DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

DATE OF DISTRIBUTION: 10 June 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.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

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 1 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 4.1 specification documents or propose new specifications to potentially be incorporated in the final OSGi Service Platform Release 4 Version 4.2 specification 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 4.1 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 4.1 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 1

Revision 1.0
10 June 2009
**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.
1.2 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>
</table>
| Initial  | Sep. 12 2008 | Initial Draft  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp |
| 0.2      | Oct. 9 2008  | 2nd Draft  
Overall architecture is changed based on REG meeting and RFC-139 definition.  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp |
| 0.3      | Nov. 10 2008 | 3rd Draft  
Definition tables for all nodes in Management Objects are added.  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp |
| 0.4      | Jan. 19 2009 | 4th Draft  
Revising tree architecture based on the discussion at the REG meeting.  
Filter format description is added.  
Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp  
Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp |
| 0.5      | Feb. 12 2009 | 5th Draft  
Revising Bundles Object 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 |
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 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.

4.2.4 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.

REQUIREMENT[4]: The management tree SHOULD cover all services defined in the release 4 of the OSGi specifications. If prioritization is needed, at least the following services MUST be covered: Start Level, Permission Admin, Conditional Permission Admin, Configuration Admin and Log Service.

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 have not already been notified.

REQUIREMENT[7]: The solution MUST have a good performance in order to run on devices with limited resources.

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

REQUIREMENT[10]: The solution SHOULD enable bundles running on an OSGi framework to manage Internet Gateway functions of a Home Gateway by using an integrated management tree.

REQUIREMENT[11]: The solution SHOULD enable bundles to hook into Internet Gateway functions such as DHCP.
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, Bundles, PackageState, ServiceState and Filters.

The Framework sub-tree is to manipulate the OSGi framework on which this management tree is implemented. The Bundles sub-tree is used to derive information of individual bundles and to control the life cycle of installed bundles instead of the Deployment sub-tree defined in the Mobile Specification. 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.

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 and Framework Lifecycle control.

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.

![Fig.2 Framework Object](image)

The Framework Object consists of 4 parts; StartLevel, InstallBundle, Lifecycle and Ext. These sub-trees represent individual functions which 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 InstallBundle sub-tree is used to install bundles into the OSGi framework. This sub-tree supports the simultaneous installation of multiple bundles. The Lifecycle sub-tree controls the state of the OSGi framework.

---

Copyright © NTT Corporation 2009 All Rights Reserved
itself, but the detail behavior corresponding to these commands depends on the implementation of the OSGi framework. The Ext sub-tree is to extend 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 life-cycle commands of a bundle except the “Install” method are located in the Bundles object, since the tree architecture has to match the OSGi framework API. In the OSGi framework API, the “Install” method is defined in org.osgi.framework.BundleContext interface while other life-cycle commands are defined in org.osgi.framework.Bundle interface. Therefore, the “Install” command is located in the Framework object, and the other commands are located in the Bundles object, see the following section, of the Residential Management Tree.

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 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 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, and the management system can't specify the identifier of the new node.

All nodes for the Framework Object sub-tree are explained in Table 3.1.

Table 3.1 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>node</td>
<td>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>node</td>
<td>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 node used to configure the Framework's StartLevel. When this node value is replaced, the B u n d l e s s u b - t r e e s t a r t s. StartLevel#setStartLevel with the specified value must be called. This value must be kept persistently.</td>
</tr>
<tr>
<td>Framework/StartLevel/ActiveStartLevel</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>int</td>
<td>1</td>
<td>P</td>
<td>A leaf node that contains the Framework's StartLevel at the moment. This node is read-only to get the Framework's StartLevel.</td>
</tr>
<tr>
<td>Framework/StartLevel/InitialBundleStartLevel</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>int</td>
<td>1</td>
<td>P</td>
<td>A node used to configure the initial bundle StartLevel. When this node value is replaced of</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the Bundles sub-tree starts, StartLevel#setInitialStartLevel with the specified value must be called.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>X</td>
<td>node 0..*</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.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/InstallBundle/&lt;id&gt;/Location</td>
<td>X</td>
<td>X</td>
<td>str 1</td>
<td>A</td>
<td>A node used to set the bundle location under which the specified bundle is installed. This node is writable because the node is used to install a bundle. If the Location is replaced and the URL is not replaced when a transaction is committed, BundleContext#installBundle(LOCATION) must be called where location is the specified value in this node. When the transaction is committed where both this node Location and URL are replaced in the transaction, BundleContext#installBundle(location, in) must be called where location is the specified value of this node and in is an InputStream retrieved from the specified URL. If the installation of a bundle succeeds, the Framework Object must delete all nodes under the Framework/&lt;id&gt;/InstallBundle/&lt;id&gt; node corresponding to the bundle. If the installation of a bundle fails, the Framework Object will NOT retry the operation and instead create an Error node to represent the reason for operation failure.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/InstallBundle/&lt;id&gt;/URL</td>
<td>X</td>
<td>X</td>
<td>str 1</td>
<td>A</td>
<td>A 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. The default value of this node is an empty string. See details of the installation operation described in the Location definition.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework/InstallBundle/&lt;id&gt;/Error</td>
<td>X</td>
<td></td>
<td>str 0,1</td>
<td>A</td>
<td>A node used to represent the reason for an installation failure. When the installation of a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 6.3 Bundles Object

The **Bundles Object** is a managed object that allows the manipulating an individual bundle. The following features are able to be controlled through this object.

- Bundle Information Monitoring (Bundle Location, SymbolicName, Version, Manifest, Signer etc.)
- Bundle State Monitoring (Bundle State, Fragment, Required etc.)
- Bundle Lifecycle Control (Start, Stop, Uninstall, Update, Resolve)
- Bundle Start Level Control (Bundle Start Level)

The **Bundles Object** must support read-write data sessions, but does not have to support transactions.

The tree structure of the **Bundles Object** can be accessed from the `$/OSGi/<instance_id>/Bundles` sub-tree. Figure 3 depicts the top level structure of the **Bundles Object** sub-tree.

<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/Lifecycle</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bundle fails, the Framework Object should create this node and set a value specifying the error reason.</td>
</tr>
<tr>
<td>Framework/Lifecycle/Restart</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A 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>
</tr>
<tr>
<td>Framework/Lifecycle/Shutdown</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A 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>
</tr>
<tr>
<td>Framework/Lifecycle/Update</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A 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>
</tr>
<tr>
<td>Framework/Ext</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interior node that can contain values for user extensions.</td>
</tr>
</tbody>
</table>
Nodes representing each bundle are located under the $/OSGi/<instance_id>/Bundles node to manipulate individual bundles. When a bundle is installed on the OSGi framework, a new node of $/OSGi/<instance_id>/Bundles/<bundle_id> is automatically added and <bundle_id> is incremented. In other words, <bundle_id> is equivalent to Bundle#getBundleId(). The details of the each sub-tree, Resources, Lifecycle, BundleState and BundleExt, are described in the Table 3.2.

All nodes for the Bundles Object sub-tree are explained in Table 3.2.

### Table 3.2 Bundles Object Top-Level 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>Bundles</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>P</td>
<td>Bundles Root node that is a management object containing instances representing individual bundle information. The children of this node must represent the actual bundle status when this sub-tree is accessed.</td>
</tr>
<tr>
<td>Bundles/&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 number must equal the bundle id, which Bundle#getBundleId() returns.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/Resources</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 resources included in the bundle. This sub-tree is detailed later.</td>
</tr>
<tr>
<td>Bundles/&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>Bundles/&lt;bundle_id&gt;/BundleState</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 status information of the bundle. This sub-tree is detailed later.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleExt</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>A</td>
<td>Interior node that can contain values for user-defined parameters.</td>
</tr>
</tbody>
</table>
6.3.1 Resources sub-tree

The Resources sub-tree is used to derive resources in the bundle jar file. The Resources 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, 

`$OSGi/<instance_id>/Bundles/<bundle_id>/Resources/META-INF/Manifest.mf`

nodes are automatically created by the Bundles Object, which implements the Resource sub-tree. Therefore, a remote manager simply gets the node value of 

`$OSGi/<instance_id>/Bundles/<bundle_id>/Resources/META-INF/Manifest.mf`

The <directory> node is automatically created and represents a directory included in a jar file. The Resources 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 Bundles Object implementation, the Bundles Object should abort the reading of file content and should throw a DmtException.
Because the Resources 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 can not 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 \$/OSGi/\<instance_id>/Bundles/\<bundle_id>/Resources node or ancestor nodes of it, heavy data transaction between the remote manager and client would occur because the Resources 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 Resources sub-tree architecture is shown in section 7.

All nodes for the Resources sub-tree are explained in Table 3.3.

<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>Bundles/&lt;bundle_id&gt;/Resources/&lt;directory&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A Represents a directory name. This node is automatically created to show a file-path in a bundle jar-file.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/Resources/&lt;file_name&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>b64</td>
<td></td>
<td></td>
<td></td>
<td>A 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. The node value is encoded in base 64 format.</td>
</tr>
</tbody>
</table>

### 6.3.2 Lifecycle sub-tree

Fig.5 Lifecycle sub-tree

---

Copyright © NTT Corporation 2009  All Rights Reserved
The Lifecycle sub-tree provides the bundle life cycle operations for an individual bundle that is defined in OSGi specification after installation. The operation is basically conducted by indicating the desired state of the bundle; Active, Resolved or Uninstalled. Note that Update is indicated directly as a command because this kind of operation doesn't represent the specific status 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 for this bundle.

All nodes for the Lifecycle sub-tree are explained in Table 3.4.

Table 3.4 Bundles Object, Lifecycle sub-tree

<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>Bundles/&lt;bundle_id&gt;/Lifecycle/DesiredState</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>int</td>
<td>1</td>
<td>A</td>
<td>A node used to control the bundle's life-cycle. When this node value is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>replaced with the state described below or the Bundles sub-tree starts,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>the Bundles sub-tree must change the bundle life-cycle to the specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bundle state. The Bundles sub-tree must return an error, if the specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state can not be understood. This state must be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 – Uninstalled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4 – Resolved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32 – Active</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The Bundles Object does not have to retry the operation if life-cycle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>control fails, but the error status should be written in the OperationResult.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This value must be kept persistently.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/Lifecycle/Update</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td>A</td>
<td>A node used to update the bundle. This node is writable to set the update</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>command. When this node value is replaced with an URL string, Bundle#</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>update(InputStream) must be called where InputStream is the specified URL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If the specified URL is an empty string, Bundle#update() must be called</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>so that the specified bundle is updated with the jar-file indicated by the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BundleLocation. If the update fails, the Bundle object does not have to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>retry the operation.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/Lifecycle/OperationResult</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td>A</td>
<td>This leaf node holds the latest operation’s result which is conducted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>through the Bundles sub-tree, so that the Management System can derive the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>result of the bundle life-cycle operation. If the operation succeeds, “</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Success: ” must be added as a prefix to the value string. On the other</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hand, “Fail: ” must be added as the prefix of this string, if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>operation fails.</td>
</tr>
</tbody>
</table>
6.3.3 BundleState sub-tree

Figure 6 shows the BundleState sub-tree architecture. This sub-tree is used to obtain the information of a bundle from the OSGi framework. 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. StartLevel shows the bundle's start level on this OSGi framework and can be manipulated by a remote manager. State shows the bundle state; Installed, Resolved, Starting, Active, Stopping and Uninstalled. Fragments, Hosts, Required, Requiring and Signers sub-trees contain the corresponding bundle identifiers.

The BundleState sub-tree 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 sub-tree of a bundle must remain available while the OSGi Framework continues to run.

All nodes for the BundleState sub-tree are explained in Table 3.5.

Table 3.5 Bundles Object, BundleState sub-tree

<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>Bundles/&lt;bundle_id&gt;/</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The Bundle-SymbolicName of the bundle.</td>
</tr>
<tr>
<td>BundleState/SymbolicName</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The version of the bundle.</td>
</tr>
<tr>
<td>BundleState/Version</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td>A</td>
<td></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>Bundles/&lt;bundle_id&gt;/BundleState/BundleType</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>int 1</td>
<td>A</td>
<td></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>Bundles/&lt;bundle_id&gt;/BundleState/Status</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>A</td>
<td></td>
<td>This node is the parent of the nodes that represent the status of the bundle.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/Status/Location</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str 1</td>
<td>A</td>
<td></td>
<td>The BundleLocation of the bundle.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/Status/State</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>int 1</td>
<td>A</td>
<td></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>Bundles/&lt;bundle_id&gt;/BundleState/Status/StartLevel</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>int 1</td>
<td>A</td>
<td></td>
<td>The StartLevel of the bundle. If this value is changed, StartLevel#setBundleStartLevel with the specified value must be called.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/Status/PersistentlyStarted</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bool 1</td>
<td>A</td>
<td></td>
<td>The status of the bundle at the last shutdown of the OSGi Framework. If the bundle is Active when the OSGi Framework shutsdons, the value must be TRUE. Otherwise, the value must be FALSE.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/Status/LastModified</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>date 1</td>
<td>A</td>
<td></td>
<td>The latest time at which the bundle has been modified.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/Host</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>A</td>
<td></td>
<td>This node is the parent of nodes that specify the bundle hosting this bundle.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/Host/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str 0,1</td>
<td>A</td>
<td></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>Bundles/&lt;bundle_id&gt;/BundleState/Fragments</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node 1</td>
<td>A</td>
<td></td>
<td>This node is the parent of nodes that specify the fragment bundles of this bundle.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/Fragments/&lt;bundle_id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>str 0..*</td>
<td>A</td>
<td></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.</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>Bundles/&lt;bundle_id&gt;/BundleState/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>Bundles/&lt;bundle_id&gt;/BundleState/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>Bundles/&lt;bundle_id&gt;/BundleState/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>Bundles/&lt;bundle_id&gt;/BundleState/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>Bundles/&lt;bundle_id&gt;/BundleState/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>Bundles/&lt;bundle_id&gt;/BundleState/Signers/&lt;id&gt;</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>0..*</td>
<td>A</td>
<td>Interim node that represents number of signs.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/Signers/&lt;id&gt;/Signer</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 chain. The last item is the root.</td>
</tr>
<tr>
<td>Bundles/&lt;bundle_id&gt;/BundleState/BundleStateExt</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td>A</td>
<td>Interim node that can contain values for user extension.</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.

![Fig.7 PackageState Object](image)

Figure 7 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.

All nodes for the PackageState Object sub-tree are explained in Table 3.6.

<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</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>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>

children of this node must represent the actual package status when this sub-tree is accessed.
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.

Figure 8 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.

All nodes for the ServiceState Object sub-tree are explained in Table 3.7.

**Table 3.7  ServiceState 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>ServiceState</td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td></td>
<td></td>
<td>P</td>
<td></td>
<td>ServiceState Root node containing package information existing on the OSGi Framework. The children of this node must represent the actual service status when this sub-tree is accessed.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;</td>
<td></td>
<td></td>
<td>node</td>
<td>0..*</td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>A node that represents the Service instance. This number must match the service id of the OSGi Framework.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties</td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>Holds the key nodes that contain the values for the service properties.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;</td>
<td></td>
<td></td>
<td>node</td>
<td>0..*</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>A node with the name of a key. The node holds the value of an entry in the service property Dictionary. The value is defined by its type, cardinality and value sub-nodes.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;/Type</td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>Type of the property. See Service Property Dictionary nodes section.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;/Cardinality</td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>Cardinality of the property. See Service Property Dictionary nodes section. The value is either: scalar, vector or array.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;/Values</td>
<td></td>
<td></td>
<td>node</td>
<td>0,1</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>Interior node that contains the actual values. Children are leaf nodes.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/Properties/&lt;key&gt;/Values/&lt;n&gt;</td>
<td></td>
<td></td>
<td>str</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A leaf node that contains a value which is defined by the &lt;key&gt;/Type node in this sub-tree. The name must be an integer that is continuous for the parent tree.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/RegisteringBundle</td>
<td></td>
<td></td>
<td>node</td>
<td>1</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>This node is the parent of nodes that specify the bundle registering the service.</td>
</tr>
<tr>
<td>ServiceState/&lt;service_id&gt;/RegisteringBundle/&lt;bundle_id&gt;</td>
<td></td>
<td></td>
<td>str</td>
<td>1</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>A leaf node that represents bundle ID of the registering bundle. This node does not have any value, so the default value should be an empty string.</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.

![Fig.9 Filters Object](image)

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 $/OSGi/Filter/<search_id>/TargetSubtree$ 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 $/OSGi$ 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 3.8.

### Table 3.8 Filters Object 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>Filters</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>1</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>node</td>
<td>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 <code>&lt;search_id&gt;</code> is assigned in ascending order to Filters instances when they are set. This is 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>
</table>

| Filters/<search_id>/TargetSubtree | X | X | str | 1 | A | 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. |

| Filters/<search_id>/Filter | X | X | str | 1 | A | 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. |

| Filters/<search_id>/Result | X | node | 1 | A | 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. |

### 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

./Bundles

./BundleState
  ./SymbolicName = org.osgi.service.log
  ./Resources
  ./Lifecycle
  ./BundleExt

./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
      .ServiceState
        .11
         .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
      Bundles
        .1
          BundleState
            BundleType = 1
        .Resources
          .Lifecycle
        .BundleExt
      .2
        Bundles
          .15
            BundleState
              BundleType = 1
            .Resources
              .Lifecycle
            .BundleExt
```

### 6.7 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.

### 6.8 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.

6.9 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.

6.10 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.

6.10.1 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.

6.10.2 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/Lifecycle/*.

6.10.3 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 subtree, 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 Resources sub-tree architecture

There is considerable alternative for designing the Resources sub-tree to retrieve resources inside the bundle jar file. The following architecture is another design of the Resources 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 6.3.1.

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 6.3.1 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 6.3.1 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>/Bundles/<bundle_id>/Resources 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
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 ............................................................................................................................... 3

2 Application Domain .................................................................................................................. 4

3 Problem Description ................................................................................................................... 5

4 Requirements ............................................................................................................................. 6
  4.1 Functional Requirements ................................................................................................. 6
  4.2 Non-Functional Requirements ......................................................................................... 7

5 Technical Solution ....................................................................................................................... 7
  5.1 Scope .................................................................................................................................. 7
  5.2 General Considerations .................................................................................................... 7
  5.3 Roles and Functional Blocks ........................................................................................... 9
  5.1 Home Gateway Admin ....................................................................................................... 10
     5.1.1 DMT Admin restrictions .......................................................................................... 10
     5.1.2 Managing and Mapping the DMT Admin Sub-Tree ................................................ 10
     5.1.3 Mapping Rules Examples ....................................................................................... 13
     5.1.4 Raising Events ........................................................................................................ 13
     5.1.5 Event Mapping Examples ....................................................................................... 15
     5.1.6 Remote Management Events and Notifications ...................................................... 15
  5.2 Mapping rules to DMT Admin Management Tree ............................................................... 15
     5.2.1 Mapping from Broadband Forum TR-069 and TR-098 to DMT ......................... 16
     5.2.2 Mapping from UPnP IGD 1.0 .................................................................................. 16
     5.2.3 Examples ................................................................................................................ 17

6 Considered Alternatives .............................................................................................................. 17
  6.1 Not using DMT Admin ...................................................................................................... 17

7 Security Considerations ............................................................................................................ 17

8 Document Support ....................................................................................................................... 18
  8.1 References ....................................................................................................................... 18
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 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. Clarified session handling. Clarified event handling. Changed security 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 session.id property. 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 Home Gateway Device. This service enables OSGi services running on an IGD to access the core level functions of a home gateway, 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 a home gateway (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 a Internet router, 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 OSGi 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 Device 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 a 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 functionalities 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.

Illustration 1: Architecture Sketch

### 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 Device 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 Device Admin SHOULD be able to reject certain requests.

• The specification of the Home Gateway Device 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 Home Gateway device drivers MUST be controlled.

• The Home Gateway Device 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 home gateway 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 (IGD) 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 home gateway (HG) management functions are tightly coupled with the HG operating system and HG 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 HG has a standardized process to find and install additional management bundles (HG Module Driver) as needed. For example, an additional IEEE 802.11n Wi-Fi access point might be plugged into the HG, 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 all management aspects of a device. Implementations of the HG Admin and Vendor HG Drivers have to use the DMT Admin service to offer a configurations-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.

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

Generic mapping rules are defined in chapter 5.2 to map IGD state variables, actions, and functions to a well defined DMT configuration sub-tree. The root of this sub-tree is "./InternetGatewayDevice".

Events that arise from the HG 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 home gateway, 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 home gateway and provides these home gateways to his customers. He orders home gateways 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 **Home Gateway Core Functions** are usually provided by the Home Gateway Vendor. The Home Gateway Core Functions layer contains functionality for all the basic aspects of a Home Gateway, such as configuration 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 **Home Gateway Admin**. As explained before, the **Device Access** service is optional. An Operator could define other means to install Home Gateway driver bundles.

The **Management Stack** contains the necessary services for managing a home gateway. The **Home Gateway Base Driver** implements the necessary base functions to access and manage the **Home Gateway Core Functions**.
Additional functions and/or drivers that can be installed if necessary and on-demand are represented in the Additional Home Gateway Services block. The Home Gateway 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 home gateway service 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 Home Gateway, a web user interface for the customer, or an implementation of the UPnP Home Gateway service.

5.1 Home Gateway Admin

This section specifies the operation of the HG Admin service.

5.1.1 DMT Admin restrictions

Using the DMT Admin to implement the HG Admin is a mandatory requirement because it allows for a loose coupling between the Management Services and the HG Drivers. To understand the function and work of the HG Admin 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 sub-trees of configuration data cannot overlap. A DMT Admin Data Plugin that was registered to manage 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".

A bundle provider can of course register the Data Plugin so that that it just registers itself somewhere in the DMT Admin's configuration tree. But in this case, any Management Service that likes to set or access configuration values to or from that bundle does not know the value's path in the configuration tree.

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 a home gateway the connection on the WAN interface was established or disconnected. This event arises in the Home Gateways 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 the DMT Admin Sub-Tree

The solution for the non-overlapping sub-trees problem described above is to let the HG Admin handle a mapping between the actual sub-tree under the home gateway configuration in the DMT Admin tree and the registered sub-tree of any Data Plugin of driver bundles. For this, the following procedure apply:

- The HG Admin registers one Data Plugin that registers itself with the DMT Admin to manage the sub-tree "./InternetGatewayDevice". The Data Plugin service must be registered with the property dataRootURIs set to "./InternetGatewayDevice". All configuration requests from management services to home gateway related configurations are received by this Data Plugin.

- The HG Admin tracks Data Plugin services that have the properties dataRootURIs and hgConfigurationPath set.

- A bundle that implements driver functionality for the home gateway and needs to be managed (be it a base driver or an additional driver) must register a Data Plugin with the property dataRootURIs set to any
value other than "./InternetGatewayDevice/...". This configuration path must not overlap with any previous registered Data Plugin. In addition it also sets the following properties:

- **hgConfigurationPath**: this property defines the Data Plugin's intended position under the "./InternetGatewayDevice" configuration sub-tree. The value for that property must not start or end with a slash (character /).
  
  Here, overlapping paths are allowed. The resolution is handled by the HG Admin, depending on the value of the **hgConfigurationMultiple** property (s. below).
  
  This property is mandatory. There is no default.

- **hgConfigurationMultiple**: this property defines whether there could be multiple instances for the given configuration path and objects.

  An example is if a home gateway 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 Data Plugins would be used for the configuration of the driver.

  It is the responsibility of the HG 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 HG Vendor Data Plugin and stored persistently by the HG Admin. This means that a once assigned identifier is always assigned to the same HG Vendor 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 HG 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, i.e. it is ignored.

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

- The HG Admin receives tracker callbacks for registered (and unregistered) Data Plugin services. It is responsible to map the "./InternetGatewayDevice" configuration paths to the actual Data Plugins that handle the sub-trees accordingly.

- Configuration requests from Management Applications are done through the DMT Admin. The DMT Admin will request a DMT Session object from the HG Admin's Data Plugin. 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 Vendor's Data Plugins. When closing, committing or rolling back sessions the HG Admin's DMT Session is responsible for performing the according actions on these session objects.

- A bundle that wants to access and manage aspects of the Home Gateway must only use the "./InternetGatewayDevice" sub-tree of the DMT.

The following illustration 4 presents the general relationships between the involved components.

---

1 The names of the properties defined here might change when a more generalized mechanism to circumvented the problems with the DMT Admin is specified.
The creation and usage of Session objects are presented in Illustration 5.

Illustration 4: Data Plugins Relationships

Illustration 5: Use of Session Objects for the HG Admin

It is strongly recommended, though not mandatory by this specification, to implement all Data Sessions as Transactional Data Session (see DMT Admin Service Specification, chapter Data Sessions). In case of transactions the HG 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 HG 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.

The HG Admin itself should, but is not restricted to, register only one Data Plugin service.

### 5.1.3 Mapping Rules Examples

The following table shows examples which reflect the mapping rules defined in the previous section.

<table>
<thead>
<tr>
<th>Vendor DataPlugin</th>
<th>Resulting Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>./vendor_x/firewall</code></td>
<td><code>service/firewall</code></td>
</tr>
<tr>
<td><code>./vendor_x/interface/WAN</code></td>
<td><code>interface/WAN</code></td>
</tr>
</tbody>
</table>

*Table 1: Mapping Rules Examples*

### 5.1.4 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 HG Admin to raise events.

The Vendor HG Driver's Data Plugin must raise events as follows:

- If a value in a driver is 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.
  - `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 driver's Data Plugin was registered with (the `dataRootURIs` property).

\(^2\) An example for an ADDED event could be when a DHCP server in the IGD registers a new client's MAC address.
The newnodes property is not set.

The HG Admin must handle events as follows:

- The HG 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, 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 "./InternetGatewayDevice" sub-tree of the HG Admin. The following rules apply here:
    - If the hgConfigurationMultiple property for the originating Data Plugin is false:
      - The prefix of the node value that matches the value of dataRootURLs is removed. Take R as the remaining value.
      - The new value is constructed as follows: 
        
        
        
        ./InternetGatewayDevice/ + <according value of hgConfigurationPath for that Data Plugin> + <remaining value R>
    - If the hgConfigurationMultiple property for the originated Data Plugin is true:
      - The prefix of the node value that matches the value of dataRootURLs is removed. The remaining value is R.
      - The new value is constructed as follows:
        
        
        
        ./InternetGatewayDevice/ + <according value of hgConfigurationPath 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 HG Driver and one by the HG Admin. Illustration 6 presents the general relationships between the involved components.
5.1.5 Event Mapping Examples

The following table shows examples which reflect the mapping rules defined in the previous section. It is assumed that the vendor's 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>
</tr>
</thead>
<tbody>
<tr>
<td>./vendor_x/firewall/enabled</td>
<td>./InternetGatewayDevice/service/firewall/enabled</td>
</tr>
<tr>
<td>./vendor_x/interface/WAN/disconnected</td>
<td>./InternetGatewayDevice/interface/WAN/1234/disconnected</td>
</tr>
</tbody>
</table>

Table 2: Event Mapping Examples

5.1.6 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.

5.2 Mapping rules to DMT Admin Management Tree

This chapter defines mapping rules for mapping various management standards to the DMT Admin tree.
5.2.1 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.

<table>
<thead>
<tr>
<th>TR-069 data type</th>
<th>DMT Admin data type</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>FORMAT_STRING</td>
<td>Created by DmtData(String)</td>
</tr>
<tr>
<td>int</td>
<td>FORMAT_INTEGER</td>
<td>Created by DmtData(int)</td>
</tr>
<tr>
<td>unsignedInt</td>
<td>FORMAT_INTEGER</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 Integer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Attention: There might be a loss in precision.</td>
</tr>
<tr>
<td>boolean</td>
<td>FORMAT_BOOLEAN</td>
<td>Created by DmtData(boolean)</td>
</tr>
<tr>
<td>dateTime</td>
<td>FORMAT_STRING</td>
<td>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.</td>
</tr>
<tr>
<td>base64</td>
<td>FORMAT_BASE64</td>
<td>Created by DmtData(byte[], boolean)</td>
</tr>
</tbody>
</table>

Table 3: Mapping of data types between TR-069 and DMT Admin
5.2.2 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.

5.2.3 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 HG 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:
• The "./InternetGatewayDevice" subtree of the DMT Admin tree. Access to this sub-tree must be handled by using Dmt Principal Permission, DMT Permission and the appropriate permission actions. Access to the sub-tree 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 sub-trees 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 HG Admin bundle should have access to the values of the vendor's sub-tree.

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

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

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

Events raised by the HG Admin or vendor-specific HG 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
[12]. OSGi Service Platform Core Specification, Permission Admin Service, Chapter 10
## 8.1 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Vivien Helmut, Andreas Kraft, Andreas Sayegh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Deutsche Telekom AG, Laboratories</td>
</tr>
<tr>
<td>Address</td>
<td>Ernst-Reuter-Platz 7, D-10587 Berlin</td>
</tr>
<tr>
<td>Voice</td>
<td>+49 30 8353-58185</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:a.kraft@telekom.de">a.kraft@telekom.de</a></td>
</tr>
</tbody>
</table>

## 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>
</tbody>
</table>

Copyright © Deutsche Telekom AG 2009 All Rights Reserved
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>Wi-Fi</td>
<td>Wireless Fidelity</td>
</tr>
</tbody>
</table>

8.3 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.