSGi services by the OSGi Event Admin [11].

5 Technical Solution

5.1 Scope

The technical solution covers all needed mechanisms to access the low level IGD functions from within the OSGi service platform. It enables the implementation of e. g. a web-based configuration user interface, a UPnP Internet Gateway Device service, or a TR-069 management agent. However, the solution does not cover the implementation of any of these examples.

5.2 General Considerations

OSGi-based implementations of IGD management functions are tightly coupled with the IGD's operating system and core functions (as specified in the HGI Residential Profile [8]). Hence, considering these implementations to be drivers in the meaning of the OSGi Device Access specification [OSGi Service Compendium, chapter 103] is very recommended.

Moreover, plugging an additional hardware module to the IGD has a standardized process to find and install additional management bundles (Module Driver) as needed. For example, an additional IEEE 802.11n Wi-Fi access point might be plugged into the IGD, and the Device Access process finds and installs a suited driver that allows for managing this new module.

However, implementing the management bundles as Device Access “driver” is not mandatory.
The OSGi DMT Admin provides a generic interface to access management aspects of a device. Implementations of the DMT Subtree Admin and Vendor IGD Drivers have to use the DMT Admin service to offer a configuration-based interface in order to set and retrieve values. State variables of the underlying Home Gateways have representations in the DMT Admin. Actions are called by setting the appropriate state variables in the Device Management Tree.

Vendor IGD Driver bundles register one or more Data Plugins to manage the configurations of all or only a subtree of the IGD. A software vendor can choose to implement the whole IGD configuration functionality in one Vendor IGD Driver bundle or to split it into many bundles to separate the functionality.

Generic mapping rules are defined in chapter 5.3 to map IGD state variables, actions, and functions to a well-defined DMT configuration sub-tree. The root of this sub-tree is "./InternetGatewayDevice". Though this section mainly focuses on aspects of IGD management, the DMT Subtree Admin can be used manifold to manage overlapping subtrees in the DMT Admin in general.

Events that arise from the IGD core functions, for example when a disconnect from the network occurs, must be raised and distributed using the OSGi Event Admin service. No special event types are defined in this specification. Instead, event types from the DMT Admin services are used (s.[9].).

Within the OSGi service platform all events are considered as “active”. That means that any event will be distributed through the Event Admin service, and any permitted entity running in the OSGi service platform is able to receive these events. The Broadband Forum's TR-069 specification allows for “passive” notifications, which are handed over to a Remote Management System (RMS) as part of the next scheduled contact between the management agent locally installed on a home gateway and the RMS. For this case, the management agent is responsible for keeping track of all events and storing them persistently until the next contact.
5.3 Roles and Functional Blocks

Illustration 3 presents the functional blocks defined or referred to in this specification and are explained in this section.

The roles used in the diagram are:

- **Gateway Vendor**: The vendor of the gateway. He is responsible for assembling the hardware and the software for the IGD, and usually delivering it to the Operator. He might develop and manufacture software and hardware components himself, or order them from a third party.

- **Framework Manufacturer**: The manufacturer of the OSGi Framework and the OSGi services. He provides an implementation to the Gateway Vendor, possibly tailored to the specific hardware.

- **Operator**: The Operator defines the product requirements of the IGD and provides these IGD to his customers. He orders IGD from the Gateway Vendor. This order might include the management-related OSGi services, but he can also obtain them from a third party provider.

The **HG Core Stack** with the **Operating System** and the **IGD Core Functions** are usually provided by the Gateway Vendor. The IGD Core Functions layer contains functionality for all the basic aspects of a IGD, such as configuration of the WAN interfaces, establishment and maintaining WAN connections, but also a firewall, NAT and routing functionality.

The **Java & OSGi stack** and other OSGi services are provided by the Framework Manufacturer. Services that are mandatory by this specification are **DMT Admin**, **Event Admin**, and the **DMT Subtree Admin**. As explained before, the **Device Access** service is optional. An Operator could define other means to install IGD plugin bundles.

The **Management Stack** contains the necessary services for managing an IGD. The **IGD Base Driver** implements the necessary base functions to access and manage the **IGD Core Functions**. Additional functions and/or drivers that can be installed if necessary and on-demand are represented in the **Additional IGD Services** block. The **IGD
**Base Driver** can be implemented and provided as a single bundle, or split into as many bundles as necessary. This specification doesn't imply any restrictions on the number of Vendor IGD Driver bundles. Finally, the Management Applications are the actual applications that are used to manage a service. This could be a TR-069 management agent for remote management of the IGD, a web user interface for the customer, or an implementation of the UPnP IGD service.

## 5.1 DMT Subtree Admin

This section specifies the operation of the DMT Subtree Admin service

### 5.1.1 DMT Admin restrictions

Using the DMT Admin to implement the a Home Gateway Administration service is a mandatory requirement because it allows for a loose coupling between the Management Services and the Vendor IGD Drivers. To understand the function and work of this loose coupling a discussion on the restrictions of the DMT Admin is necessary. The HG Admin is implementing solutions to overcome these restrictions.

The DMT Admin has the restrictions that subtrees of configuration data cannot overlap. A DMT Admin Data Plugin that was registered to manage, for example, the sub-tree 
"/InternetGatewayDevice" receives all requests to data objects below that name. It is not allowed to register a second Data Plugin that, for example, manages the sub-tree "/InternetGatewayDevice/wan". Because of this restriction services which register later in time are not allowed to register a Data Plugin to manage the tree or any sub-tree below "/InternetGatewayDevice". Figure 4 illustrates overlapping subtrees, one of an IGD Data Plugin and the overlapping one, the Vendor IGD Driver Data Plugin that can manage the data model for SomeVendorDevice.

![Illustration 4: Overlapping Subtrees in DMT Admin](image-url)
Another restriction concerns the raising of events. The DMT Admin only raises events when a part of the managed configuration is changed via a Data Plugin. A bundle that likes its configuration been managed by the DMT Admin and therefore registers a Data Plugin cannot use the DMT Admin to notify interested bundles if some part of its configuration has changed through other means. An example for this is that in an IGD the connection on the WAN interface was established or disconnected. This event arises in the IGD Core Functions layer. The responsible driver bundle can reflect this in its internal configuration mapping, but there are no means to trigger the DMT Admin to raise an event.

5.1.2 Managing and Mapping DMT Admin subtrees

The solution for the non-overlapping subtrees problem described above is to let the DMT Subtree Admin handle the mappings between the actual subtree under a specific configuration in the DMT Admin tree and the registered subtree of any Data Plugin of driver bundles. For this, the following procedure apply:

- The DMT Subtree Admin must implement the `ManagedServiceFactory` interface and register a ManagedServiceFactory service with the `PID=org.osgi.service.dmtsubtree`.

- For each configuration dictionary received, the DMT Subtree Admin must create and register a new Data Plugin. The only configuration value of this dictionary is `rootnode`. The Data Plugin must be registered with the property `dataRootURIs` set to the value of `rootnode`. This value indicates the subtree below the DMT root node it manages. The value contains only the name of the first path element, not a fully qualified path.

- When more than one dictionary contains the same value for `rootnode` only the first one received is registered. All others must be ignored.

- The DMT Subtree Admin must unregister the Data Plugin when the according configuration dictionary is deleted.

- The DMT Subtree Admin tracks Data Plugin services that have the properties `dataRootURIs` and `configurationPaths` set. The Filter string to track this is `(&(objectclass=info.dmtree.spi.DataPlugin)(dataRootURIs=*)(configurationPaths=*))`.

- A bundle that wants to be configured through the DMT Admin and that maps its configuration subtree into the subtree of another bundle must register a Data Plugin with the property `dataRootURIs` set to any value. Since this property is an array of Strings, a Data Plugin can be registered to handle more than one configuration path. It is recommended that the bundle first checks with the DMT Admin for the availability of the desired path within the DMT Admin tree. This configuration path must not overlap with any previous registered Data Plugin. In addition it also sets the following properties:

  - `configurationPaths`: this property defines the Data Plugin's intended position under a tree in the DMT Admin. This property contains an array of Strings. The number of entries in this array must match the number of entries of the `dataRootURIs` property. The value for that property must not start or end with a slash (character `/`). The path must be fully qualified, i.e. it starts from the beginning of the DMT Tree. Example: `./InternetGatewayDevice/Devices/SomeVendorDevice`. The DMT Subtree Admin takes the first path element after the root node (here: `InternetGatewayDevice`) as an identifier and associates the tracked Data Plugin to one of its own registered Data Plugins with the property `dataRootURIs` set to the same value. If none of the own registered Data Plugins matches then the tracked Data Plugin is ignored for now. Note, that later the DMT Subtree Admin might register a Data Plugin that matches the (then) ignored tracked Data Plugin. In this case the tracked Data Plugin must be associated now.

---

1 The names of the properties defined here might change when a more generalized mechanism to circumvent the problems with the DMT Admin is specified.
Overlapping paths are allowed. The resolution is handled by the DMT Subtree Admin, depending on the value of the `configurationMultiples` property (s. below). This property is mandatory. There is no default.

- `configurationMultiples`: this property defines whether there could be multiple instances for the given configuration path and objects. This property contains an array of Strings. If set, the number of entries in this array must match the number of entries of the `dataRootURIs` property. An example is if an IGD has more than one WAN interfaces (e.g. one DSL and one 3G) and needs a driver for each of the WAN ports. In this case two different Data Plugin implementations would be used for the configuration of the drivers.

It is the responsibility of the DMT Subtree Admin to assign a unique identifier for the configuration path to ensure a correct mapping for each Data Plugin. This identifier must be uniquely assigned for the configuration path and Data Plugin and stored persistently by the DMT Subtree Admin. This means that a once assigned identifier is always assigned to the same Data Plugin, even when that Data Plugin is unregistered and registered again, or the OSGi framework is restarted. Note, that the unique identifier storage might be removed when the DMT Subtree Admin is uninstalled. It is therefore necessary for a management domain to define other means to ensure the persistency of the unique identifiers.

Allowed values for this property are the string values “true” and “false”. If the value is `true` then multiple instances are created. If the value is `false` then only one instance for the given configuration path is allowed. In the later case a newer registration of a Data Plugin with the same configuration path does not override an existing one ie. it is ignored.

It is possible and allowed for a node to contain both multiple instances of subtrees (as indicated by the property `configurationMultiples` set to true) and non-multiple subtrees. In other words, a node can contain „multiple“ and „normal“ child nodes. A „normal“ child node, though, must not overwrite the path name of an existing „multiple“ child node. This includes „multiple“ child nodes that are currently mapped and those that are currently not mapped (but the multiple-ID has already been assigned). In case an overlapping is detected by the DMT Subtree admin, a DmtException with the cause `NODE_ALREADY_EXISTS` is thrown.

The setting of this property is optional. The default is `false`.

- `dmtActions`: This optional property contains a relative path that must point to a node below the Vendor Data Plugin’s subtree (specified in `dataRootURIs`) that contains leaf nodes for internal DMT Subtree Admin management purposes. This way the DMT Subtree Admin can communicate with the Data Plugins it manages (see also 5.1.4). This property contains an array of Strings. If set, the number of entries in this array must match the number of entries of the `dataRootURIs` property. There is currently only one leaf node defined that must be supported by a Vendor Data Plugin that sets this property. See also illustration 7 for an example.

  - `mappedPath`: The value of this leaf node is a String and contains the path under which the DMT Subtree Admin has put the mapping to this Vendor Data Plugin. If the leaf node does not exist, it is created by the DMT Subtree Admin. It is removed by the DMT Subtree Admin when the Vendor Data Plugin is unmapped.

- The DMT Subtree Admin receives tracker callbacks for registered (and unregistered) Data Plugin services. It is responsible to map the Configuration paths, which are assigned through the configuration, to the real Data Plugins that handle the subtrees accordingly.

- If a `ManagedFactoryConfiguration` is changed or removed, all tracked Data Plugins must be newly associated or the associations must be removed, accordingly.

- Configuration requests from Management Applications are done through the DMT Admin. The DMT Admin will request a DMT Session object from the DMT Subtree Admin's Data Plugin that is registered for handling the designated path. This DMT Session then will handle the actual requests by mapping the
request paths and forwards the requested actions to the actual driver bundles' Data Plugins. This is done by getting DMT Session objects via the DMT Admin from the actual Data Plugins. When closing, committing or rolling back sessions the DMT Subtree Admin's DMT Session is responsible for performing the according actions on these session objects (see also 5.1.4 for a more detailed discussion).

The following illustration 5 presents the general relationships between the involved components.

![Illustration 5: Data Plugins Relationships]

The creation and usage of Session objects are presented in illustration 6.
It is strongly recommended, though not mandatory by this specification, to implement all Data Sessions as Transactional Data Session (s. DMT Admin Service Specification, chapter Data Sessions). In case of transactions the DMT Subtree Admin's DMT Session is responsible to create transactions, and to call the appropriate actions when the transaction is committed or rolled-back by the using Management Service.

It is strongly recommend, though not mandatory by this specification, to implement the DMT Session objects of the DMT Subtree Admin’s Data Plugin in a way that it only requests DMT Session objects on-demand from the drivers' Data Plugins that are involved in the requests.

### Mapping Rules Examples

The following table shows examples which reflect the mapping rules defined in the previous section. It is assumed that the DMT Subtree Admin registered a Data Plugin for the `dataRootURIs = ./InternetGatewayDevice`.
Table 1: Mapping Rules Examples

The following illustration shows an exemplary tree, the properties for the mappings as well as the values set by the DMT Subtree Admin.

5.1.4 Additional expected behaviour when using DMT Subtree Admin

The following rules apply when implementing and using the DMT Subtree Admin.

- Since the DMT Subtree Admin needs to open sessions to other Data Plugins, some restrictions for accessing the DMT Tree apply. There must be no session that accesses the root of the DMT in a way that acquire a lock in any way. Otherwise the DMT Subtree Admin is not able to get sessions to subtrees via other Data Plugins.

- The DMT Admin specification does not specify the minimum length of a node name and leaves this up to the implementation to decide. A domain that defines a data model and wants this to be managed by the DMT Admin must specify the minimum node name length the DMT Admin must support.
• The DMT Admin specification does not specify the length for the timeout for a session. Since heavy configuration and management activities can take some time, especially when done via a remote management system, the domain that defines the actual integration environment must define this value for the DMT Admin to be of appropriate length.

• A Vendor Data Plugin can be mapped into the DMT Tree under any path, even if between the root node and the mapping point no „real” nodes exists. Therefore, the DMT Subtree Admin must emulate the missing nodes in between. These nodes can be accessed only for traversing. Any attempt to write to such a node must result in an DmtException with the case PERMISSION_DENIED.

• When a transactional session is closed, committed, or rolled back the DMT Subtree Admin must call the appropriate actions on all the DataSessions it opened during the session. The order in which the sessions are handled is defined as follows: The DMT Subtree Admin must check which one of the involved sessions handled the longest path (the deepest path from the root node in regard to number of path elements) and close/commit/rollback this session first. If there is more than one session for which this criteria is true (same number of path elements from the root node) then the session that has been opened last must be closed/committed/rolled back first.

• If a Vendor Data Plugin is registered with the configurationMultiples set to „true” the actual path in the mapped configuration tree is determined by the DMT Subtree Admin. By default, there is no easy mean for a Vendor Data Plugin to get this path. This is, however, essential if the data model that is implemented by the Vendor Data Plugin itself contains tables or something similar that is also needed to be registered in a multiple fashion. The Vendor Data Plugin can manage this on its own, but it would be much simpler if the DMT Subtree Admin could manage these multiple parts, too.

To solve this problem the Vendor Data Plugin can make use of the dmtActions property. When the DMT Subtree Admin does the mapping it creates the leaf node mappedPath<n> in the Vendor Data Plugin. The Vendor Data Plugin can now determine the real path and start to register the multiple subtrees below that path.

5.1.5 Raising Events

As described above the DMT Admin only raises events for nodes and values that are changed through a DMT Session object. Values that change on the level of a Data Plugin (e.g. an internal value is changed by external means) are not evented. The following steps specify the procedure that must be implemented for the DMT Subtree Admin to raise events.

The Vendor Driver must raise events as follows:

• If a value in a driver has changed by any other means than through its associated Data Plugin, an event can be raised. It is up to the underlying management model of the driver bundle to decide only to raise events for some values.

• The events raised must use the Event Admin and must conform to the description of section 117.10.1, Event Admin based Events [11].

• Only events with the following event topics must be raised. Any other event topic is not allowed to be raised by a vendor driver.
  
  • info/dmtree/DmtEvent/ADDED: New nodes were added.²
  
  • info/dmtree/DmtEvent/DELETED: Existing nodes were removed.

² An example for an ADDED event could be when a DHCP server in the IGD registers a new client's MAC address.
•  `info/dmtree/DmtEvent/REPLACED`: Existing node values or other properties were changed.

•  No session.id property must be set. This distinguishes an event that is raised by the vendor driver from one that is raised by the DMT Admin.

•  The `nodes` property contains the URI of the affected nodes. The URI path must start with the same value the Vendors Driver Data Plugin was registered with (the `dataRootURIs` property).

•  The `newnodes` property is not set.

The DMT Subtree Admin must handle events as follows:

The DMT Subtree Admin is catching all DMT Admin events with the topic of either "`info/dmtree/DmtEvent/ADDED`", "`info/dmtree/DmtEvent/DELETED`", or "`info/dmtree/DmtEvent/REPLACED.`" that originate from any Vendor Driver Data Plugin, indicated by the missing `session.id` property. If an event originates from a vendor’s driver, then a new DMT Admin event is raised, where the creation is following these rules:

•  A new Event object is created with the same topic as the original event and no `session.id` property.

•  The values of the `nodes` property are mapped to their virtual counterparts in the according mapped subtree of the DMT Subtree Admin. The following rules apply here:

  •  If the `configurationMultiples` property for the event-originating Data Plugin is false:
    
    •  The prefix of the node value that matches the value of `dataRootURIs` is removed. Take R as the remaining value.
    
    •  The new value is constructed as follows:
      
      "./" + `<the first path element of the mapping DMT Subtree Admin's Data Plugin>` + `<according value of `configurationPaths` for that Data Plugin>` + `<remaining value R>`

  •  If the `configurationMultiples` property for the originated Data Plugin is true:
    
    •  The prefix of the node value that matches the value of `dataRootURIs` is removed. The remaining value is R.
    
    •  The new value is constructed as follows:
      
      "./" + `<the first path element of the mapping DMT Subtree Admin's Data Plugin>` + `<according value of `configurationPaths` for that Data Plugin>` + `<uniquely assigned identifier for that Data Plugin>` + `<remaining value R>`

  •  The new event is raised via the Event Admin service.

Note, that in the end two events are raised: one that is raised by the Vendor Driver and one by the DMT Subtree Admin.

Illustration 8 presents the general relationships between the involved components.
5.1.6 Event Mapping Examples

The following table shows examples which reflect the mapping rules defined in the previous section. It is assumed that the Vendor Driver has been registered as shown as in the examples in section 5.1.3.

<table>
<thead>
<tr>
<th>Node value of the vendor’s driver event</th>
<th>Mapped node value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>./vendor_x/firewall/enabled</td>
<td>./InternetGatewayDevice/service/firewall/enabled</td>
<td></td>
</tr>
<tr>
<td>./vendor_x/interface/WAN/disconnected</td>
<td>./InternetGatewayDevice/interface/WAN/1234/disconnected</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Event Mapping Examples

5.2 Mapping for an Internet Gateway Device

This section specifies how a subtree in the DMT Admin is mapped for the data model for an IGD to the DMT tree and various Vendor IGD Data Plugins.

5.2.1 Registering IGD related Data Plugins

The following rules apply for mapping an IGD management tree.

- A new ServiceFactoryConfiguration `org.osgi.service.dmtsubtree` is created. The value „InternetGatewayDevice“ is assigned to the key `rootnode`. The DMT Subtree Admin creates a new Data Plugin that is responsible to manage and map the configurations under the subtree „./InternetGatewayDevice“ in the DMT Admin.
Vendor IGD Drivers must register and map their Data Plugins via the DMT Subtree by registering the Data Plugins with the property `configurationPaths` set. The value of the property must start with "/InternetGatewaysDevice" followed by a slash (/) and the appropriate sub-path of the Data Plugin's configuration path.

Optionally, the Vendor IGD Data Plugins can have the property `configurationMultiples` set to `true` or `false`, depending on whether their configuration tree needs to be managed in a multiple fashion.

### 5.2.2 Remote Management Events and Notifications

Within the OSGi service platform all events are considered as “active”. That means that any event will be distributed through the Event Admin service, and any permitted entity running in the OSGi service platform is able to receive these events.

Broadband Forum’s TR-69 specification allows for “passive” notifications, which are handed over to a Remote Management System (RMS) as part of the next scheduled contact between the management agent and the RMS. For this case, the management agent is responsible for keeping track of all events and storing them persistently until the next contact. Specification of the management agent functionality is out of scope of this specification.

### 5.3 Mapping rules to DMT Admin Management Tree

This chapter defines mapping rules for mapping various management standards to the DMT Admin tree.

#### 5.3.1 Mapping from Broadband Forum TR-069 and TR-098 to DMT

This specification does not define management entities of a home gateway. However, node paths in the DMT look slightly different than in the TR-069 specification. Following mapping rules apply:

- Every node path for TR-069 data models must start with "/" plus the root path of the respective data model. For TR-098 this is "/InternetGatewayDevice”.
- Every dot (.) in the original TR-098 path must be substituted by a slash (/).
- Arrays must be represented by using the index number as a node (e.g. "/InternetGatewayDevice/WANDevice/1").
- Any TR-098 element that ends with "numberOfEntries" must be computed, not stored.

Like Java, TR-069 knows data types. The following table presents the mapping between TR-069 and DMT Admin data types.
### TR-069 data type | DMT Admin data type | Remarks
--- | --- | ---
String | FORMAT_STRING | Created by DmtData(String)
int | FORMAT_INTEGER | Created by DmtData(int)
unsignedInt | FORMAT_STRING | The unsignedInt is an unsigned integer. Since there is no suitable data type defined in Java, this data type should be mapped as String. Note, that a client that access this value needs some knowledge about the underlying data model, otherwise it has no means to detect the actual type (unsigned int) be requesting the data type.
boolean | FORMAT_BOOLEAN | Created by DmtData(boolean)
dateTime | FORMAT_STRING | The dateTime is a String object that is interpreted as an dateTime type defined in ISO 8601, which is used as a value of date and time in TR-069. Since there is no corresponding data type defined in DMT admin, this data type should be mapped as String. Note, that a client that access this value needs some knowledge about the underlying data model, otherwise it has no means to detect the actual type (dateTime) be requesting the data type.
based64 | FORMAT_BASE64 | Created by DmtData(byte[], boolean)

*Table 3: Mapping of data types between TR-069 and DMT Admin*

#### 5.3.2 Mapping from UPnP IGD 1.0

The TR-098 data model is a superset of the UPnP IGD 1.0 specification, so all rules that apply to TR-098 apply also to the mapping of UPnP IGD 1.0.
5.3.3 Examples

The following table presents an example mapping between TR-098, UPnP IGD 1.0, and the DMT Admin tree.

<table>
<thead>
<tr>
<th>TR-098</th>
<th>UPnP IGD 1.0</th>
<th>DMT Admin tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>InternetGatewayDevice.Layer3Forwarding.</td>
<td>&lt;&lt;service&gt;&gt; Layer3Forwarding</td>
<td>./InternetGatewayDevice/Layer3Forwarding</td>
</tr>
<tr>
<td>DefaultConnectionService</td>
<td>DefaultConnectionService</td>
<td>as TR-98</td>
</tr>
<tr>
<td>ForwardNumberOfEntries</td>
<td>not needed</td>
<td>computed</td>
</tr>
</tbody>
</table>

*Table 4: Example mapping between TR-098, UPnP IGD, DMT Admin Tree*

6 Considered Alternatives

6.1 Not using DMT Admin

DMT Admin does not meet all important requirements regarding the management of a Home Gateway, so it would be tempting not to use it rather than specifying a service that meets the requirements 100 percent. But specifying another service which replicates DMT Admin functionality by 80% and adds the 20% needed for IGD management would be inefficient as well.

An implementation that would mostly replicate most of the functionality of the DMT Admin would more burdensome than the proposed solution. This solution has the advantage that it enhances the DMT Admin without changing its specification.

7 Security Considerations

Read and write access to the following services, objects and entities must be restricted to authorized bundles and services only:

- Any access to new subtrees, especially the "./InternetGatewayDevice" subtree of the DMT Admin tree. Access to this subtree must be handled by using Dmt Principal Permission, DMT Permission and the appropriate permission actions. Access to the subtree must be disabled by default.

- Every vendor-specific subtree a vendor driver uses to add his own configuration tree to the DMT Admin. Access to the vendor subtrees must be handled by using Dmt Principal Permission and the appropriate
permission actions. Access to the sub-tree must be disabled by default. Only the DMT Subtree Admin bundle should have access to the values of the vendor's subtree.

- Data Plugins of the DMT Subtree Admin. By using the OSGi Permission Admin service [12], no other than the DMT Admin is allowed to access methods of the DMT Subtree Admin's Data Plugins.

- Data Plugins of vendor-specific IGD drivers. By using the OSGi Permission Admin service [12], no other than the DMT Admin is allowed to access methods of the vendors' Data Plugins.

Detailed information on security of the DMT Admin Service specification can be found at [10].

Events raised by the DMT Subtree Admin or vendor-specific drivers can be received by any bundle, which has enough TopicPermission to receive DMT Events.

---

8 Document Support

8.1 References

[3]. Internet Gateway Device (IGD) Standardized Device Control Protocol V 1.0, UPnP Forum
[4]. TR-064, Broadband Forum LAN-Side DSL CPE Configuration, BBF, May 2004
[6]. TR-098, Internet Gateway Device Data Model for TR-069, Amendment 1, BBF, November 2006
[7]. TR-104, Provisioning Parameters for VoIP CPE, BBF, September 2005
[9]. OSGi Companion Specification, DMT Admin, Events, Section 117.10
[10]. OSGi Companion Specification, DMT Admin, Security, Section 117.12
[12]. OSGi Service Platform Core Specification, Permission Admin Service, Chapter 10

8.1 Author’s Address
### 8.2 Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2BUA</td>
<td>Back-to-back User Agent</td>
</tr>
<tr>
<td>BBF</td>
<td>Broadband Forum</td>
</tr>
<tr>
<td>CPE</td>
<td>Customer-Premises Equipment</td>
</tr>
<tr>
<td>DMT</td>
<td>Device Management Tree</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HG</td>
<td>Home Gateway</td>
</tr>
<tr>
<td>HGI</td>
<td>Home Gateway Initiative</td>
</tr>
<tr>
<td>IGD</td>
<td>Internet Gateway Device</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>PAT</td>
<td>Port Address Translation</td>
</tr>
<tr>
<td>RMS</td>
<td>Remote Management System</td>
</tr>
<tr>
<td>RTSP</td>
<td>Real-Time Streaming Protocol</td>
</tr>
<tr>
<td>SSID</td>
<td>Service Set Identifier</td>
</tr>
<tr>
<td>VoD</td>
<td>Video on Demand</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over IP</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
</tbody>
</table>

Copyright © Deutsche Telekom AG 2009 All Rights Reserved
8.3 End of Document
Abstract

Different industries are interested in applying OSGi to advance their businesses, in which remote management is a key issue. In the residential area, the TR-069 is the one of the de-facto standard protocol for remote management. The best way to realize remote management based on the TR-069 on OSGi is utilizing DMT Admin service, which has been defined in the OSGi Alliance for the mobile device management. In this case, TR-069 is implemented as a protocol adapter of the DMT Admin. The DMT Admin service, however, has the interfaces inspired by OMA-DM, which is the de-facto standard protocol for mobile area. Although these two protocols, the TR-069 and the OMA-DM, have the similar objectives and functionality, there are several architectural differences between them. Due to the differences, the mapping guideline between the TR-069 and the DMT Admin for implementing a protocol adapter is needed. This RFC defines the handling of notification mechanism, the mapping between TR-069 RPCs and the DMT Admin interfaces and data type translation.
1 Document Information

1.1 Table of Contents

1 Document Information ................................................................................................................. 2
  1.1 Table of Contents ................................................................................................................ 2
  1.2 Terminology and Document Conventions ............................................................................ 3
  1.3 Revision History .................................................................................................................. 3

2 Introduction .................................................................................................................................. 4

3 Application Domain ..................................................................................................................... 5

4 Problem Description .................................................................................................................... 5
  4.1 Management agent making use of OSGi standardized Java interfaces ......................... 6
  4.2 Mobile specification approach ............................................................................................. 6
  4.2.1 Support for TR-069 notifications ....................................................................................... 6
  4.2.2 Limitations in the number of services available in the DMT .............................................. 7
  4.2.3 Mapping TR-069 to the OMA-DM inspired DMT model .................................................... 8
  4.3 Conclusion .......................................................................................................................... 8

5 Requirements ............................................................................................................................... 9

6 Technical Solution ....................................................................................................................... 9
  6.1 Basic architecture of TR-069 protocol implementation ....................................................... 10
  6.2 Notification handling for TR-069 .......................................................................................... 10
  6.3 RPC mapping for TR-069 .................................................................................................... 14
  6.4 Session management .......................................................................................................... 17
  6.5 Mapping of data types ......................................................................................................... 19
  6.6 Error code and SOAP Fault ................................................................................................. 20
  6.7 Limitation of notification ....................................................................................................... 22

7 Considered Alternatives .............................................................................................................. 22

8 Security Considerations .............................................................................................................. 23

9 Document Support ....................................................................................................................... 23
  9.1 References .......................................................................................................................... 23
  9.2 Author's Address ................................................................................................................ 24
  9.3 Acronyms and Abbreviations ............................................................................................... 24
  9.4 End of Document .................................................................................................................. 24
1.2 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 9.1.

Source code is shown in this typeface.

1.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Initial  | Sep. 16 2008 | Initial Draft  
  Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp |
| 2nd      | Jan. 20 2009  | 2nd Draft  
  Generic setter/getter methods are added.  
  Limitation of notification is described.  
  Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp |
| 3rd      | Jun. 2 2009   | 3rd Draft  
  The technical solution has been changed to describe guideline of RPC mapping for TR-069.  
  Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
  Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp |
| 4th      | Jul. 24 2009  | 4th Draft  
  The detail of the Notification handling and the RPC mapping is defined. And the Session Management is added.  
  Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
  Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp |
| 5th      | Sep. 18 2009  | 5th Draft  
  The number of this RFC has been changed to 148.  
  The behavior for notification and session management is changed based on the REG F2F discussion.  
  The error code mapping is added.  
  Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
  Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp |
2 Introduction

Traditionally, fixed telecommunication operators don’t have knowledge about what runs in the customer's local area network (LAN). They provide connectivity and manage the wide area network (WAN) that provides this connectivity, but they do not know anything about the devices and networks behind the gateway (xDSL mainly) that interconnect WAN and LAN. Recently the need for management of customer networks and devices is increasing in order to make the deployment of new complex services at home (home automation, tele-health, VoIP, IPTV, surveillance, etc) feasible with reasonable costs. For example to avoid sending technicians to the customer premises for solving problems.

There are two main kinds of devices that need to be managed in operator’s business: those which come from the fixed business managed through TR-069 and standardized by the Broadband FORUM, and those which come from the mobile business, managed through OMA-DM and standardized by the Open Mobile Alliance. The DMT Admin Service specification of OSGi covers the OMA-DM ones. In addition, the design of DMT Admin Service specification potentially allows adaption of other remote management protocols other than the OMA-DM.

The best way to realize remote management based on TR-069 on OSGi is utilizing the DMT Admin service. In this case, TR-069 is implemented as a protocol adapter of the DMT Admin. However, there are several architectural differences between the TR-069 and the OMA-DM, although these two protocols have the similar objectives and functionality.

Therefore, this RFC defines the mapping guideline between the TR-069 and the DMT Admin to use them together. In addition, this RFC defines the handling of notification mechanism and data type translation.

Copyright © NTT Corporation 2009 All Rights Reserved
3 Application Domain

Driven by triple play service delivery in the home network, fixed line access service providers have the need to configure home devices to ensure the proper service delivery. Broadband Forum's CPE WAN Management Protocol (CWMP, alias TR-069) enables them to do this. By using a remote management server (Auto Configuration Server, ACS), they are able to manage TR-069 enabled devices. TR-069 provides them with possibilities to configure parameters, be informed of parameter changes (notification), start diagnostic tools, update firmware, etc.

Similarly, for the mobile world, the OMA defined the OMA-Device Management specification for remote management of mobile devices. OMA-DM offers similar tools to the mobile service providers as TR-069 to fixed line service provider, but OMA-DM is of course tailored to the specifics of the mobile environment.

As OSGi technology offers a flexible software environment for all these different devices, the remote management of the platform is of interest for both fixed and mobile service providers. As such, it should be possible to integrate the remote management of the OSGi platform, and the applications running on top of it, in the existing management infrastructure.

The DMT Admin service with its mobile management tree in the Mobile specification for OSGi R4 standardizes the remote management of an OSGi platform. As it is largely inspired by OMA-DM, it needs to be evaluated for multi protocol support.

4 Problem Description

In a scenario in which service providers offer a growing number of services, to use specific solutions for the management of those services is not the most suitable option. To speed up the deployment of these services, such as triple play, home automation or tele-health, it is essential to offer general management solutions that allow for the management of a large number of services and the flexible life-cycle management of applications.

These devices usually are already managed by a standard protocol, so it makes sense that an OSGi framework, which hosts the services, running on a device could be managed in the same way as the other resources of the device. Of course, the remote management should be fully integrated in the existing remote management solutions of the service provider to avoid duplicating management infrastructure and to increase performance on the devices.

Currently, there are two options in OSGi for remote management:

- create a management agent bundle making use of the Java interfaces,
create a protocol adapter bundle that interacts with the DMT Admin service, as defined in the OSGi Mobile specification.

4.1 Management agent making use of OSGi standardized Java interfaces

Currently, for the management of a bundle, the OSGi specifications define different Java objects with which a management application can interact. Using this approach, a management agent can implement extensive management of the OSGi framework, as well as any service standardized. Mapping the Java interfaces to the specific remote management protocol and data model tree is up to the management agent.

For runtime interaction with a bundle, a bundle can register a service in the service registry. However, this service interface is not standardized. Also, mapping the service interface to a general management model is not standardized. A current approach is to implement a proprietary service interface on all bundles to be managed. By tailoring this interface so that it easily maps to the management protocol primitives, it is simple for the management agent to map remote management commands to the bundle's service interface. The disadvantage is the proprietary service interface, so that 3rd party bundles might not be compliant.

As a conclusion we can say that this current approach allows for extensive remote management of any aspect of the OSGi platform, but lacks a standardized service interface definition for bundles to implement.

4.2 Mobile specification approach

The Mobile Expert Group has provided its own solution based on the OMA [3] Device Management [4] specification to provide a remote management solution. The OSGi Mobile specification contains two chapters related to remote management:

- chapter 3: detailing the mobile management tree
- chapter 117: detailing: the DMT Admin service, bundle plugin interface specification, the notification service

The Device Management Tree model of OMA-DM was chosen as meta-data model and operational model. However, it was intended to be mappable to other protocols.

An analysis of mapping the Mobile specification DMT model to TR-069, however, shows that the current DMT model approach (as defined in the OSGi R4 Mobile Specification) introduces some issues. For example:

- Limitations for active or passive notifications on any parameter in the object tree
- A limited number of services have been mapped to the DMT model
- The complexity of mapping a new protocol to the OMA-inspired DMT model, which could imply performance issues on limited devices.

4.2.1 Support for TR-069 notifications

TR-069 offers the feature of active and passive notifications. By setting a parameter's notification attribute, a remote manager requests to be notified with the parameter's new value at the time the value changes (active notification) or at the next periodic inform (passive notification). Notification can be configured on any parameter of the TR-069 object tree. This approach enables the remote manager to be informed not only of changes in status variables of the platform, but also of configuration changes performed by a local manager, e.g. through a local Web interface.

The Mobile specification offers a few features that could help to implement TR-069 notification support:
The DMT Admin service sends events using the Event Admin service when operations have been performed on nodes (nodes added, removed or copied; node values changed etc.)

The OSGi Notification service defines a way to alert a remote management server. Protocol adapters on their turn have to implement a RemoteAlertSender interface (and register it) for use by the notification service. Notifications are sent by calling sendNotification on the notification service:

The Monitor Admin service: A bundle can register a Monitorable service, to be used by the Monitor Admin service. By registering a Monitorable service, the bundle exposes access to a number of status variables. Notification can be implemented by the StatusVariable provider. If it does, it will call the update method on the Monitor Listener. The Monitor Admin service then generates an event on the Event Admin service. The Monitor service is currently also represented in the DMT tree.

Two problems arise when trying to map the current approach to TR-069:

TR-069 defines that notification is applicable to any parameter in the object tree.

Currently, the DMT Admin service only send events for operations on DMT nodes that were performed using the DMT Admin API. The DMT Admin Spec does not require Data Plugins to send events when DMT nodes are changed through except the DMT Admin API. It is called internal changes hereafter. For example: if configuration changes are performed by using the Configuration Admin service API, there might be no events sent. Such internal changes should be supported for TR-69 notification. However, it cannot be supported.

The OSGi Monitor service only supports notification of changes on Status Variables, exposed through a Monitorable service, and enabled by the bundle to support on-change notification (i.e. dynamic Status Variables).

Requesting notification is not fully under the control of the remote manager. In the case of a bundle using the notification service, there is no standardized way to configure the bundle to send alerts when the value of one of the implemented DMT nodes changes. In the case of the monitor service, the sending of events can be controlled, but is limited to dynamic Status Variables.

The current DMT Admin service has no attributes properties on its nodes to be used to configure notification behavior, such as active notification and passive notification defined in TR-069. Therefore, a remote manager cannot control the notification behavior of DMT nodes in a standardized way.

To conclude, the current options, as provided in the Mobile specification, limit notification of parameter changes to StatusVariables, explicitly enabled for monitoring. There is no standardized approach available to monitor changes on any node in the DMT.

**4.2.2 Limitations in the number of services available in the DMT**

The OSGi R4 Mobile specification mapped a number of services to the DMT tree. However, these services are limited to the services listed in the Mobile Specification. Other interesting services, as listed in the OSGi R4 Service Compendium are not yet mapped (standardized) in the DMT tree:

The DMT tree defined in the Mobile Specification contains objects for the following services:

- Configuration Admin service
- Log service
Monitor Admin service

Application Admin service

Conditional Permission Admin service

Deployment service

A number of areas that could be of interest to a remote manager are currently missing in the DMT tree:

Startlevel management

Bundle management: managing individual bundles as opposed to deployment packages (inventory, life-cycle management, exported services, …)

Service management: getting a remote view on services registered in the service registry

Permission Admin management

Home Gateway Core Function management: handling Home Gateway core functions, such as firewall configuration and port forwarding control, from bundles running on an OSGi framework

4.2.3 Mapping TR-069 to the OMA-DM inspired DMT model

Within the OSGi Mobile specification, the choice has been made to model the DMT after OMA-DM.

As a result, creating an OMA-DM protocol adapter is quite straightforward. Although no major hurdles have been identified in creating a TR-069 protocol adapter, it is less straightforward:

- The TR-069 RPC primitives have to be translated to the DMT Admin service interface methods (which are OMA-DM RPC inspired).
- The TR-069 tree has to be mapped to the DMT tree. Translating object model specific features like DMT meta nodes, or TR-069 attributes is not straightforward. It might require specific extensions to the DMT, e.g. to support TR-069 attributes, or a single node in the DMT might result in multiple objects in a TR-069 data model, etc.
- The TR-069 data types have to be mapped to the DMT Admin data types. However, TR-069 data types, such as “unsignedInt” and “dateTime” (ISO 8601), cannot be translated appropriately into DMT Admin data types defined in the current specification. Translating these data types might result a limitation of the available value range and a complex object that consists of multiple nodes, respectively.

4.3 Conclusion

The OSGi Mobile specification delivers a standardized data model (the DMT), and standardized interface (on the DMT Admin service) to enable remote management through a protocol adapter. However, the current specification lacks management objects for a number of interesting areas. Also, there is some support lacking for TR-069 notifications. Furthermore, since the DMT model is OMA-DM inspired, implementing a TR-069 protocol adapter is not straightforward, although not impossible.
5 Requirements

REQUIREMENT[5]: Support for notification of parameter value changes is required for both framework and services sub-trees, as well as for bundles implementing a sub-tree of the management tree. Some lightweight mechanism MUST allow identifying which parameters have changed and haven’t already been notified.

REQUIREMENT[6]: The remote management solution MUST provide some mechanism to notify the remote management entities, where the remote manager controls which information to receive.

REQUIREMENT[8]: The solution should specify a guideline of RPC mapping between DMT Admin service interfaces and remote management protocols, such as TR-069.

REQUIREMENT[9]: The solution should be able to handle the data types of “unsignedint” and “dateTime” (ISO 8601) to accommodate TR-069 protocol.

6 Technical Solution

This RFC provides a guideline for RPC mapping between DMT Admin service interfaces and TR-069 protocol, which is defined by Broadband Forum [6]. Since the DMT Admin service focuses on the OMA-DM protocol for remote management, some features, such as RPC mapping and notification mechanism, are inadequate in terms supporting the TR-069 protocol.

This RFC makes the following recommendations to support TR-069.

● Handling rules of the notification mechanism between DMT Admin and TR-069.

● Guideline of RPC mapping between the DMT Admin interfaces and TR-069 protocol

● Management of sessions including read-only and transactional session

● Data type mapping not previously defined in the DMT Admin specification

● Error code mapping and SOAP Fault handling for TR-069 protocol

The basic architecture of DMT Admin service does not need to be modified to support the TR-069 protocol. Therefore, only the specifications and APIs that are related to the above recommendations are included in this document.
6.1 Basic architecture of TR-069 protocol implementation

The basic architecture of a system using TR-069 follows that defined in DMT Admin specification (see Figure 1). In the architecture, there must be a protocol adapter that uses the TR-069 protocol to communicate with remote manager. On the other hand, each data model, which is intended to be managed through the protocol adapter, must be implemented as a data plugin for DMT Admin service.

![Fig.1 Basic Architecture of TR-069 Protocol Implementation](image)

Basically, the TR-069 protocol adapter should call `DmtSession` interface provided by DMT Admin service when the remote manager calls a protocol adapter's RPC specified by TR-069 protocol. Data plugins are used to implement the data models that are to be manipulated via the interfaces defined in the DMT Admin specification. Therefore, a remote manager's operation through the TR-069 protocol can be propagated to the data plugins.

However, there are several differences in the notification mechanism between DMT Admin interfaces and TR-069 RPCs. According to the problem description in Section 4.2.1, DMT Admin specification offers no suitable mechanism with which to handle TR-069 notification. Therefore, the TR-069 protocol implementation should adhere to the following recommendations.

6.2 Notification handling for TR-069

The following definitions are based on the extended DMT Admin specification described in RFC-141, which newly defines a notification property and methods manipulating it. See also RFC-141.

6.2.1 Notification Attribute Definition for TR-069

According to the extended DMT Admin specification, a detailed definition of the notification property (String) should be defined for each protocol such as TR-069. Therefore, this RFC defines the following notification attributes for TR-069 that are kept as the values of notification properties. A protocol adapter and a data plugin, which are implemented based on the TR-069 protocol, are strongly recommended to adhere to this notification attribute definition.
• “0_Notification_Off” - The TR-069 notification is off; this means notifications must not be issued. This notification attribute has been defined in the extended DMT Admin specification as a common definition.
• “1_TR-069_Passive_Notification” - The TR-069 passive notification is available.
• “2_TR-069_Active_Notification” - The TR-069 active notification is available.

6.2.2 Data Plugin Procedures

The implementation of notification function described below is optional for data plugins. However, it is strongly recommended to implement the following guideline for data plugins, because the notification function is really useful in managing CPEs remotely from the TR-069 perspective.

Notification Attribute Management

Data plugins must keep notification attribute persistently as a value of notification property. Since the management of attribute information is the responsibility of the data plugin, it must store notification attributes persistently beyond the restart of bundles that implement the data plugin.

These attributes are specified in two ways; the first one is that the notification attribute has been defined in the data model implemented by the data plugin, and the second one is that the notification attribute is specified by a remote manager through the DMT Admin interface. In that former, the data plugin should set the defined notification attribute for the appropriate node as an initial property value. In the latter, on the other hand, the data plugin should set the node attribute specified by the remote manager for the indicated node's property.

Assume that a method to change a notification attribute of a node is called. If the notification attribute specified in a data model is immutable, or if the specified node is incapable of issuing notifications, the data plugin should throw the DmtException with COMMAND_FAILED code against the setNodeNotification(String[], String). For example, the data model, which is implemented by the data plugin, defines the notification attribute of the indicated node permanently has to be “2_TR-069_Active_Notification”.

If there are multiple principals in one system, different notification attributes for the same node could be specified by the principals. In order to address this situation, a data plugin should keep pairs of notification attribute and principal, so that the appropriate attribute can be seen from each principal. Therefore, getNodeNotification(String, String) should return a different value according to the principal. The principal information can be obtained from DmtSession.getPrincipal() when the protocol adapter creates a TransactionalDataSession, ReadWriteDataSession or ReadableDataSession object. In this case, the notification attribute that has been defined as a data model specification should be automatically applied to all principals as initial values for the Notification property.

Notification Issuing

Values and properties of nodes can be changed either through DMT Admin interfaces or on the level of a data plugin internally. One example of the latter is ServiceState Object; it can be changed not though a DMT Session object. Only in the former case, the DMT Admin will raise events as defined by the DMT Admin Specification.

In the latter case, the data plugin must call the Event Admin API to fire the Event objects with the following specified topics. This topic represents the notification demanded by the TR-069 protocol, and a data plugin must use this topic as a TR-069 notification event.
- org/broadbandforum/TR069/Notification: this event represents that the existing node values were changed where the data model of the node is based on TR-069 protocol.

The Event object must contains the following properties:

- source – (String) the URI of the node which is the source of the notification
- notification – (String) the notification attribute of the node kept in its Notification property
- data – (DmtData) a DmtData object that contains the format and value of the data of the target node
- principals – (String[]) a list of the principals’ names, which are the recipients of the notification

Note: in general, the Event object should contain only String objects and the eight primitive types to assure immutability. Although DmtData is not the one of the specified types, it should be used as a property representing the format and value of the node for the convenience that provides to both data plugins and protocol adapters; DmtData is absolutely immutable as defined in the DMT Admin specification.

When the node value changes internally and has notification attribute other than “0_Notification_Off”, the related data plugin must create an event object for TR-069 notification. The data plugin obtains related information, which will be included in the event properties. The “data” is a DmtData object which includes the changed node value. The “notification” is the notification attribute kept in the Notification property. If multiple principals specify different notification attributes for the node, the data plugin should issue multiple events for each notification attribute type. The “principals” is a list of the recipients of this event, and several principals can be specified in this property if multiple principals have set the same notification attribute for the node.

The wild-card “any” can be used as a value corresponding with “principals” key, if all principals should receive the notification. In this case, the “principals” property must have only one element for its String array value, and String[0] must be “any”.

In order to issue an event, the data plugin can use either the sendEvent method or the postEvent method, which are defined in the Event Admin specification. It is highly recommended for a developer of a data plugin to carefully choose one of them for each node taking into account the features of the methods.

In terms of system performance, the maximum frequency of issuing events from a data plugin for passive notification should not exceed one event per second. On the other hand, no restriction is placed on active notification. A data plugin, therefore, must issue all changes of the node as events, if the notification attribute is specified as active notification.

### 6.2.3 Protocol Adapter Procedures

The following guideline of notification function must be implemented by a protocol adapter based on the TR-069 protocol, because it is possible that this function will be used by a remote manager or data plugins.

**Notification Attribute Configuration**

A protocol adapter must propagate the notification attributes of nodes from TR-069 SetParameterAttributes RPC to DmtSession.setNodeNotification(String, String). See also section 6.4 RPC mapping for TR-069.

If a notification attribute cannot be changed because the feature is not supported by a data plugin, the DmtException will be thrown when a protocol adapter calls the DmtSession.setNodeNotification(String, String).
However, the protocol adapter should expect that all nodes implemented by data plugins support notification function and should try to set notification attributes for the specified nodes.

**Notification event handling**

To implement the notification function in accordance with the architecture described in Section 6.2.2, a protocol adapter must listen to the Events, which are defined in the Event Admin specification, with the topic of “org/broadbandforum/TR069/Notification”. Therefore, the protocol adapter must register an Event Handler service.

The protocol adapter must handle the Events as the following procedures. Note that these logics can be implemented by utilizing the filter function defined in the Event Admin specification.

1. The protocol adapter retrieves the “principals” property to check whether the notification intends to notify the principals who are associated with the protocol adapter. If not, the event is discarded.

2. The protocol adapter retrieves the “notification” property to decide notification pattern.
   a) If the notification attribute is specified as “1_TR-069_Passive_Notification”, the protocol adapter should store the pair of the “source” property as a key and the DmtData object included in the “data” property as a value until next Inform RPC is issued; if there already exists a stored pair with the same key, the stored value will be updated. When the Inform RPC is issued to the principal on behalf of the remote manager, the protocol adapter creates Inform RPC following step 3.
      ▪ The stored “source” properties and the paired DmtData objects should be discarded after the Inform RPC, including the corresponding notifications, is fired to the remote manager.
   b) If the notification attribute is specified as “2_TR-069_Active_Notification”, the protocol adapter must send the Inform RPC immediately to the associated remote manager. Proceed to step 3.
   c) Otherwise, the event should be discarded.

3. The protocol adapter sends the notification using “Inform” RPC according to the TR-069 specification. The Event Struct of Inform RPC must include “4 VALUE CHANGE” as one of the EventCode elements. In addition, the “source” property and the DmtData object must be interpreted as the Name and the Value of the “ParameterList” argument of the Inform RPC, respectively.

---

### 6.2.4 Security

Because the notification is sent as an event, all event handlers indicating “org/broadbandforum/TR069/Notification” as the EVENT_TOPIC service property can receive the data of the notification source as a part of the event. Therefore, the TopicPermission with SUBSCRIBE action must be strictly granted to the only protocol adapters that are allowed to receive the data with the TR-069 notification event. It is highly recommended that the system manager carefully determines which bundles can listen the event.

On the other hand, the data plugins that could issue the TR-069 notification event must have the TopicPermission with PUBLISH action.

In terms of the access control list (ACL) mechanism defined in the DMT Admin specification, the notification event should comply with the access rights defined by the ACL. According to the TR-069 protocol specification, there is no access control for reading node value; this means all node values can be read by all principals that implement the TR-069 protocol. Therefore, the notification event can be sent to all TR-069 protocol adapters without ACL...
verification, because the notification just provides information of the specified node and is a kind of the read operation.

6.3 RPC mapping for TR-069

The following definitions are based on the extended DMT Admin specification described in RFC-141, which includes the notification property and the methods manipulating it. See also RFC-141.

DMT Admin interface is not restricted to any specific protocol but is inspired by the RPCs of OMA-DM, because OMA-DM is a primary remote management protocol for the mobile industry.

In TR-069 specification, there are 17 RPCs for handling client devices, which include basic setter / getter methods and methods modifying tree-structured data. Although these RPCs are very similar to OMA-DM RPCs, there is a slight mismatch between TR-069 RPCs and the methods defined by the DMT Admin specification.

Therefore, mapping rules between them should be defined in order to manipulate data models properly such as the Residential Management Model defined in RFC-140.

This RFC provides guidelines on: how an implementation of TR-069 RPCs protocol adapter should call one or more DMT Admin methods, when it receives TR-069 RPCs from a remote manager.

6.3.1 TR-069 CPE RPCs to DMT Admin Interfaces

Table 4.1 shows the mapping guideline of TR-069 CPE RPCs to DMT Admin interfaces. TR-069 CPE RPCs are the RPCs implemented in the client side program and called by a remote server. TR-069 protocol adapter, therefore, calls DMT Admin service through the DmtSession interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>TR-069 RPCs</th>
<th>DMT Admin methods</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR-069 CPE required RPC</td>
<td>GetRPCMethods</td>
<td>DmtSession.getNodeValue()</td>
<td>This RPC gets the values from the specified nodes. Therefore, DmtSession.getNodeValues() should be called as the corresponding method. When the RPC contains multiple ParameterValueStructs, the method should be called for each name-value pair. Moreover, if the indicated string is given as a partial path name, the method should be called recursively for all the nodes in the subtree contained in the branch of the naming path names. Therefore, DmtSession.getChildNodeNames() should be called as the corresponding method. When the RPC contains multiple strings indicating path names, the method should be called for each path name. Moreover, if the indicated string is given as a partial path name, the method should be called recursively for all the nodes in the subtree contained in the branch of the naming path.</td>
</tr>
<tr>
<td>SetParameterValues</td>
<td>DmtSession.setNodeValue()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>TR-069 RPCs</td>
<td>DMT Admin methods</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>hierarchy that shares the same partial path name as the prefix. In this case, DmtSession.getChildNodeNames() should be used to get names of the children nodes sharing the same partial path. CPE should return the GetParameterValuesResponse, which is the corresponding response of the GetParameterValues, after completion of all method calls for the arguments.</td>
</tr>
<tr>
<td>GetParameterNames</td>
<td>DmtSession.getEffectiveNodeAcl()</td>
<td>DmtSession.getChildNodeNames()</td>
<td>This RPC is used to obtain parameters of the specified path. Therefore, DmtSession.getChildNodeNames() should be called to get the node name as a parameter name. Because the obtained information includes the permission for overwriting the specified node, DmtSession.getEffectiveNodeAcl() should be also used to get the information. These methods should be called recursively for all of the nodes in the sub-tree indicated by the parameter path. CPE should return the GetParameterNamesResponse after completion of all method calls for the arguments.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetParameterAttributes</td>
<td>DmtSession.setNodeAcl()</td>
<td>DmtSession.setNodeNotification()</td>
<td>This RPC is used to set the specified node’s attributes which include notification configuration and ACL configuration. The notification attribute should be set to the DMT Admin through DmtSession.setNodeNotification() method. The Notification parameter in the RPC should be interpreted as the notification attributes defined in section 6.2.1. On the other hand, the ACL configuration should be set to the DMT Admin through DmtSession.setNodeAcl() method. Entities of AccessList parameter of the RPC should be interpreted as a principal defined in the DMT Admin specification. If there is no entity in the AccessList, CPE should consider this situation as only default principal has write permission for the node. These methods should be called recursively for all of the nodes in the sub-tree indicated by the parameter path, if the Name parameter is specified as a partial path ended with a “.” (dot).</td>
</tr>
<tr>
<td>GetParameterAttributes</td>
<td>DmtSession.getEffectiveNodeAcl()</td>
<td>DmtSession.getNodeNotification()</td>
<td>This RPC is to obtain the attributes, which include notification attribute and ACL configuration. The notification attribute should be retrieved by the DmtSession.getNodeNotification() method. The ACL configuration should be derived by the DmtSession.getEffectiveNodeAcl() method. If multiple path names or sub-trees containing several nodes are indicated, the method should be recursively called for each node specified. The protocol adapter should return GetParameterAttributesResponse including the derived ACL configuration and the notification attribute. The notification attribute should be interpreted as Notification parameter in the RPC with the definition described in section 6.2.1.</td>
</tr>
<tr>
<td>Type</td>
<td>TR-069 RPCs</td>
<td>DMT Admin methods</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AddObject</td>
<td>DmtSession.createInteriorNode()</td>
<td>DmtSession.getChildNodeNames()</td>
<td>This RPC is used to create an object sub-tree including interior nodes and leaf nodes, and only interior node indicating object name, which is usually the node locating at the top of sub-tree hierarchy, can be specified. At first, the protocol adapter should call DmtSession.getChildNodeNames() to get instance IDs currently existing to check the instance IDs which is created by the data plugin internally. Second, the protocol adapter chooses one instance ID that has been never used before, which means the instance ID must not be reused for the same object beyond reboots of an OSGi framework. The protocol adapter should then call the DmtSession.createInteriorNode() with the specified path name and the selected instance ID, and the data plugin tries to create the desired object under the indicated instance ID. If the indicated ID is not acceptable for the data plugin, for example the ID is already used by the data plugin internally or the ID is reserved to create instance in the future, the data plugin must throw DmtException with COMMAND_FAILED code to the protocol adapter. If the DmtException is thrown, the protocol adapter should change the instance ID and should retry the creation of the object by calling DmtSession.createInteriorNode() until the prescribed number of times has been exceeded. Finally, the protocol adapter must return the AddObjectResponse with the instance ID to the remote manager if the operation succeeded. [Remarks] A data plugin may give the same instance ID to the same target in order to keep consistency of the node path, if the instance is created by not the operation from the protocol adapter but by the data plugin internally.</td>
</tr>
<tr>
<td>DeleteObject</td>
<td>DmtSession.deleteNode()</td>
<td></td>
<td>This RPC is to delete the specified object including interior and leaf nodes. Therefore, the DmtSession.DeleteNode() should be called with the instance ID of the object, which is specified as the parameter of DeleteObject RPC.</td>
</tr>
<tr>
<td>Reboot</td>
<td>None</td>
<td></td>
<td>There is no recommendation for these RPCs. These RPCs’ action depends on the implementation of the protocol adapter.</td>
</tr>
<tr>
<td>Download</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR-069 CPE</td>
<td>ScheduleInform</td>
<td>None</td>
<td>This RPC is not related to DMT Admin method. Therefore, there is no recommendation for this RPC.</td>
</tr>
<tr>
<td>optional RPC</td>
<td>FactoryReset</td>
<td>None</td>
<td>There is no recommendation for these RPCs. These RPCs’ action depends on the implementation of the protocol adapter.</td>
</tr>
<tr>
<td></td>
<td>GetQueuedTransfers</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetAllQueuedTransfers</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Upload</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetVouchers</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetOptions</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
6.3.2 DMT Admin Interfaces to TR-069 ACS RPCs

Table 4.2 shows the mapping guideline of DMT Admin interfaces to TR-069 ACS RPCs. TR-069 ACS RPCs are the RPCs implemented in the server side program and called by a managed client. Therefore, the TR-069 protocol adapter should call TR-069 ACS RPCs when the DMT Admin calls the TR-069 protocol adapter through the RemoteAlertSender interface.

A protocol adapter must register the RemoteAlertSender service, which is defined in the DMT Admin specification, with the "principals" service property that represents the associated principals of the protocol adapter.

Table 4.2 Mapping of TR-069 ACS RPCs to DMT Admin interfaces

<table>
<thead>
<tr>
<th>Type</th>
<th>DMT Admin methods</th>
<th>TR-069 RPCs</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR-069 ACS required RPC</td>
<td>RemoteAlertSender.sendAlert()</td>
<td>Inform</td>
<td>A protocol adapter must register the RemoteAlertSender service with the &quot;principals&quot; property that represents the associated principals of the protocol adapter. When the RemoteAlertSender.sendAlert() implemented by the protocol adapter is called, the Inform RPC should be fired to the associated ACS. The &quot;code&quot; specified as the parameter of the method should be interpreted as the EventCode included in the EventStruct defined in the TR-069 specification. And the ParameterList that is contained in the Inform RPC should include the ParameterValueStruct consisting of the Name and Value, which should be the &quot;source&quot; and the &quot;data&quot; included in the AlertItem, respectively. If the AlertItem does not contain any &quot;source&quot; as the node that is the cause of the alert, the protocol adapter should discard the AlertItem and does not have to send Inform RPC.</td>
</tr>
<tr>
<td>None</td>
<td>TransferComplete</td>
<td></td>
<td>There is no recommendation for these RPCs. This RPC's action depends on the implementation of the protocol adapter.</td>
</tr>
<tr>
<td>TR-069 ACS optional RPC</td>
<td>None</td>
<td>GetRPCMethods</td>
<td>This RPC is not related to DMT Admin method. Therefore, there is no recommendation for this RPC.</td>
</tr>
<tr>
<td>None</td>
<td>AutonomousTransferComplete</td>
<td></td>
<td>There is no recommendation for these RPCs. These RPCs' action depends on the implementation of the protocol adapter.</td>
</tr>
<tr>
<td>None</td>
<td>RequestDownload</td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>Kicked</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.4 Session management

6.4.1 Opening Session

The DMT Admin specification defines three types of locking modes of sessions for protocol adapters to open a DmtSession for reading or writing node values; LOCK_TYPE_SHARED, LOCK_TYPE_EXCLUSIVE and LOCK_TYPE_ATOMIC, which correspond to the ReadableDataSession, the ReadWriteDataSession and the TransactionalDataSession respectively.
The protocol adapter adaptively chooses either LOCK_TYPE_ATOMIC or LOCK_TYPE_EXCLUSIVE as the lock mode based on the RPC, when it receives the SOAP request from the remote manager. If the received RPC needs to write node values, the protocol adapter must use LOCK_TYPE_ATOMIC session. If there already exists an opened LOCK_TYPE_ATOMIC session, the session will continue to be used. Otherwise, the protocol adapter should open a new session as LOCK_TYPE_ATOMIC. If the received RPC does not need to write node values, the protocol adapter should use LOCK_TYPE_SHARED session. If there already exists an opened LOCK_TYPE_SHARED session, the session will continue to be used. Otherwise, the protocol adapter should open a new session as LOCK_TYPE_SHARED.

On the other hand, the timing of closing session is not defined in this RFC. The close of the session should be conducted in a reasonable way depends on the implementation. In general, the protocol adapter should avoid opening too many sessions concurrently.

Note: in case that the SOAP request requires to read many node values simultaneously in one RPC operation such as GetParameterValues and GetParameterNames, there is risk that the consistency of data may not be kept because of value changes during the operation. Therefore, the protocol adapter is recommended to open the session as LOCK_TYPE_EXCLUSIVE for those RPCs, although these RPCs do not need to write any node value.

Note: if the received RPC needs to write only one node value, the protocol adapter may use LOCK_TYPE_EXCLUSIVE because there is no reason to support transactions.

If a data plugin does not support the TransactionalDataSession but does support the ReadWriteDataSession, the protocol adapter may use the LOCK_TYPE_EXCLUSIVE instead of the LOCK_TYPE_ATOMIC. In this case, the atomicity of the data is not ensured by the DMT Admin service. If a data plugin does not support either the TransactionalDataSession or ReadWriteDataSession, the protocol adapter can not change values for any nodes and should return the corresponding SOAP Fault response to the remote manager.

The protocol adapter may open or close the session every time it receives a SOAP Request or sends the corresponding SOAP Response, respectively.

### 6.4.2 Commit for Atomic Session

The DMT Admin supports the transaction management of the operations conducted through the DmtSession interface. However, there is no explicit RPC in the TR-069 specification to handle atomicity of those operations. Therefore, this RFC recommends the following implementation regarding the management of the transactional session.

Let us assume that the protocol adapter receives the SOAP request from the remote manager and the protocol adapter needs to use a LOCK_TYPE_ATOMIC session as described in Section 6.4.1. In that case, the session should be opened at the start of processing the request that needs to write node values, if the session has not been opened. If the corresponding RPC operation has been completed normally, the protocol adapter should call the DmtSession.commit() method before returning the SOAP response to the remote manager. After calling commit(), it can close the session.

Note: even if the corresponding SOAP response fails to send to the remote manager, the changed data will be stored in the data plugin because the protocol adapter cannot rollback the operation after calling commit().

If the protocol adapter encounters any error conditions during the operation, such as DmtExceptions, it should call the DmtSession.rollback() to restore data consistency and should return SOAP Fault response to the remote manager.
6.5 Mapping of data types

The following definitions are based on the extended DMT Admin specification described in the RFC-141, which introduces new data types: FORMAT_LONG and FORMAT_DATETIME. See also RFC-141.

There 6 data types are defined in the TR-069 specification. Table 4.3 shows the mapping of these data types to DmtData types defined in DMT Admin specification.

Two data types, unsignedInt and dateTime, are missing in the DMT Admin specification as regards accommodating TR-069 protocol. Therefore, these data types should be handled in the following way.

<table>
<thead>
<tr>
<th>Type</th>
<th>TR-069 data type</th>
<th>DMT Admin data type</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>FORMAT_STRING</td>
<td>Created by DmtData(String)</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>FORMAT_INTEGER</td>
<td>Created by DmtData(int)</td>
<td></td>
</tr>
<tr>
<td>unsignedInt</td>
<td>FORMAT_LONG</td>
<td>The unsignedInt is an unsigned integer. Since there is no suitable data type defined in Java, this data type should be mapped as String.</td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td>FORMAT_BOOLEAN</td>
<td>Created by DmtData(boolean)</td>
<td></td>
</tr>
<tr>
<td>dateTime</td>
<td>FORMAT_DATETIME</td>
<td>The dateTime is a String object that is interpreted as an dateTime type defined in ISO 8601; it is used as a value of date and time in TR-069. Since there is no corresponding data type defined in DMT Admin, this</td>
<td></td>
</tr>
</tbody>
</table>
data type should be mapped as String.

| base64     | FORMAT_BASE64 | Created by DmtData(byte[], boolean) |

### 6.6 Error code and SOAP Fault

#### 6.6.1 Error code and Fault code mapping

The DMT Admin specification defines the DmtIllegalStateException and the DmtException containing 17 codes to represent the cause of an error. On the other hand, TR-069 defines 20 fault codes as standard errors in its specification. Therefore, this RFC defines the mapping between these error codes and fault codes.

**Table 4.4 Mapping of Exception code to TR-069 Fault Code**

<table>
<thead>
<tr>
<th>Class</th>
<th>DmtException code</th>
<th>TR-069 Fault code</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DmtException</td>
<td>ALERT_NOT_ROUTED</td>
<td>None</td>
<td>This error never happens against a protocol adapter.</td>
</tr>
<tr>
<td></td>
<td>COMMAND_FAILED</td>
<td>9002</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMMAND_NOT_ALLOWED</td>
<td>9003</td>
<td>For the CWMP fault element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9008</td>
<td>For the SetParameterValuesFault element.</td>
</tr>
<tr>
<td></td>
<td>CONCURRENT_ACCESS</td>
<td>9003</td>
<td>For the CWMP fault element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9008</td>
<td>For the SetParameterValuesFault element.</td>
</tr>
<tr>
<td></td>
<td>DATA_STORE_FAILURE</td>
<td>9002</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FEATURE_NOT_SUPPORTED</td>
<td>9003</td>
<td>Only for the CWMP fault due to the SetParameterValues RPC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9008</td>
<td>For the SetParameterValuesFault element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9009</td>
<td>Only for the CWMP fault due to the SetParameterAttributes RPC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9002</td>
<td>For the CWMP fault due to the other RPCs.</td>
</tr>
<tr>
<td></td>
<td>INVALID_URI</td>
<td>9003</td>
<td>Only for the CWMP fault due to the SetParameterValues RPC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9005</td>
<td>For the SetParameterValuesFault element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9005</td>
<td>For the CWMP fault due to the other RPCs.</td>
</tr>
<tr>
<td></td>
<td>METADATA_MISMATCH</td>
<td>9003</td>
<td>For the CWMP fault element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9007</td>
<td>For the SetParameterValuesFault element.</td>
</tr>
<tr>
<td></td>
<td>NODE_ALREADY_EXISTS</td>
<td>9004</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODE_NOT_FOUND</td>
<td>9003</td>
<td>Only for the CWMP fault due to the</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9005</td>
<td>For the SetParameterValuesFault element.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9005</td>
<td>For the CWMP fault due to the other RPCs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERMISSION_DENIED</td>
<td>Only for the CWMP fault due to the SetParameterValues RPC.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9008</td>
<td>For the SetParameterValuesFault element.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9001</td>
<td>For the CWMP fault due to the other RPCs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REMOTE_ERROR</td>
<td>None This error never happens against a protocol adapter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROLLBACK_FAILED</td>
<td>9002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SESSION_CREATION_TIMEOUT</td>
<td>9002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSACTION_ERROR</td>
<td>9003 For the CWMP fault element.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9008</td>
<td>For the SetParameterValuesFault element.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNAUTHORIZED</td>
<td>9001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>URI_TOO_LONG</td>
<td>9003 Only for the CWMP fault due to the SetParameterValues RPC.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9005</td>
<td>For the SetParameterValuesFault element.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9005</td>
<td>For the CWMP fault due to the other RPCs.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.6.2 SOAP Fault expression

In this section, the expression of the SOAP Fault response is defined based on the error code and fault code mapping.

If the SOAP fault does not contain SetParameterValuesFault, the protocol adapter must set a code and a message contained in a DmtException instance to a FaultCode element and a FaultString element in the SOAP Fault response, respectively. The code in the DmtException must change the Fault code according to Table 4.4. Other information such as a path and causes included in the DmtException instance may be added to the FaultString element.

If the SOAP fault is raised by the SetParameterValues and needs to include SetParameterValuesFault element, the protocol adapter must set path, code and message contained in a DmtException instance to ParameterName, FaultCode element and FaultString element in the SetParameterValuesFault element, respectively. The code in the DmtException must change the Fault code according to Table 4.4. If there are multiple paths in the DmtException, each path must be described as one SetParameterValuesFault element.
Other information such as cause information included in the DmtException instance may be added to the FaultString element included in the SetParameterValuesFalt element.

6.7 Limitation of notification

The recommendations described in this RFC have limitations on notification, which depend on the OSGi Framework and services. Since several features of an OSGi Framework and standard services do not support eventing mechanism for value/state changing as described in Section 4.2.1, some data plugins corresponding with the features cannot implement notification mechanism even though their notification properties set to issue notifications. This limitation depends on features specified in the OSGi specification.

7 Considered Alternatives

7.1 The reason for using event instead of the NotificationService

There is an alternative to sending a notification from a data plugin to a protocol adapter; the NotificationService and the RemoteAlertSender are defined in the DMT Admin specification for this purpose. The problem is that these interfaces are able to have only one principal for the argument of the method sending notification. A data plugin, therefore, has to repeatedly call this method for each principal who is a recipient of the notification, and the implementation might become complicated to assure atomicity of the notification. On the other hand, the Event Admin API is designed to send an event to many recipients synchronously, so the data plugin easily supports the atomicity of notification if there are many listeners.

For the reason described above, this RFC uses the Event Admin model to send a notification.

7.2 The reason to define the new event type for notification

The DmtEvent defined in the DMT Admin specification does not include DmtData as a property of the event. The reason is that it is difficult to assure the ACL constraints for every DmtEvent; the recipient of the event can get the node value, although it is not allowed to read the node value by the ACL.

In terms of atomic operation to issue notification, the changed node value should be included in the event notifying the protocol adapter, because the node value would be changed again after a data plugin issued an event sending notification.

Based on these considerations, this RFC defines the new event type called "org/broadbandforum/TR069/Notification" to send TR-069 notification event, and the DmtData is contained in the event. Therefore, the protocol adapter is able to get node values synchronously together with the notifications and is able to send them without loss of node value. On the other hand, the security of node values should be strictly controlled by the TopicPermission defined in the Event Admin specification.
8 Security Considerations

All security requirements follow the DMT Admin specification. Security guidelines for the notification mechanism are described in this RFC. See also Section 6.2.4 security.

9 Document Support

9.1 References


[3]. OMA, Open Mobile Alliance. The mission of the Open Mobile Alliance is to facilitate global user adoption of mobile data services by specifying market driven mobile service enablers that ensure service interoperability across devices, geographies, service providers, operators, and networks, while allowing businesses to compete through innovation and differentiation. http://www.openmobilealliance.org/

[4]. OMA Device Management specification v1.2. The goal of the Device Management Working Group is to specify protocols and mechanisms that achieve management of mobile devices including the necessary configuration to access services and management of the software on mobile devices. http://www.openmobilealliance.org/release_program/dm_v1_2C.html


[6]. The Broadband Forum is a global consortium of nearly 200 leading industry players covering telecommunications, equipment, computing, networking and service provider companies. Established in 1994, originally as the ADSL Forum and later the DSL Forum, the Broadband Forum continues its drive for a global mass market for broadband, to deliver the benefits of this technology to end users around the world over existing copper telephone wire infrastructures. http://www.broadband-forum.org/about/forumhistory.php

[7]. Data model template for TR-069 enabled devices, TR-106 amendment 1, November 2006
9.2 Author's Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Koya Mori</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81-46-859-3446</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:mori.kouya@lab.ntt.co.jp">mori.kouya@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Ikuo Yamasaki</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>NTT Corporation</td>
</tr>
<tr>
<td>Address</td>
<td>Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan</td>
</tr>
<tr>
<td>Voice</td>
<td>+81-46-859-8537</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:yamasaki.ikuo@lab.ntt.co.jp">yamasaki.ikuo@lab.ntt.co.jp</a></td>
</tr>
</tbody>
</table>

9.3 Acronyms and Abbreviations

9.4 End of Document
The OSGi Alliance and its members specify, create, advance, and promote wide industry adoption of an open delivery and management platform for application services in home, commercial buildings, automotive and industrial environments. The OSGi Alliance serves as the focal point for a collaborative ecosystem of service providers, developers, manufacturers, and consumers. The OSGi specifications define a standardized, component oriented, computing environment for networked services. OSGi technology is currently being delivered in products and services shipping from several Fortune 100 companies. The OSGi Alliance's horizontal software integration platform is ideal for both vertical and cross-industry business models within home, vehicle, mobile and industrial environments. As an independent non-profit corporation, the OSGi Alliance also provides for the fair and uniform creation and distribution of relevant intellectual property – including specifications, reference implementations, and test suites – to all its members.

HOW TO REACH US:

OSGi Alliance
Bishop Ranch 6
2400 Camino Ramon, Suite 375
San Ramon, CA 94583 USA

Phone: +1.925.275.6625
E-mail: marketinginfo@osgi.org
Web: http://www.osgi.org

OSGi is a trademark of the OSGi Alliance in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

All other marks are trademarks of their respective companies.