RFC-149 TR-069 Protocol Implementation Guideline

Draft

53 Pages

Abstract

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

1 Document Information

1.1 Table of Contents

1 Document Information **2**
1.1 Table of Contents 2
1.2 Terminology and Document Conventions 3
1.3 Revision History 4

2 Introduction **8**

3 Application Domain **8**

4 Problem Description **9**
4.1 Management agent making use of OSGi standardized Java interfaces 9
4.2 Mobile specification approach 10
4.2.1 Support for TR-069 notifications 10
4.3 Conclusion 11

5 Requirements **12**

6 Technical Solution **12**
6.1 Basic architecture of TR-069 Enabled OSGi implementation 13
6.2 RPC mapping for TR-069 14
6.2.1 TR-069 CPE RPCs to DMT Admin Interfaces 14
6.2.2 Notification Interfaces defined by DMT Admin Spec. to TR-069 ACS RPCs 21
6.2.3 EventAdmin Interfaces or DmtEventListener Interface to TR-069 ACS RPCs 21
6.3 Session management 22
6.3.1 Opening Session 22
6.3.2 Commit for Atomic Session 24
6.4 Mapping of data types 25
6.4.1 Mapping for GetParameterValues RPC 25
6.4.2 Mapping for SetParameterValues RPC 26
6.4.3 Responsibility on XML Values Handling 29
6.5 Error code and SOAP Fault 30
6.5.1 Error code and Fault code mapping 30
6.5.1.1 Mapping for handling SetParameterValues RPC 30
6.5.1.2 Mapping for handling AddObject or DeleteObject RPC 31
6.5.1.3 Mapping for handling GetParameterValues or GetParameterNames RPC 33
6.5.2 SOAP Fault expression 34
6.6 Translation on node names 34
6.7 Guidelines for Data Plugin Implementer 35
6.7.1 Session Type 35
6.7.2 FORMAT_XML and FORMAT_STRING 35

6.7.3 Design of a DMT Data Model to Support Monitoring of Instances Modification....35
6.7.4 Execute Operation....35

7 Javadoc....36
7.1 org.osgi.util.tr069
Class TR069ParameterValue....36
7.1.1 TR069_TYPE_INT.... 37
7.1.2 TR069_TYPE_UNSIGNED_INT.... 37
7.1.3 TR069_TYPE_LONG.... 38
7.1.4 TR069_TYPE_UNSIGNED_LONG.... 38
7.1.5 TR069_TYPE_STRING.... 38
7.1.6 TR069_TYPE_BOOLEAN.... 38
7.1.7 TR069_TYPE_BASE64.... 38
7.1.8 TR069_TYPE_HEXBINARY.... 39
7.1.9 TR069_TYPE_DATETIME.... 39
7.1.10 TR069ParameterValue.... 39
7.1.11 getValue.... 40
7.1.12 getType.... 40
7.1.13 getDmtData.... 40
7.1.14 getDmtDatasForList.... 44
7.1.15 getTR069ParameterValueForList.... 45
7.1.16 getTR069ParameterValue.... 45
7.2 org.osgi.util.tr069
Class TR069URI....48
7.2.1 getDmtUri.... 49
7.2.2 getTR069Path.... 49
7.2.3 isValidTR069Path.... 49

8 Considered Alternatives....50
8.1 Future work for notification handling 50
8.2 Future work for forced inform parameters 50
8.3 Future work for Set/GetParameterAttributes RPC 50

9 Security Considerations....50

10 Document Support....51
10.1 References....51
10.2 Author's Address....51
10.3 Acronyms and Abbreviations....52
10.4 End of Document....52

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 10.1.

`Source code is shown in this typeface.`

1.3 Revision History

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

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

Revision	Date	Comments
1.0.0	*Oct. 16 2009*	6th Draft The number of this RFC has been changed to 149. Notification handling is changed based on REG discussion. The operation for forced inform parameter is added. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.1.0	*Jan. 21 2010*	7th Draft Two implementation models are added to allow the RFC-145. Forced inform parameters handling mechanism is added. Notification event definition is changed to use multiple events. The data mapping is fixed to accommodate the new data types. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.2.0	*Apr. 5 2010*	8th Draft Based on the discussion of REG F2F in Mountain View, the following modifications are done; • abstract was modified. • some descriptions which has no relationships with what this RFC provides technically are removed in Section 4. • Requirements which this RFC cannot meet are removed in Section 5. • all descriptions of concrete solutions on notifications and forced inform parameters are removed in Section 6. • Section 7.1 and 7.2 are added for future work on notification and forced inform parameters, respectively. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp

Revision	Date	Comments
1.3.0	*Oct. 8 2010*	9th Draft Based on the discussion of REG F2F, June and August, and discussion on ML and Bugzilla, the following modifications are done; • Section 6.4 “Mapping of data types” was completely updated. • Details of mapping SetParameterValues RPC and GetParameterValues RPC is added in Section 6.2.1 • Javadoc for the utility class is added in Section 7. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.4.0	*Oct. 19 2010*	10th Draft Based on the discussion on ML and Bugzilla, the following modifications are done; • Section 6.4 “Mapping of data types” was completely updated. • Details of mapping SetParameterValues RPC and GetParameterValues RPC is added in Section 6.2.1 • Javadoc for the utility class is added in Section 7. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.5.0	*Nov. 26 2010*	11th Draft Based on the discussion on ML and Bugzilla, the following modifications are done; • Bugzilla#1672/1806/1739/1719/1796/ 1772/1975/1798. • Javadoc for the utility class are completely updated in Section 7. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.6.0	*Dec. 3 2010*	12th Draft Based on the discussion on ML and Bugzilla and the review by REG members, the RFC is updated. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp

Revision	Date	Comments
1.7.0	*Dec. 7 2010*	13th Draft Based on the discussion on ML and Bugzilla and the review by REG members, the RFC is updated. • Bugzilla#1819. ◦ Minor correction of "6.2.1 TR-069 CPE RPCs to DMT Admin Interfaes". ◦ 6.2.1.5 Mapping of DeletObject RPC to DMT Admin Interfaces" is newly added. ◦ "6.3.1 Opening Sessions" is updated. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.8.0	*Dec. 10 2010*	14th Draft • "6.3.1 Opening Sessions" is updated for correction. • Many correction against ProSyst Comments. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.9.0	*Dec. 11 2010*	15h Draft • Correction based on the comments from ProSyst.. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.10.0	*Dec. 14 2010*	16h Draft • Modification on Table 7 and section 7.1.3 based on the comment from ProSyst.. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.11.0	*Dec. 24 2010*	17h Draft • Modification on section 6.2.1.2 based on Bugzilla#1844. • Modification on section 6.3.1 based on Bugzilla#1844. • Modification on section 6.4.3 based on Bugzilla#1739. • Modification on section 6.6 based on Bugzilla#1828. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.12.0	*Jan. 18 2011*	18h Draft • Modification on section 6.3.1. based on Bugzilla#1844#c5. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp

Revision	Date	Comments
1.13.0	*Feb. 15 2011*	19h Draft • Update based on Bugzilla#1884 and 1885. 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.

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

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

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

3 Application Domain

Driven by triple play service delivery in the home network, fixed line access service providers have the need to configure home devices to ensure the proper service delivery. Broadband Forum's CPE WAN Management

Protocol (CWMP, alias TR-069) enables them to do this. By using a remote management server (Auto Configuration Server, ACS), they are able to manage TR-069 enabled devices. TR-069 provides them with possibilities to configure parameters, be informed of parameter changes (notification), start diagnostic tools, update firmware, etc.

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

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

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

4 Problem Description

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

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

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

- create a management agent bundle making use of the Java interfaces,

- create a protocol adapter bundle that interacts with the DMT Admin service, as defined in the OSGi Mobile specification.

4.1 Management agent making use of OSGi standardized Java interfaces

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

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

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

4.2 Mobile specification approach

The Mobile Expert Group has provided its own solution based on the OMA [3] Device Management [4] specification to provide a remote management solution. The OSGi Mobile specification [8]. contains two chapters related to remote management:

- chapter 3 "Mobile Management Tree": detailing the mobile management tree

- chapter 117 "DMT Admin Service Specification": detailing the DMT Admin service, plugins, and 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

- The complexity of mapping a new protocol to the OMA-inspired DMT model, which could imply performance issues on limited devices.

4.2.1 Support for TR-069 notifications

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

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

- The DMT Admin service sends events using the Event Admin service when operations have been performed on nodes (nodes added, removed or copied; node values changed etc.)

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

- The Monitor Admin service: A bundle can register a Monitorable service, to be used by the Monitor Admin service. By registering a Monitorable service, the bundle exposes access to a number of status variables. Notification can be implemented by the StatusVariable provider. If it does, it will call the update method on the

Monitor Listener. The Monitor Admin service then generates an event on the Event Admin service. The Monitor service is currently also represented in the DMT tree.

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

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

  Currently, the DMT Admin service only send events for operations on DMT nodes that were performed using the DMT Admin API. The DMT Admin Spec does not require Data Plugins to send events when DMT nodes are changed through except the DMT Admin API. It is called internal changes hereafter. For example: if configuration changes are performed by using the Configuration Admin service API, there might be no events sent. Such internal changes should be supported for TR-069 notification. However, it cannot be supported.

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

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

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

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

4.2.2 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 data types have to be mapped to the DMT Admin data types. However, TR-069 data types, such as “unsignedint” and “dateTime” (ISO 8601), cannot be translated appropriately into DMT Admin data types defined in the current specification. Translating these data types might result a limitation of the available value range and a complex object that consists of multiple nodes, respectively.

4.3 Conclusion

The OSGi Mobile specification delivers a standardized data model (the DMT), and standardized interface (on the DMT Admin service) to enable remote management through a protocol adapter. However, in the current

specification 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[~~8~~1]: The solution should specify a guideline of RPC mapping between DMT Admin service interfaces and remote management protocols, such as TR-069.

6 Technical Solution

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

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

- Guideline mapping between the DMT Admin interfaces and TR-069 protocol RPCs
- Management of sessions including read-only and transactional session

- Data type mapping between TR-069 data types and DMT Formats defined in the DMT Admin specification version 1.0 and newly defined in RFC141.

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

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

6.1 Basic architecture of TR-069 Enabled OSGi implementation

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

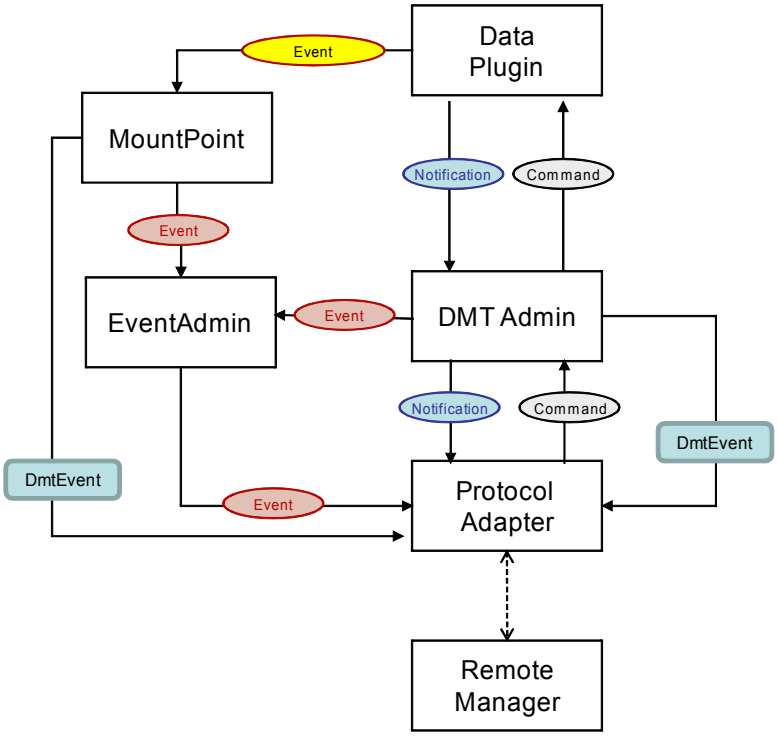

Fig.1 Basic aArchitecture of TR-069 enabled OSGiProtocol Implementation .

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

On the other hand, notifications from data plugins to remote managers are propagated through DMT Admin and TR-069 protocol adapter. There are two possible ways to send notifications to the TR-069 protocol adapter; the first one is using the Notification Service registered by the DMT Admin, and the second one is using the Mount Point. In the first case, the data plugin gets the services and calls the method to send a notification. The Remote Alert Sender service registered by the TR-069 protocol adapter will receive it. In the second case, the data plugin registers MountPlugin service and gets the Mount Point through it and calls the method to fire a event and the Event Handler service registered or DmtEventListener registered by the TR-069 protocol adapter will receive it. In

either case, then, the TR-069 protocol adapter should send the TR-069 notification to the remote manager properly based on the notification attributes.

6.2 RPC mapping for TR-069

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

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

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

This RFC provides the guidelines:

- how an implementation of TR-069 protocol adapter should call one or more DMT Admin methods, when it receives TR-069 RPCs from a remote manager, and
- how an implementation of TR-069 protocol adapter should call one or more TR-069 RPCs, when it receives DMT Admin methods callback, originated by Data Plugins.

6.2.1 TR-069 CPE RPCs to DMT Admin Interfaces

Table 1 shows the mapping guideline of TR-069 CPE RPCs to DMT Admin interfaces. TR-069 CPE RPCs are the RPCs implemented in the client side program and called by a remote server. TR-069 protocol adapter, therefore, calls DMT Admin service through the DmtSession interface.

Table 1:*Mapping of TR-069 CPE RPCs to DMT Admin interfaces*

Type	TR-069 RPCs	DMT Admin methods	Remarks
TR-069 CPE required RPC	GetRPCMethods	None	There is no recommendation for these RPCs. These RPCs' action depends on the implementation of the protocol adapter.
	GetParameterValues	See 6.2.1.1	See 6.2.1.1
	SetParameteValues	See 6.2.1.2	See 6.2.1.2
	GetParameterNames	See 6.2.1.3	See 6.2.1.3
	SetParameterAttributes	None	There is no recommendation for these RPCs. These RPCs' action depends on the implementation of the protocol adapter.
	GetParameterAttributes	None	
	AddObject	See 6.2.1.4.	See 6.2.1.4.
	DeleteObject	See 6.2.1.5.	See 6.2.1.5.
	Reboot	None	There is no recommendation for these RPCs. These RPCs' action depends on the implementation of the protocol adapter.
	Download	None	
TR-069 CPE optional RPC	ScheduleInform	None	This RPC is not related to DMT Admin method. Therefore, there is no recommendation for this RPC.

Type	TR-069 RPCs	DMT Admin methods	Remarks
	FactoryReset	None	There is no recommendation for these RPCs. These RPCs' action depends on the implementation of the protocol adapter.
	GetQueuedTransfers	None	
	Upload	None	
	SetVouchers	None	
	GetOptions	None	

6.2.1.1 Mapping of GetParameterValues RPC to DMT Admin Interfaces

[Done] Updated based on Bug#1975.

When a TR-069 Protocol Adapter(PA) receives GetParameterValues RPC, it will handle each requested Parameter names as follows:

1. The PA judges whether a Parameter name is partial path name or not. It can be done by checking if name ends with "." (dot) or not. If it is partial path name, the PA needs to return all of the Parameters in the branch of the naming hierarchy that shares the same prefix as the argument. For each name requested, the followings procedure Step 2 to Step 8 will be done:

2. The PA gets the node URI corresponding to the requested name by calling TR069URI.getDmtUri() with the name, the TR-069 path. The PA can check whether the name is valid TR-069 path by calling TR069URI.isValidTR069Path().

3. The PA uses DmtSession against the node URI as described in 6.3.1.

4. The PA calls DmtSession.getNodeType() with the node URI and if the node type is DmtConstant.DDF_LIST_SUBTREE, go to step 5. Otherwise go to step 6.

5. The PA gets the list of DmtData of all child leaf nodes of the node by DmtSession.getChildNodeNames() and DmtSession.getNodeValue(). Then it calls TR069ParameterValue.getTR069ParameterValueForList(). The method returns TR069ParameterValue instance. Go to step 7.

6. The PA gets the DmtData of nodes by DmtSession.getNodeValue() and calls TR069ParameterValue.getTR069ParameterValue() with it. The method returns TR069ParameterValue instance. Go to step 7.

7. The PA uses the data type and the value, which the returned TR069ParameterValue contains, for GetParameterValuesResponse RPC to ACS.

8. If the PA encounters the DmtException with PERMISSION_DENIED code in the procedure from Step2 to 7, it should continue operation for getting the next node name and should log it on info level.

9. If the PA encounters any Exception except DmtException.PERMISSION_DENIED or error in the procedure from Step2 to 7, the PA should stop to continue further operation on any other requested Parameter names. Then, the PA sends the CWMP Fault with the appropriate fault code in Table 7 to the ACS.

10. After all requested ParameterValueStruct are handled successfully, the PA will send GetParameterValuesResponse RPC to the ACS.

6.2.1.2 Mapping of SetParameterValues RPC to DMT Admin Interfaces

[Done] Updated based on Bug#1975/1796.

When a TR-069 Protocol Adapter(PA) receives SetParameterValues RPC, it will handle each requested ParameterValueStruct as follows:

1. The PA extracts each requested ParameterValueStruct.
2. For each of them, Step 2a to 2b should be done.
   a. The PA retrieves the value as String and the type.
      I. The PA should collapse whitespace as defined by XML Schema[9].. The simple rule is that this whitespace processing needs to be done for all but “xs:string” (See 6.4.3).. [DONE] Bug#1739#c20
      II. The value can be lexical representation as defined by XML Schema[9]. (See 6.4.3).[DONE] Bug#1739#c20
   b. The PA gets the node URI corresponding to the requested TR-069 path by calling TR069URI.getDmtUri() with the path name. The PA can check whether the name is valid TR-069 path by calling TR069URI.isValidTR069Path().
3. The PA uses DmtSession against the node URI as described in 6.3.1. and the PA finds the character set name specified as its SOAP request encoding. [DONE]Bug#1844
4. Then Step 4a to 4df should be done.
   a. The PA calls DmtSession.getNodeType() with the node URI and if the type of the node is DmtConstant.DDF_LIST_SUBTREE, go to Step 4b. Otherwise go to Step 4c. Before the PA sets a value to a node in Step 4b or 4c, it can try to identify potential problems for all target nodes. However it is up to the implementation of the PA. For example, it can do the following procedure:
      I. The PA verifies that all nodes are valid and support the intended operation by using MetaNode#isValidName(String name), MetaNode#isValidValue(DmtData value), and MetaNode#can(int operation). If verification fails, the following fault code can be used as SetParameterValuesFault for the ParameterValue.
         1) "9005 Invalid parameter name" if MetaNode#isValidName(String name) returns false,
         2) "9007 Invalid parameter value" if MetaNode#isValidValue(DmtData value) returns false,
         3) "9001 Request denied" if MetaNode#can(int operation) returns false.

II. Even if the verification of one node fails, the PA can continue to verify for all other nodes. After verification of all nodes finishes, the PA should send one CWMP Fault with the fault code "9003 Invalid arguments" , which contains multiple SetParamterValuesFaults, to the ACS.

III. Only if no potential problem is found for all nodes, the PA tries to change status by calling DmtSession#setNodeValue() or createLeafNode() for each ParameterValue. If some error happens, the PA should send one CWMP fault with fault code 9003 with the SetParameterValuesFault of the fault code in Table 5 to the ACS. Further operation for changing status must not be done. The PA can roll back by setting old values to the node already set by this SetParameterValues operation. If the rollback fails, that should be logged but Fault code for this rollback failure should not be included in CWMP Fault or SetParameterValuesFault to the ACS.

1) [CAUTION] When a DmtSession is operated for setting a value to a node, DMT Admin does the same verification as specified in Step 4a I.. It means, verification will be duplicated and might cause a performance problem.

b. The PA gets the MetaNode object of the child node of the specified node URI by DmtSession.getMetaNode() and calls TR069ParameterValue.getDmtDataForList() with the value, the TR-069 type, the character set name, the node URI and the retrieved MetaNode. The method returns array of DmtData. For each element, the TR-069 PA sets the DmtData to the child node of the specified node URI. Go to Step 4d.

I. The first element of the array of DmtData should be set to the node name “1” and the node name should be incremented one by one. For example, if the size of the array is 4, the leaf node names must be “1”,”2”,”3”, and “4”. If the node does not exist, DmtSession.createLeafNode() is used. How to handle existing node depends on the implementation of PAs. For instance, deleting all child nodes by DmtSession.deleteNode() and creating new all child nodes by DmtSession.createLeafNode() is one way.

c. The PA gets the MetaNode object of the specified node URI by DmtSession.getMetaNode(). Then it calls TR069ParameterValue.getDmtData() with the value, the TR-069 type, the character set name, the node URI and the retrieved MetaNode. The method will return an DmtData. The PA sets the DmtData to the specified node URI. Go to Step 4d.

d. If the used session is atomic and ~~, go to Step 4c. Otherwise Step 4f.~~

e. ~~If~~ the PA encounters any error in Step 4a to 4c, the CWMP fault with fault code 9003 with the SetParameterValuesFault with the fault code in Table 5 will be sent to the ACS. Further operation for other ParamereterValues must not be done. The protocol adapter should call DmtSession#rollback(). If the rollback fails, that should be logged on error level but CWMP Fault or SetParameterValuesFault to the ACS should not include any information for this rollback failure.

~~Before the PA sets a value to a node in Step 4b or 4c, it can try to identify potential problems for all target nodes. However it is up to the implementation of the PA. For example, it can do the following procedure:~~

I. ~~The PA verifies that all nodes are valid and support the intended operation by using MetaNode#isValidName(String name), MetaNode#isValidValue(DmtData value), and MetaNode#can(int operation). If verification fails, the following fault code can be used as SetParameterValuesFault for the ParameterValue.~~

1) ~~"9005 Invalid parameter name" if MetaNode#isValidName(String name) returns false,~~

2) ~~"9007 Invalid parameter value" if MetaNode#isValidValue(DmtData value) returns false,~~

3) ~~"9001 Request denied" if MetaNode#can(int operation) returns false.~~

II. ~~Even if the verification of one node fails, the PA can continue to verify for all other nodes. After verification of all nodes finishes, the PA should send one CWMP Fault with the fault code "9003 Invalid arguments" , which contains multiple SetParamterValuesFaults, to the ACS.~~

III. ~~Only if no potential problem is found for all nodes, the PA tries to change status by calling DmtSession#setNodeValue() or createLeafNode() for each ParameterValue. If some error happens, the PA should send one CWMP fault with fault code 9003 with the SetParameterValuesFault of the fault code in Table 5 to the ACS. Further operation for changing status must not be done. The PA can roll back by setting old values to the node already set by this SetParameterValues operation. If the rollback fails, that should be logged but Fault code for this rollback failure should not be included in CWMP Fault or SetParameterValuesFault to the ACS.~~

f. ~~[CAUTION] When a DmtSession is operated for setting a value to a node, DMT Admin does the same verification as specified in Step 4 f エラー: 参照先が見つかりません。. It means, verification will be duplicated and might cause a performance problem.~~

5. After all requested ParameterValueStruct are handled successfully, if the used_session is atomic, go to Step 6~~5~~. Otherwise go to Step 7.

6. The PA should commit. If commit succeeds, go to Step 7. Otherwise, the PA should send one CWMP fault with fault code 9003 with the SetParameterValuesFault of the fault code in Table 5 to the ACS. Then DmtSession#rollback() must be called. If commit fails, go to Step 6.a. [DONE] Bug1795

   a. The PA checks whether the thrown DmtException.getURI() returns null or not. If null, go to Step 6.b. Otherwise, go to Step 6.c.

   b. [case that DmtException.getURI() == null] The PA gets the root uri of the session by DmtSession#getRootUri() and extracts all node names to be set for SetParameterValues RPC, which is under the root uri. For all nodes of them, SetParameterValuesFault with "9002 Internal error" are included in one CWMP Fault. Go to Step 6.d.

   c. [case that DmtException.getURI() != null] The PA should send one CWMP fault with fault code 9003 with one SetParameterValuesFault of the fault code in Table 5 and the return uri to the ACS. Go to Step 6.d.

   d. The PA should rollback. If the rollback fails, that should be logged on warn level. No Fault code should be included in a CWMP Fault or SetParameterValuesFault to the ACS for the failure of the rollback.

7. The PA will send SetParameterValuesResponse RPC to the ACS.

   a. The value of Status of the SetParameterValuesResponse should be "0".[DONE] Bug1798.

6.2.1.3 Mapping of GetParameterNames RPC to DMT Admin Interfaces

This RPC is used to obtain parameters of the specified ParameterPath, which can be either a complete Parameter name, or a partial path name. When a TR-069 Protocol Adapter(PA) receives GetParameterNames RPC, it will handle the requested ParameterPath as follows:

1. If the ParameterPath is a complete Parameter name, go to Step 2. Otherwise go to Step 3.

2. [Complete Parameter name case] If the NextLevel specified is true, it is an error and go to Step 5. Otherwise go to Step 2.a.

    a. The PA translates the Parameter name into DMT uri. Go to Step 2.b.

    b. The PA uses a DmtSession against the uri as described in 6.3.1.

    c. The PA calls DmtSession.isLeaf() against the uri. If the node is leaf, go to Step 2.d. Otherwise, go to Step 2.e.

    d. [Leaf Case] The PA needs to check whether the node is writable or not. The PA gets the MetaNode of the uri by DmtSession.getMetaNode(). If the MetaNode.can(MetaNode.CMD_REPLACE) returns false, the node is assumed as not writable. Otherwise, the PA gets the Acl of the uri by DmtSession.getEffectiveNodeAcl(). If the Acl.isPermitted() with both the principal used for the session and Acl.REPLACE returns false, the node is assumed as not writable. Otherwise, the node is assumed as writable.

    e. [Interior Case] The PA gets the node type of the uri by DmtSession.getNodeType(). If the type is not DmtConstants.DDF_LIST_SUBTEE, it is an error and go to step 5. Otherwise, go to step 2.e.I.

        I. The PA gets the MetaNode of the list subtree leaf nodes by DmtSession.getMetaNode(). If MetaNode.can(MetaNode.CMD_REPLACE | MetaNode.CMD_ADD | MetaNode.CMD_DELETE) returns false, the node is assumed as not writable.

        II. the PA gets the names of the list subtree by DmtSession.getChildNodeNames(). For each leaf nodes, Step 2.e.II.A is done. After all are done, go to Step 2.e.III.

            A. The PA gets the Acl of the node by DmtSession.getEffectiveNodeAcl() and check weather if the Acl allows the all operation of add, delete, and replace. For the check, Acl.isPermitted() with both the principal used for the session and (Acl.REPLACE | Acl.ADD | Acl.DELETE) can be used. If the Acl of any leaf node does not allow it, the Parameter is assumed as not writable.

        III. If the Acl of all leaf nodes allows it, the Parameter is assumed as writable. Ggo to Step 2.e.IV..

        IV. The PA converts the node uri to TR-069 path. Then the PA constructs a ParameterInfoStruct with the converted path and the Writable info. The constructed ParameterInfoStruct is kept.

3. [Partial path name case]

    a. The PA converts the partial path name into DMT uri.

    b. The PA uses DmtSession against the uri as described in 6.3.1.

    c. The PA calls DmtSession.isLeaf() against the uri. If the node is leaf, it is an error and go to Step 5. Otherwise, the PA gets the node type. If the node type is DmtConstants.DDF_LIST_SUBTEE, it is an error and go to Step 5. Otherwise, if the NextLevel specified is true, go to Step 3.d. Otherwise go to Step 3.e.

d. [case that NextLevel is true] The PA gets child node names by DmtSession.getChildNodeNames() with the uri. For each child nodes, the following steps are done.

I. If the child node is leaf, PA checks whether the node is writable as same as Step 2.d. Otherwise, PA checks whether the list subtree is writable as same as Step 2.e

II. The PA converts the node uri to TR-069 path. Then the PA constructs a ParameterInfoStruct with the converted path and the Writable info. The constructed ParameterInfoStruct is kept.

e. [case that NextLevel is false] The PA gets the all node names under the subtree, whose root node is the converted uri. For it, DmtSession.getChildNodeNames() is called recursively. For each child nodes, Step 3.d.I and 3.d.II are done. Remember that the ParameterInfoStructs kept must include the one of the converted uri.

4. After all ParameterInfoStruct are constructed, the PA returns GetParameterNamesResponse with the all ParameterInfoStruncts kept.

5. The PA returns a fault response with fault code 9003(Invalid Arguments).

6. If the PA encounters the DmtException with PERMISSION_DENIED code, it should continue operation for getting the next node name and should log it on info level. That node is hidden for the ACS. If it encounters other exception, it should stop to proceed the operation for other nodes and return CWMP Fault with the code in Table 7 to the ACS .

6.2.1.4 Mapping of AddObject RPC to DMT Admin Interfaces

This RPC is used to create an object sub-tree including interior nodes and leaf nodes, and only interior node indicating object name, which is usually the node locating at the top of sub-tree hierarchy, can be specified. When a TR-069 Protocol Adapter(PA) receives AddObject RPC, it will handle the request as follows:

1. The PA translates the ObjectName into DMT uri.

2. The PA uses DmtSession against the node URI as described in 6.3.1.

3. The PA should call DmtSession.getChildNodeNames() to get instance IDs currently existing and update the mapping table as described in 6.6. According to the mapping table, t~~to check the instance IDs which is created by the data plugin internally. All node names of children must be integer greater than 0. If not, it is an error condition. Then,~~ the PA chooses one instance ID that has been never used before, which means the instance ID must not be reused for the same object beyond reboots of an OSGi framework.

4. The PA should then call the DmtSession.createInteriorNode() with the node uri for the chosen instance ID in Step 3. Then the data plugin tries to create the desired object under the indicated instance ID. If the indicated ID is not acceptable for the data plugin, for example the ID is already used by the data plugin internally or the ID is reserved to create instance in the future, the data plugin must throw DmtException with NODE_ALREAD_EXIST code to the PA. If that DmtException is thrown, the PA should change the instance ID and should retry the creation of the object by calling DmtSession.createInteriorNode() until the prescribed retry count has been exceeded. The retry count is up to the PA implementation.

5. If the PA encounters any error, the CWMP fault code in Table 6 must be used. Especially, if a DmtSession#createInteriorNode(node) throws METADATA_MISMATCH, the PA should check whether the MetaNode#getMaxOccurence() <= the size of array of DmtSession#getChildNodeNames() against the parent node. If true, 9004 should be chosen for the CWMP fault code.

6. If the PA succeeds to create the interior node, the PA must return the AddObjectResponse with the instance ID to the ACS. The the value of Status of the AddObjectResponse should be "0".

[Remarks] A data plugin may give the same instance ID to the same target in order to keep consistency of the node path, if the instance is created by not the operation from the PA but by the data plugin internally.

6.2.1.5 Mapping of DeleteObject RPC to DMT Admin Interfaces

This RPC is to delete the specified object including interior and leaf nodes. When a TR-069 Protocol Adapter(PA) receives AddObject RPC, it will handle the request as follows:

1. The PA translates the ObjectName into DMT uri.
2. The PA uses DmtSession against the node URI as described in 6.3.1.
3. The DmtSession.deleteNode() should be called with the translated uri.
4. If the PA encounters any error, it must send the CWMP Fault with the appropriate fault code in Table 6 to the ACS.
5. If the PA succeeds to delete the interior node, the PA must return the DeleteObjectResponse. The the value of Status of the DeleteObjectResponse should be "0".

6.2.2 Notification Interfaces defined by DMT Admin Spec. to TR-069 ACS RPCs

Table 2 shows the mapping guideline of DMT Admin interfaces to TR-069 ACS RPCs. TR-069 ACS RPCs are the RPCs implemented in the server side program and called by a managed client. Therefore, the TR-069 protocol adapter should call TR-069 ACS RPCs when the DMT Admin calls the TR-069 protocol adapter through the RemoteAlertSender interface.

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

Table 2:*Mapping of ~~to TR-069 ACS RPCs~~ DMT Admin interfaces to TR-069 ACS RPCs*

Type	DMT Admin methods	TR-069 RPCs	Remarks
TR-069 ACS required RPC	RemoteAlertSender.sendAlert()	Inform	A protocol adapter must register the RemoteAlertSender service with the “principals” property that represents the associated principals of the protocol adapter. When the RemoteAlertSender.sendAlert() implemented by the protocol adapter is called, the Inform RPC should be fired to the associated ACS. The “code” specified as the parameter of the method should be interpreted as the EventCode

Type	DMT Admin methods	TR-069 RPCs	Remarks
			included in the EventStruct defined in the TR-069 specification. And the ParameterList that is contained in the Inform RPC should include the ParameterValueStruct consisting of the Name and Value, which should be the “source” and the “data” included in the AlertItem, respectively. If the AlertItem does not contain any “source” as the node that is the cause of the alert, the protocol adapter should discard the AlertItem and does not have to send Inform RPC. Note that this definition intends to implement scenarios in which an alert emerges from data plugins. Therefore, other use cases, such as Periodic, Boot and TransferComplete, should be implemented in an user-defined way.
	None	TransferComplete	There is no recommendation for these RPCs. This RPC's action depends on the implementation of the protocol adapter.
TR-069 ACS optional RPC	None	GetRPCMethods	There is no recommendation for these RPCs. These RPCs' action depends on the implementation of the protocol adapter.
	None	AutonomousTransferComplete	
	None	RequestDownload	

6.2.3 EventAdmin Interfaces or DmtEventListener Interface to TR-069 ACS RPCs

A TR069 protocol adapter(PA) must receive events delivered through Event Admin service or DmtEventListener.

When a PA receives info/dmtree/DmtEvent/ADDED, info/dmtree/DmtEvent/DELETED, info/dmtree/DmtEvent/REPLACED, info/dmtree/DmtEvent/RENAMED, or info/dmtree/DmtEvent/COPIED via EventHandler, or when a PA receives corresponding DmtEvents via DmtEventListener, it should call TR-069 ACS RPCs “Inform” appropriately to send notification to the ACS. The details are out of scope of this RFC.

When a PA receives info/dmtree/DmtEvent/DESTRUCTIVE_OPEARTION through EventHandler#handleEvent() or DmtEvent.DESTRUCTIVE_OPERATION through SynchronousDmtEventListener#changeOccured(), it should finish to call TR-069 ACS RPCs “Inform” appropriately to send notification to the ACS before EventHandler#handleEvent() method returns. In other words, the PA should not return the method before it finishes to send notifications to ACS or it finishes current operation. If the method returns, all bundles on the framework including the protocol adapter bundle might be stopped.

6.3 Session management

6.3.1 Opening Session

The DMT Admin specification defines three types of locking modes of sessions for protocol adapters to open a DmtSession for reading or writing node values: LOCK_TYPE_SHARED, LOCK_TYPE_EXCLUSIVE and LOCK_TYPE_ATOMIC, which correspond to the ReadableDataSession, the ReadWriteDataSession and the TransactionalDataSession respectively.

The TR-069 Protocol Adapter(PA) adaptively chooses any of LOCK_TYPE_SHARED, LOCK_TYPE_EXCLUSIVE, and LOCK_TYPE_ATOMIC as the lock mode based on the RPC, when it receives the SOAP request from the remote manager.

- If the received RPC may need to write multiple node values, such as SetParameterValues which contains multiple ParameterValueStructs or SetParameterValues which contains only one ParameterValueStruct with the type of “string”, the PA should basically use LOCK_TYPE_ATOMIC session.
  - Note that if the SetParameterValues contains only one ParameterValueStruct with the type of “string” and the type of the node in DMT corresponding to the parameter name is DmtConstant.DDF_LIST_SUBTREE~~_URI~~, the PA may need to write multiple node values.
  - The PA, firstly, should find the nearest common ancestor node for all ParameterValueStrucsts before opening session. [DONE] Bug#1844
  - If there already exists an opened LOCK_TYPE_ATOMIC session which satisfies the following both conditions, the session will continue to be used. Otherwise, the PA should try to open a new session as LOCK_TYPE_ATOMIC for the found ancestor node.
    - Condition1: The root uri of the session points the found ancestor node or any ancestor nodes of the found ancestor node.
    - Condition2: The session is opened with the same principal.
  - If a PA can use not LOCK_TYPE_ATOMIC but LOCK_TYPE_EXCLUSIVE, the PA may use the LOCK_TYPE_EXCLUSIVE instead of the LOCK_TYPE_ATOMIC.
    - If there already exists an opened LOCK_TYPE_EXCLUSIVE session which satisfies the following both conditions, the session will continue to be used. Otherwise, the PA should try to open a new session as LOCK_TYPE_EXCLUSIVE for the found ancestor node.
      - Condition1: The root uri of the session points the found ancestor node or any ancestor nodes of the found ancestor node.
      - Condition2: The session is opened with the same principal.
    - In the case of LOCK_TYPE_EXCLUSIVE session, the atomicity of the data is not ensured by the DMT Admin service.
    - Note that plugin can support TransactionalDataSession, but not ReadWriteDataSession.
  - If a PA cannot use LOCK_TYPE_ATOMIC either LOCK_TYPE_EXCLUSIVE, the PA can not change values for any nodes and should return the corresponding SOAP Fault response to the ACS.
- If the received RPC needs to modify a node but does not need to write multiple node values, such as SetParameterValues which contains only one ParameterValueStruct except the one with the type of “string”, AddObejct, or DeleteObject, the PA should basically use LOCK_TYPE_EXCLUSIVE session.
  - If there already exists an opened LOCK_TYPE_EXCLUSIVE session which satisfies the following both conditions, the session will continue to be used. Otherwise if there already exists an opened LOCK_TYPE_ATOMIC session which satisfies the following both conditions, it depends on the implementation of PA whether to continue to use the opened session or open another session as LOCK_TYPE_EXCLUSIVE. Otherwise, the PA should open a new session as LOCK_TYPE_EXCLUSIVE.

- Condition1: The root uri of the session points the found ancestor node or any ancestor nodes of the found ancestor node.
- Condition2: The session is opened with the same principal.

- If the received RPC does not need to modify any node, such as GetParameterValues or GetParameterNames, the PA should basically use LOCK_TYPE_EXCLUSIVE or LOCK_TYPE_SHARED session.
  - If there already exists an opened LOCK_TYPE_EXCLUSIVE or LOCK_TYPE_SHARED session which satisfies the following both conditions, the session will continue to be used. Otherwise if there already exists an opened LOCK_TYPE_ATOMIC session which satisfies the following both conditions, it depends on the implementation of PA whether to continue to use the opened session or open another session as LOCK_TYPE_EXCLUSIVE or LOCK_TYPE_SHARED. Otherwise, the PA should open a new session as LOCK_TYPE_EXCLUSIVE or LOCK_TYPE_SHARED.
    - Condition1: The root uri of the session points the found ancestor node or any ancestor nodes of the found ancestor node.
    - Condition2: The session is opened with the same principal.
  - *Note:* Even if the LOCK_TYPE_SHARED session is used, as long as the session is opened, the subtree under the root uri of the opened session must not be changed by other DmtSession operations according to the DMT Admin Service Specification.
  - A Data Plugin can support neither LOCK_TYPE_SHARED nor LOCK_TYPE_EXCLUSIVE but only LOCK_TYPE_ATOMIC. Therefore, if the PA tries to open a new session as LOCK_TYPE_EXCLUSIVE or LOCK_TYPE_SHARED but failed to do for neither of them, the PA should open a new session as LOCK_TYPE_ATOMIC

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

I

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

6.3.2 Commit for Atomic Session

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

Let us assume that the protocol adapter receives the SOAP request from the remote manager and the protocol adapter needs to use a LOCK_TYPE_ATOMIC session as described in Section ~~6.4.1~~6.3.1. In that case, the session should be opened at the start of processing the request that needs to write node values, if the session

has not been opened. If the corresponding RPC operation has been completed normally, the protocol adapter should call the DmtSession.commit() method before returning the SOAP response to the remote manager. After calling commit(), it can close the session.

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

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

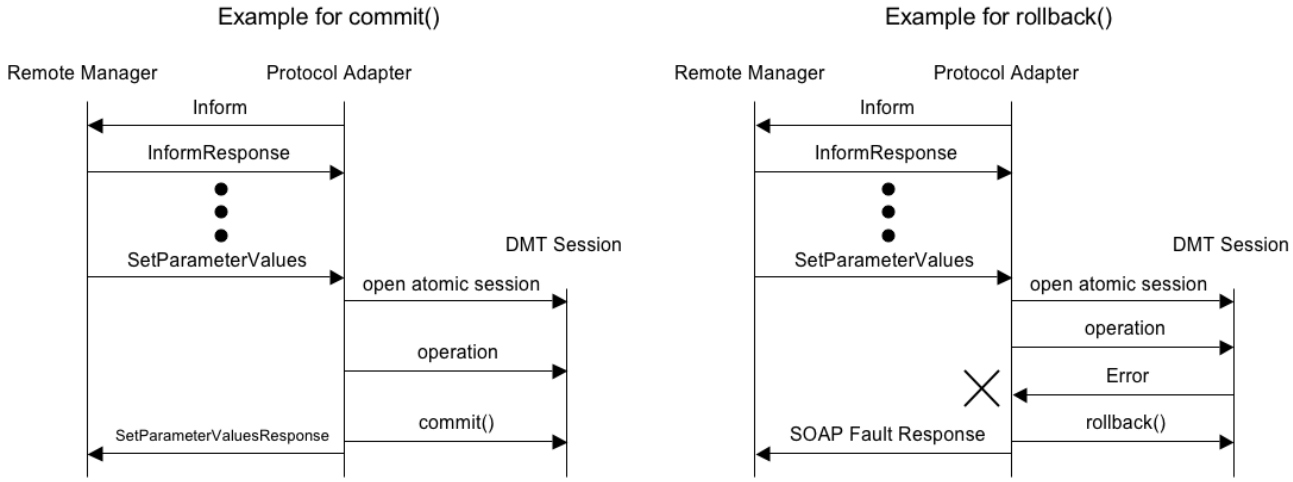

Fig.23 Sequence Examples of RPCs including Response and atomic session management

6.4 Mapping of data types

There 9 data types are defined in the TR-106 [7], while extended Dmt Admin Service specification based on RFC141 defines more than 9 DmtData types. It means, there exists no obvious one-to-one mapping. Therefore the mapping rule is needed.

6.4.1 Mapping for GetParameterValues RPC

For handling GetParameterValues RPC, the TR-069 Protocol Adapter can get DmtData of the corresponding node URI and the DmtData must have its format. This RFC defines mapping rule from the DmtData format to the TR-069 data type to be used in GetParameterValueResponse RPC as Table 3.

Table 3:Mapping from the DmtData format to TR-069 data type for GetParameterValues RPC

Leaf or Interior in DMT	Format of the DmtData of the specified node	The dataType that TR069 GetParameterValuesResponse needs to specify .	comment
Leaf	FORMAT_BASE64	base64	The byte array returned by DmtData.getBase64() will be converted into base64. The result of the conversion will be used as value.
	FORMAT_BINARY	hexBinary	The byte array returned by DmtData.getBinary() will be

			converted into hexBinary as described in 6.4.3, e.g. upper-case Hex letters without any whitespaces embedded. The result of the conversion will be used as value. [DONE] Bug#1739#c20
	FORMAT_BOOLEAN	boolean	“0” or “1”, which DmtData.getBoolean() is translated into, will be used as value.
	FORMAT_DATE	string	DmtData.toString() or DmtData.getDate() will be used as value.
	FORMAT_FLOAT	string	DmtData.toString() will be used as value.
	FORMAT_INTEGER	int	DmtData.toString() will be used as value.
	FORMAT_NULL	string	DmtData.toString() will be used as value.
	FORMAT_RAW_BINARY	base64	The byte array returned by DmtData#getRawBinary() will be converted into base64. The result of the conversion will be used as value.
	FORMAT_RAW_STRING	string	DmtData.toString() or DmtData.getRawString() will be used as value.
	FORMAT_STRING	string	DmtData.toString() or DmtData.getString() will be used as value.
	FORMAT_TIME	string	DmtData.toString() or DmtData.getTime() will be used as value.
	FORMAT_XML	string	DmtData.toString() or DmtData.getXml() will be used as value.
	FORMAT_UNSIGNED_INTEGER	unsignedInt	DmtData.toString() or DmtData.getUnsignedInteger() will be used as value.
	FORMAT_LONG	long	DmtData.toString() will be used as value.
	FORMAT_UNSIGNED_LONG	unsignedLong	DmtData.toString() or DmtData.getUnsignedLong() will be used as value.
	FORMAT_DATETIME	dateTime	DmtData.toString() or or DmtData.getDateTime() will be used as value.
	FORMAT_HEXBINARY	hexBinary	The byte array returned by DmtData.getHexBinary() will be converted into hexBinary as described in 6.4.3, e.g. upper-case Hex letters without any whitespaces embedded. The result of the conversion will be used as value. [DONE] Bug#1739#c20
	FORMAT_NODE_URI	string	The returned String by DmtData.toString() or DmtData.getNodeUri() will be translated into TR-069Path. For the translation, TR069URI.getTR069Path() can be used. The result of the translation will be used as value.
Interior	FORMAT_NODE	string, or error.	If the type of the node is DmtConstant.DDF_LIST_SUBTREE, the subtree under the node should be treated as a list subtree and comma-separated string should be constructed. Otherwise, error. All DmtData objects of the child leaf nodes will be

			retrieved and values of each will be gotten as different rows in this table show. The values need to be escaped according to TR-106. Then the values are concatenated with each others into Comma-Separated String.

6.4.2 Mapping for SetParameterValues RPC

Mapping for SetParameterValues RPC is more complicated than the one for GetParameterValues RPC, because one node in DMT can support multiple formats. The formats which a node supports can be retrieved by MetaNode.getFormats().

If the data type is any of “boolean”, “int”, ”unsignedInt”, “long”, “unsignedLong” and “dateTime”, the mapping is simple as Table 4. If the corresponding node in DMT does not support the FORMAT specified in the table, setting PearaemeaterValue for the node is failed.

Table 4:Mapping from the TR-069 data type to DmtData Format for SGetParameterValues RPC for “boolean”, “int”, “unsignedInt”, “long”, “unsignedLong”, and “dateTime”.

The dataType that TR069 XML in SetParameterValues specified .	FORMAT of the DmtData of the specified node to be used.	comment
boolean	FORMAT_BOOLEAN	DmtData(boolean) is used according to the value, which can be either “0”/“1” or “true”/”false”. If the value is not any of them, setting ParematerValue for the node is failed. [DONE] Bug#1739#c21
int	FORMAT_INTEGER	DmtData(int) will be used. If the parsing the value as int is failed, setting ParematerValue for the node is failed.
unsignedInt	FORMAT_UNSIGNED_INTEGER	DmtData(String, int format) will be used. If the constructor throws Exception, setting ParematerValue for the node is failed.
long	FORMAT_LONG	DmtData(long) will be used. If the parsing the value as long is failed, setting ParematerValue for the node is failed.
unsignedLong	FORMAT_UNSIGNED_LONG	DmtData(String, int format) will be used. If the constructor throws Exception, setting ParematerValue for the node is failed.
dateTime	FORMAT_DATETIME	DmtData(String, int format) will be used. If the constructor throws Exception, setting ParematerValue for the node is failed.

If the data type is any of “base64”, “hexBinary”, and “string”, the mapping is more complicated, because one node in DMT can support multiple formats. For example, in the case that the data type is “string” and the corresponding

node can supports both of FORMAT_DATE and FORMAT_STRING, TR-069 Protocol Adapter needs to choose any of them. To keep consistent behavior of TR-069 Protocol Adapter, this RFC defines the following algorithms:

Case1: TR-069 data type is “base64”.

1. If the node supports FORMAT_BASE64, go to Step 1.a., otherwise, go to Step 2.
   a. The TR-069 Protocol Adapter chooses FORMAT_BASE64: it coverts the specified value to base64 byte array and constructs DmtData(byte[], boolean) with “true” as the second argument.
2. If the node supports FORMAT_RAW_BINARY, go to Step 2.a., otherwise, go to Step 3.
   a. The TR-069 Protocol Adapter gets array of raw format names by MetaNode.getRawFormatNames().
   b. If the returned array has size zero or the first element of it is null or empty String, go to Step 3. Otherwise, the TR-069 Protocol Adapter chooses FORMAT_RAW_BINARY. It encodes the value String into a sequence of bytes using the character set specified in SOAP request. If the encoding fails, go to Step3. Otherwise, then it constructs DmtData(String~~int~~ formatName, byte[]) with the first elements in the array of raw format names as the first argument.
3. S~~s~~etting ParemeterValue for the node is failed.

Case2: TR-069 data type is “hexBinary”.

1. If the node supports FORMAT_HEXBINARY, go to Step 1.a., otherwise, go to Step 2.
   a. The TR-069 Protocol Adapter chooses FORMAT_HEXBINARY: it coverts the specified value to hexBinary byte array and constructs DmtData(byte[], FORMAT_HEXBINARY).
2. If the node supports FORMAT_BINARY, go to Step 2.a., otherwise, go to Step 3.
   a. TR-069 Protocol Adapter choose FORMAT_BINARY. It encodes the value String into a sequence of bytes using the character set specified in SOAP request. If the encoding fails, go to Step3. Otherwise, then it constructs DmtData(byte[], FORMAT_BINARY) .
3. setting ParemeterValue for the node is failed.

Case3: TR-069 data type is “string”.

1. If the node supports FORMAT_NULL, go to Step 1.a., otherwise, go to Step 2.
   a. if the value is empty String, then FORMAT_NULL is chosen. Otherwise go to Step 2.
2. If the node supports FORMAT_FLOAT, go to Step 2.a., otherwise, go to Step 3.
   a. If the parsing as float succeeds, then FORMAT_FLOAT is chosen. Otherwise go to Step 3.
3. If the node supports FORMAT_DATE, go to Step 3.a., otherwise, go to Step 4.

a. If the construct DmtData(String, FORMAT_DATE) succeeds, then FORMAT_DATE is chosen, ~~O~~otherwise go to Step 4.

4. If the node supports FORMAT_TIME, go to Step 4.a., otherwise, go to Step 5.
   a. If the construct DmtData(String, FORMAT_TIME) succeeds, then FORMAT_TIME is chosen, Otherwise go to Step 5.
5. If the node supports FORMAT_NODE_URI, go to Step 5.a., otherwise, go to Step 6.
   a. If the construct DmtData(String, FORMAT_NODE_URI) succeeds, then FORMAT_NODE_URI is chosen, Otherwise go to Step 6. Before construct a DmtData object, TR-069 Protocol Adapter has to translate the value to absolute URI, if the value is relative path reference, which starts with "./".
6. If the node supports FORMAT_RAW_STRING, go to Step 6.a., otherwise, go to Step 7.
   a. The TR-069 Protocol Adapter gets array of raw format names by MetaNode.getRawFormatNames().
   b. If the returned array has size zero or the first element of it is null or empty String, go to Step 7. Otherwise, the TR-069 Protocol Adapter chooses FORMAT_RAW_STRING: it constructs DmtData(String, String).
7. If the node supports FORMAT_STRING, go to Step 7.a., otherwise, go to Step 8.
   a. The TR-069 Protocol Adapter chooses FORMAT_STRING: it constructs DmtData(String).
8. If the node supports FORMAT_XML, go to Step 8.a., otherwise, go to Step 9.
   a. The TR-069 Protocol Adapter chooses FORMAT_XML: it constructs DmtData(String, FORMAT_XML).
9. setting ParemeterValue for the node is failed.

6.4.3 Responsibility on XML Values Handling

[DONE]Bug#1739#c20

For handling SetParameterValues RPC, a TR-069 protocol adapter (PA) will extract parameter values from it.

- The data type can be inferred from the XML itself via the xsi:type attribute.
- These values can be anything that is valid for the corresponding SOAP data type.
  - It means that the value can be in either lexical representation or canonical representation as defined by XML Schema[9]. . The PA should handle either of them.
    - Remark that TR069ParameterValue#getDmtData() and getDmtDataForList() are responsible for handling the value in lexical representation.
  - It means that the PA should collapse whitespace for all non-string data types, as defined by XML Schema [9]. .

- Remark that TR069ParameterValue#getDmtData() and getDmtDataForList() are not responsible for collapsing it.

For handling GetParameterValues RPC,

- If operation succeeds, a PA must return GetParameterValuesResponse whose values are valid for the corresponding SOAP type.
  - It means that hexBinary and numeric values cannot contain embedded spaces.
  - It means that the values included the GetParameterValuesResponse should be canonical representation as defined by XML Schema[9]..
    - Remark that the String returned by getValue() method of the TR069ParameterValue instance must be canonical representation. [DONE] Bug#1739#c20

[Bug#1719]

When a PA needs to separate the list into its items for SetParameterValues RPC, it should unescape the individual items as specified by TR-106[7]. before passing them to the Data Plugin.

- e.g. SetParameterValues of "a%20,%0db, c", for which the escaped items are "a%20", "%0db" and "c", and the unescaped items (sent to the data plugin) are "a ", "\rb" and "c".
- Remark that TR069ParameterValue.getDmtDataForList() is responsible for the unescaping.

When a PA needs to create a comma-separated list for GetParameterValues, it should escape each item before creating the comma-separated list as specified by TR-106[7]. .

- e.g. for GetParameterValues, the encoded comma-separated list might be "a%20,%0db,c" (note no space before the last item) but could equally well be "a%20, %0db, c" (note a space after each comma).
- Remark that TR069ParameterValue.getTR069ParmeterValueForList() is responsible for the escaping.

6.5 Error code and SOAP Fault

6.5.1 Error code and Fault code mapping

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

6.5.1.1 Mapping for handling SetParameterValues RPC

If the PA encounters any error for handling SetParameterValues RPC, 9003 must be used for CWMP Fault. The CWMP Fault can contain multiple SetParameterValuesFault. Table 5 shows the fault code to be used in SetParameterValuesFault.

[Done] Bug#1796

Table 5:*Mapping of Exception code to TR-069 Fault Code for SetParameterValuesFault*

Class	DmtException code	TR-069 Fault code	Remarks
DmtException	ALERT_NOT_ROUTED	no recommendation	A protocol adapter is not expected to receive this Exception.
	COMMAND_FAILED	9002 Internal error	
	COMMAND_NOT_ALLOWED	9001 Request denied	
	CONCURRENT_ACCESS	9002 Internal error	In case of atomic sessions, this Exception is thrown if the commit operation failed because of some modification outside the scope of the DMT to the nodes affected in the session.
	DATA_STORE_FAILURE	9002 Internal error	
	FEATURE_NOT_SUPPORTED	9001 Request denied	
	INVALID_URI	9005 Invalid parameter name	
	METADATA_MISMATCH	9007 Invalid parameter value	
	NODE_ALREADY_EXISTS	9002 Internal error	
	NODE_NOT_FOUND	9005 Invalid parameter name	
	PERMISSION_DENIED	9008 Attempt to set non-writable parameter	
	REMOTE_ERROR	no recommendation	A protocol adapter is not expected to receive this Exception.
	ROLLBACK_FAILED	9002 Internal error	
	SESSION_CREATION_TIMEOUT	9001 Request denied	
	TRANSACTION_ERROR	9001 Request denied	In case that an updating method within an atomic session can not be executed because the underlying plugin is read-only or does not support atomic writing. - such as DmtSession#setNodeValue()/createLeafNode().
		9002 Internal error	In case that a commit operation at the end of an atomic session failed because one of the underlying plugins failed to close. - such as DmtSession#close() / commit()
	UNAUTHORIZED	9001 Request denied	
	URI_TOO_LONG	9005 Invalid parameter name	

DmtIllegalStateException	None	9002 Internal error	
SecurityException	None	9001 Request denied	
Other Exceptions	None	9002 Internal error	

6.5.1.2 Mapping for handling AddObject or DeleteObject RPC

If the PA encounters any error for handling AddObject or DeleteObject RPC, tThe CWMP Fault code specified in Table 6 should be used.

[Done] Bug#1796

Table 6:*Mapping of Exception code to TR-069 Fault Code for handling AddObject or DeleteObject RPC*

Class	DmtException code	TR-069 Fault code	Remarks
DmtException	ALERT_NOT_ROUTED	no recommendation	A protocol adapter is not expected to receive this Exception.
	COMMAND_FAILED	9002 Internal error	
	COMMAND_NOT_ALLOWED	9003 Invalid arguments	
	CONCURRENT_ACCESS	9002 Internal error	In case of atomic sessions, this Exception is thrown if the commit operation failed because of some modification outside the scope of the DMT to the nodes affected in the session.
	DATA_STORE_FAILURE	9002 Internal error	
	FEATURE_NOT_SUPPORTED	9002 Internal error	
	INVALID_URI	9005 Invalid parameter name	
	METADATA_MISMATCH	9003 Invalid arguments or 9004 Resource exceeded	Only for AddObject, if a DmtSession#createXxxNode(node) throws METADATA_MISMATCH, PA should check whether the MetaNode#getMaxOccurence() <= the size of array of DmtSession#getChildNodeNames() against the parent node. If true, 9004 should be chosen for the CWMP fault code. Otherwise, 9003 should be chosen.
	NODE_ALREADY_EXISTS	9002 Internal error	Only for AddObject RPC. A protocol adapter is not expected to receive this for DeleteObject RPC.
	NODE_NOT_FOUND	9005 Invalid parameter name	
	PERMISSION_DENIED	9008 Attempt to set non-writable	

		parameter	
	REMOTE_ERROR	no recommendation	A protocol adapter is not expected to receive this Exception.
	ROLLBACK_FAILED	9002 Internal error	
	SESSION_CREATION_TIMEOUT	9001 Request denied	
	TRANSACTION_ERROR	9001 Request denied	In case that an updating method within an atomic session can not be executed because the underlying plugin is read-only or does not support atomic writing. - such as DmtSession#createInteriorNode() /deleteNode() .
		9002 Internal error	In case that a commit operation at the end of an atomic session failed because one of the underlying plugins failed to close. - such as DmtSession#close() / commit()
	UNAUTHORIZED	9001 Request denied	
	URI_TOO_LONG	9005 Invalid parameter name	
DmtIllegalStateException	None	9002 Internal error	
SecurityException	None	9001 Request denied	
Other Exceptions	None	9002 Internal error	

6.5.1.3 Mapping for handling GetParameterValues or GetParameterNames RPC

If the PA encounters any error for handling GetParameterValues or GetParameterNames RPC, The CWMP Fault code specified in Table 7 should be used.

[Done] Bug#1796

Table 7:*Mapping of Exception code to TR-069 Fault Code for handling GetParameterValues or GetParameterNames RPC*

Class	DmtException code	TR-069 Fault code	Remarks
DmtException	ALERT_NOT_ROUTED	no recommendation	A protocol adapter is not expected to receive this Exception.
	COMMAND_FAILED	9002 Internal error	
	COMMAND_NOT_ALLOWED	9003 Invalid arguments	
	CONCURRENT_ACCESS	no recommendation	A protocol adapter is not expected to receive this Exception for either GetParameterValues or

			GetParameterNames RPC.
	DATA_STORE_FAILURE	no recommendation	A protocol adapter is not expected to receive this Exception.
	FEATURE_NOT_SUPPORTED	9002 Internal error	
	INVALID_URI	9005 Invalid parameter name	
	METADATA_MISMATCH	9003 Invalid arguments	
	NODE_ALREADY_EXISTS	no recommendation	A protocol adapter is not expected to receive this Exception for either GetParameterValues or GetParameterNames RPC.
	NODE_NOT_FOUND	9005 Invalid parameter name	
	PERMISSION_DENIED	No Fault (Continue operation for other nodes)	ACL can be used to restrict the visible data model from the ACS. That's the reason to continue the GetParameterNames and GetParameterValues operation in this case.
	REMOTE_ERROR	no recommendation	A protocol adapter is not expected to receive this Exception.
	ROLLBACK_FAILED	no recommendation	A protocol adapter is not expected to receive this Exception for either GetParameterValues or GetParameterNames RPC.
	SESSION_CREATION_TIMEOUT	9001 Request denied	
	TRANSACTION_ERROR	no recommendation	A protocol adapter is not expected to receive this Exception for either GetParameterValues or GetParameterNames RPC.
	UNAUTHORIZED	9001 Request denied	
	URI_TOO_LONG	9005 Invalid parameter name	
DmtIllegalStateException	None	9002 Internal error	
SecurityException	None	9001 Request denied	
Other Exceptions	None	9002 Internal error	

6.5.2 SOAP Fault expression

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

~~If the SOAP fault does not contain SetParameterValuesFault, the protocol adapter must set a code and a message contained in a DmtException instance to a FaultCode element and a FaultString element in the SOAP Fault response, respectively. The code in the DmtException must change the Fault code according to Table 5. Other information such as a path and causes included in the DmtException instance may be added to the FaultString element.~~

If the SOAP fault is raised by the SetParameterValues, and if the protocol adapter tries to include SetParameterValuesFault element, the protocol adapter must set path, code and message contained in a DmtException instance to ParameterName, FaultCode element and FaultString element in the SetParameterValuesFault element, respectively. The code in the DmtException must change the Fault code according to Table 5. If there are multiple DmtExceptions, each path of each DmtException must be described as one SetParameterValuesFault element. Other information such as cause information included in the DmtException instance may be added to the FaultString element included in the SetParameterValuesFault element.

6.6 Translation on node names

[Done] Bug#1772

1. TR-069 protocol adapter should keep the mapping table between instance ID for TR-069 ACS and the node name in DMT, and should translate them based on the table.
   a. If the node name can be parsed to integer larger than 0 and no more than the maximum value of unsignedInt, go to b. Otherwise go to c.[Done] Bug#1828#c4
   b. If the integer has never been assigned before, the node is mapped to the same integer. Otherwise go to c.
   c. The node name is mapped to the integer, larger than 0, which has never been assigned before.
   d. For GetParameterNames RPC or GetParameterValues RPC, if the TR-069 protocol adapter fails to map to some nodes because the integers more than 0 and no more than the maximum value of unsignedInt run out, it will log an error for all failed node names, and send GetParameterNamesReponse or GetParameterValuesResponse with all node names except failed ones back to the ACS. [Done] Bug#1828#c14
   e. When the TR-069 protocol adapter receives AddObject RPC, it will find the node name to be newly created according to the mapping table.
2. The table must be persistent and the assigned integer in the table must not be re-used.
3. The scope applied to the mapping mechanism is only an interior node whose MetaNode#getMaxOccurrence() > 1, in either case that the node name in DMT contains any character incompatible with TR-069 or not.
4. The table is prepared for each position in DMT. For example, if each MetaNode of "./A/B/<any>" and "./A/C/<any>" matches the condition specified in 3, protocol adapter should keep the table for each, separately.
5. How TR-069 protocol adapter manages the mapping table depends on the implementation of the protocol adapter. It could be set through Configuration Admin, or the protocol adapter listens DmtEvent and update the mapping table by him/herself.
6. An interior node whose MetaNode#getMaxOccurrence() =< 1 and a leaf node are out of scope of the mapping mechanism.

7. How TR-069 protocol adapter handles the node name in DMT which is not compatible with TR-069 depends on the implementation of the protocol adapter. An example is the node name including '.' (period).

6.7 Guidelines for Data Plugin Implementer

6.7.1 Session Type

A Data Plugin which manages writable nodes should support atomic sessions because TR-069 expects that the most operations are executed atomically.

6.7.2 FORMAT_XML and FORMAT_STRING

[DONE] Bug#1672#c42.

From just TR-069 perspective, the node which contains MetaNode with both FORMAT_XML and FORMAT_STRING is just as same as the node which contains **Meta**Node~~Data~~ with FORMAT_STRING because FORMAT_XML will never be selected for SetParameterValues RPC (see ~~6.2.1.2~~6.4.2 Case3) . However, the data model might be used for another remote management protocol or local managers. Therefore, it is up to the implementation of a data model (DataPlugin) what FORMAT a MetaNode of a node supports.

6.7.3 Design of a DMT Data Model to Support Monitoring of Instances Modification

TR-069 does not have notification functionality for telling the node creation or deletion to the ACS explicitly. That is why “A.B.NumOfXxx” node is prepared for the instance number of “A.B.<id>”. The “value changed” notification of the “A.B.NumOfXxx” node would tell the ACS to the creation or deletion of the instance. Therefore, from the point of view of TR-069, when a data model of DMT is designed, the designers are recommended to take it into account that the leaf node except the ones in a list subtree should not appear or disappear dynamically but exist unconditionally. [DONE] Bug1806#c4

6.7.4 Execute Operation

Because TR-069 protocol adapter will not call DmtSession#execute(), a Plugin does not need to implement execute operation from TR-069 perspective.

7 Javadoc

7.1 org.osgi.util.tr069
Class TR069ParameterValue

```
java.lang.Object
  org.osgi.util.tr069.TR069ParameterValue
```

```
public class TR069ParameterValue extends java.lang.Object
```

Class which contains value and data type for TR-069 parameter, and static methods of utilities.

Field Summary

static final java.lang.String	**TR069_TYPE_BASE64**
static final java.lang.String	**TR069_TYPE_BOOLEAN**
static final java.lang.String	**TR069_TYPE_DATETIME**
static final java.lang.String	**TR069_TYPE_HEXBINARY**
static final java.lang.String	**TR069_TYPE_INT**
static final java.lang.String	**TR069_TYPE_LONG**
static final java.lang.String	**TR069_TYPE_STRING**
static jfinal ava.lang.String	**TR069_TYPE_UNSIGNED_INT**
static final java.lang.String	**TR069_TYPE_UNSIGNED_LONG**

Constructor Summary

TR069ParameterValue(java.lang.String value, java.lang.String type) Constructor of TR-069 Parameter Value.	

Method Summary

static DmtData	**getDmtData**(java.lang.String value, java.lang.String tr069Type, java.lang.String valueCharsetName, java.lang.String nodeUri, MetaNode metaNode) Get DmtData to be used for DmtSession#setNodeValue() against a leaf node.
static DmtData[]	**getDmtDataForList**(java.lang.String value, java.lang.String tr069Type, java.lang.String valueCharsetName, java.lang.String nodeUri, MetaNode metaNode) Get DmtData array to be used for DmtSession#setNodeValue() against the child nodes of the specified node uri.
static TR069ParameterValue	**getTR069ParameterValue**(DmtData data) Get the TR069ParameterValue be used for GetParameterValuesResponse RPC
static TR069ParameterValue	**getTR069ParameterValueForList**(DmtData[] data) Get the TR069ParameterValue be used for GetParameterValuesResponse RPC
java.lang.String	**getType**()
java.lang.String	**getValue**()

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

7.1.1 TR069_TYPE_INT

public static final java.lang.String **TR069_TYPE_INT**

Constant representing the TR-069 integer type.

The value of TR069_TYPE_INT is “int”.

7.1.2 TR069_TYPE_UNSIGNED_INT

public static final java.lang.String **TR069_TYPE_UNSIGNED_INT**

Constant representing the TR-069 unsigned integer type.

The value of TR069_TYPE_UNSIGNED_INT is “unsignedInt”.

7.1.3 TR069_TYPE_LONG

```
public static final java.lang.String TR069_TYPE_LONG
```

Constant representing the TR-069 long type.

The value of TR069_TYPE_LONG is “long”.

7.1.4 TR069_TYPE_UNSIGNED_LONG

```
public static final java.lang.String TR069_TYPE_UNSIGNED_LONG
```

Constant representing the TR-069 unsigned long type.

The value of TR069_TYPE_UNSIGNED_LONG is “unsignedLong”.

7.1.5 TR069_TYPE_STRING

```
public static final java.lang.String TR069_TYPE_STRING
```

Constant representing the TR-069 string type.

The value of TR069_TYPE_STRING is “string”.

7.1.6 TR069_TYPE_BOOLEAN

```
public static final java.lang.String TR069_TYPE_BOOLEAN
```

Constant representing the TR-069 boolean type.

The value of TR069_TYPE_BOOLEAN is “boolean”.

7.1.7 TR069_TYPE_BASE64

```
public static final java.lang.String TR069_TYPE_BASE64
```

Constant representing the TR-069 base64 type.

The value of TR069_TYPE_BASE64 is “base64”.

7.1.8 TR069_TYPE_HEXBINARY

```
public static final java.lang.String TR069_TYPE_HEXBINARY
```

Constant representing the TR-069 hex binary type.

The value of TR069_TYPE_HEXBINARY is “hexBinary”.

7.1.9 TR069_TYPE_DATETIME

```
public static final java.lang.String TR069_TYPE_DATETIME
```

Constant representing the TR-069 date time type.

The value of TR069_TYPE_DATETIME is “dataTime”.

Constructor Detail

7.1.10 TR069ParameterValue

```
public TR069ParameterValue(java.lang.String value,
                           java.lang.String type)
```

Constructor of TR-069 Parameter Value.

Parameters:

`value` - value to be used. It can be either lexical or canonical representation as defined by XML Schema[9].
`tr069DataType` - data type defined in TR-069.

Throws:

`java.lang.IllegalArgumetException` - if value is null, if type is null, or if type is not any of defined as the TR069ParameterValue Constants.

Method Detail

7.1.11 getValue

```
public java.lang.String getValue()
```

Get the value this TR-069 Parameter value contains.

The value must be canonical representation as defined by XML Schema[9]., e.g. upper-case Hex letters in hexBinary, and no leading zeroes in numeric values.

7.1.12 getType

```
public java.lang.String getType()
```

Get the type this TR-069 Parameter value contains.

It must not be null and must be one of constants defined in TR069ParameterValue.

7.1.13 getDmtData

```
public static DmtData getDmtData(java.lang.String value,
                                 java.lang.String tr069Type,
                                 java.lang.String valueCharsetName,
                                 java.lang.String nodeUri,
                                 MetaNode metaNode)
                          throws TR069MappingException,
                                 java.lang.IllegalArgumentException,
                                 java.io.UnsupportedEncodingException
```

Convert the TR-069 value to a DmtData.

Case A: the tr069Type equals TR069ParameterValue.TR069_TYPE_BOOLEAN.

1. If the metaNode supports FORMAT _BOOLEAN,
   a. if the value equals “0” or “false”, return DmtData.FALSE_VALUE.
   b. if the value equals “1” or “true”, return DmtData.TRUE_VALUE.
   c. Otherwise, `IllegalArgumentException` will be thrown.
2. Otherwise, `ITR069MappingException` will be thrown.

Case B: the tr069Type equals TR069ParameterValue.TR069_TYPE_INT.

1. If the metaNode supports FORMAT_INTEGER,

    a. if the value can be interpreted as an integer, create a new DmtData by DmtData(int) with the interpreted integer and return it.

    b. Otherwise, `IllegalArgumentException` will be thrown.

2. Otherwise, `ITR069MappingException` will be thrown.

Case C: the tr069Type equals TR069ParameterValue.TR069_TYPE_UNSIGNED_INT.

1. If the metaNode supports FORMAT_UNSIGNED_INTEGER,

    a. Create a new DmtData by DmtData(String, int) with the of FORMAT_UNSIGNED_INTEGER If succeeded, the DmtData will be returned.

    b. Otherwise, `IllegalArgumentException` will be thrown.

2. Otherwise, `TR069MappingException` will be thrown.

Case D: the tr069Type equals TR069ParameterValue.TR069_TYPE_UNSIGNED_LONG.

1. If the metaNode supports FORMAT_UNSIGNED_LONG,

    a. Create a new DmtData by DmtData(String, int) with the of FORMAT_UNSIGNED_LONG. If succeeded, the DmtData will be returned.

    b. Otherwise, `IllegalArgumentException` will be thrown.

2. Otherwise, `TR069MappingException` will be thrown.

Case E: the tr069Type equals TR069ParameterValue.TR069_TYPE_LONG.

1. If the metaNode supports FORMAT_LONG,

    a. Create a new DmtData by DmtData(String, int) with the of FORMAT_LONG. If succeeded, the DmtData will be returned.

    b. Otherwise, `IllegalArgumentException` will be thrown.

2. Otherwise, `TR069MappingException` will be thrown.

Case F: the tr069Type equals TR069ParameterValue.TR069_TYPE_DATETIME.

1. If the metaNode supports FORMAT_DATETIME,

    a. Create a new DmtData by DmtData(String, int) with the of FORMAT_DATETIME. If succeeded, the DmtData will be returned.

b. Otherwise, `IllegalArgumentException` will be thrown.

2. Otherwise, `TR069MappingException` will be thrown.

Case G: the tr069Type equals TR069ParameterValue.TR069_TYPE_BASE64.

1. If the metaNode supports FORMAT_BASE64, go to Step 1.a., otherwise, go to Step 2.

   a. The specified value will be converted into base64 byte array and create a new DmtData by DmtData(byte[], boolean) with “true” as the second argument.

2. If the metaNode supports FORMAT_RAW_BINARY, go to Step 2.a., otherwise, go to Step 3.

   a. The array of raw format names is retrieved by MetaNode.getRawFormatNames().

   b. If the returned array has size zero, or the first element of it is null or empty String, `IllegalArgumentException` will be thrown. Otherwise, go to Step 2.c

   c. The value will be encoded into a sequence of bytes using the character set specified. If the encoding fails, go to Step3. Otherwise, then a new DmtData will be created by DmtData(String, byte[]) with the first elements in the array of raw format names as the first argument.

3. `TR069MappingException` will be thrown.

Case H: the tr069Type equals TR069ParameterValue.TR069_TYPE_HEXBINARY.

1. If the metaNode supports FORMAT_HEX_BINARYT, go to Step 1.a., otherwise, go to Step 2.

   a. The specified value will be converted into hexBinary byte array.

   b. A new DmtData will be created by DmtData(byte[], int) with FORMAT_HEX_BINARY as the second argument and be returned.

2. If the metaNode supports FORMAT_BINARY, go to Step 2.a., otherwise, `IllegalArgumentException` will be thrown.The value will be encoded into a sequence of bytes using the character set specified. If the encoding fails, go to Step3. Otherwise, then a new DmtData will be created by DmtData(byte[], int) with FORMAT_BINARY as the second argument and be returned.

3. `TR069MappingException` will be thrown.

Case I: the tr069Type equals TR069ParameterValue.TR069_TYPE_STRING.

1. If the node supports FORMAT_NULL, go to Step 1.a., otherwise, go to Step 2.

a. If the value is empty String, then return DmtData.NULL_VALUE. Otherwise go to Step 2.

2. If the node supports FORMAT_FLOAT, go to Step 2.a., otherwise, go to Step 3.

   a. If the parsing the value as float succeeds, then FORMAT_FLOAT is chosen. Otherwise go to Step 3.

3. If the node supports FORMAT_DATE, go to Step 3.a., otherwise, go to Step 4.

   a. If the constructing DmtData(String, int) with FORMAT_DATE as the first argument succeeds, then the DmtData will be returned. Otherwise go to Step 4.

4. If the node supports FORMAT_TIME, go to Step 4.a., otherwise, go to Step 5.
   a. If the constructing DmtData(String, int) with FORMAT_TIME as the first argument succeeds, then the DmtData will be returned. Otherwise go to Step 5.
5. If the node supports FORMAT_NODE_URI, go to Step 5.a., otherwise, go to Step 6.
   a. TR-069 Protocol Adapter translates the value to absolute URI, if the value is relative path reference, which starts with "./". The specified nodeUri is used for the translation.
   b. If the constructing DmtData(String, FORMAT_NODE_URI) succeeds, then the DmtData will be returned. Otherwise go to Step 6.
6. If the node supports FORMAT_RAW_STRING, go to Step 6.a., otherwise, go to Step 7.

   a. The TR-069 Protocol Adapter gets array of raw format names by MetaNode.getRawFormatNames().

   b. If the returned array has size zero or the first element of it is null or empty String, go to Step 7. Otherwise,go to Step 6.c.

   c. The DmtData is created by DmtData(String, String) with the first element in the returned array as the first argument. Then the DmtData will be returned.

7. If the node supports FORMAT_STRING, go to Step 7.a., otherwise, go to Step 8.

   a. The DmtData is created by DmtData(String). Then the DmtData will be returned.

8. If the node supports FORMAT_XML, go to Step 8.a., otherwise, go to Step 9.

   a. The DmtData is constructed by DmtData(String, int) with FORMAT_XML as the first argument. Then the DmtData will be returned.

9. `IllegalArgumentException` will be thrown.

Parameters:

`value` - value to be set to the specified node uri. It must not be null and can be lexical representation as defined by XML Schema.
`tr069Type` - TR069 data type. It must not be null or empty.
`valueCharsetName` - character set to be used for the value. If null, a platform default character set will be used. The character set is used for getting byte array from the specified value.
`nodeUri` - URI of the leaf node in DMT. It must be valid absolute DMT URI.
`metaNode` - Meta node of the specified node uri. It must not be null.

Returns:

DmtData to be used for DmtSession#setNodeValue(). It cannot be null.

Throws:

`TR069MappingException` - if creating DmtData is failed for some reasons.
`java.io.UnsupportedEncodingException` - if the specified character set name is not supported.
`java.lang.IllegalArgumetException` - if value is null, if tr069Type is either null or empty, if nodeUri is invalid absolute DMT URI, or if metaNode is null.

7.1.14 getDmtDatasForList

```
public static DmtData[] getDmtDataForList(java.lang.String value,
                                          java.lang.String tr069Type,
                                          java.lang.String valueCharsetName,
                                          java.lang.String nodeUri,
                                          MetaNode metaNode)
                                   throws TR069MappingException,
                                          java.lang.IllegalArgumentException,
                                          java.io.UnsupportedEncodingException
```

Convert the TR-069 value to a DmtData array for a list subtree.

The node specified by the node uri must be interior node which has the node type of DmtConstant.DDF_LIST_SUBTREE..

The value must be parsed by using a separator of comma. The each item parsed must be unescaped as specified in TR-106 and unescaped item will be converted into a DmtData. For the conversion, TR069ParameterValue.getDmtData() can be used The returned array will have the converted DmtData for each item.

Parameters:

`value` - value to be set to the specified interior node. It must not be null and can be lexical representation as defined by XML Schema.
`tr069Type` - TR069 data type. It must not be null or empty.
`valueCharsetName` - character set to be used for the value. If null, a platform default character set will be used. The character set is used for getting byte array from the specified value.
`nodeUri` - URI of the interior node in DMT. It must be valid absolute DMT URI.
`metaNode` - Meta node of the child leaf node of the specified node uri. It must not be null.

Returns:

array of DmtData to be used for DmtSession#setNodeValue() against the child nodes of the specified node uri. It can be empty array but cannot be null.

Throws:

`TR069MappingException` - if creating DmtData is failed for some reasons.
`java.io.UnsupportedEncodingException` - if the specified character set name is not supported.
`java.lang.IllegalArgumetException` - if value is null, if tr069Type is either null or empty, if nodeUri is invalid absolute DMT URI, or if metaNode is null.

7.1.15 getTR069ParameterValueForList

```
public static TR069ParameterValue getTR069ParameterValueForList(DmtData[] data)
```

Convert the array of DmtData to a TR-069 value for a list subtree.

1. For each item of the specified array, the following will be done:
   a. Convert the DmtData object to a TR069ParameterValue according to the rules in . gtTR069ParameterValue(DmtData).
   b. Retrieve the value as String from the converted TR069ParameterValue.
   c. Escaping must be done as specified by TR-106.
2.
3. Creating the comma-separated list by concatenating the retrieved values in the order of the appearance of the array.
4. Construct new TR069ParameterValue with “string” and the created comma-separated list, and return it.

This method does not check whether the all items of the specified array have the same format. [Done]Bugzilla#1739#c30.

Parameters:
`data` – Array of DmtData retrieved from DMT.
Returns:
Converted TR-069 value for list subtree.

7.1.16 getTR069ParameterValue

```
public static TR069ParameterValue getTR069ParameterValue(DmtData data)
```

Convert the DmtData to a TR069 value.

1. In case that the format of the data is FORMAT_BASE64,
   a. The byte array returned by DmtData.getBase64() will be converted into base64. The result of the conversion will be used as the value.
   b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_BASE64 and the value, and return it.

2. In case that the format of the data is FORMAT_BINARY,

   a. The byte array returned by DmtData.getBinary() will be converted into hexBinary in canonical representation as defined in XML Schema. The result of the conversion will be used as the value.

   b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_HEXBINARY and the value, and return it.

3. In case that the format of the data is FORMAT_HEXBINARY,

   a. The byte array returned by DmtData.getHexBinary() will be converted into hexBinary in canonical representation as defined in XML Schema. The result of the conversion will be used as the value.

   b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_HEXBINARY and the value, and return it.

4. In case that the format of the data is FORMAT_BOOLEAN,

   a. If DmtData.getBoolean() returns Boolean.TRUE or the DmtData equals DmtData.TRUE_VALUE, the value will be “1”. Otherwise the value will be “0”.

   b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_BOOLEAN and the value, and return it.

5. In case that the format of the data is FORMAT_DATE,

   a. DmtData.toString() or DmtData.getDate() will be used as the value.

   b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_STRING and the value, and return it.

6. In case that the format of the data is any of FORMAT_FLOAT, FORMAT_INTEGER, FORMAT_LONG, and FORMAT_NULL,

   a. DmtData.toString() will be used as the value.

   b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_STRING and the value, and return it.

7. In case that the format of the data is FORMAT_STRING.

   a. DmtData.toString() or DmtData.getString() will be used as the value.

   b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_STRING and the value, and return it.

8. In case that the format of the data is FORMAT_RAW_STRING,

a. DmtData.toString() or DmtData.getRawString() will be used as the value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_STRING and the value, and return it.

9. In case that the format of the data is FORMAT_INTEGER,

a. DmtData.toString() will be used as the value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_INT and the value, and return it.

10. In case that the format of the data is FORMAT_LONG,

a. DmtData.toString() will be used as the value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_LONG and the value and return it.

11. In case that the format of the data is FORMAT_TIME,

a. DmtData.toString() or DmtData.geetTime() will be used as the value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_STRING and the value and return it.

12. In case that the format of the data is FORMAT_XML,

a. DmtData.toString() or DmtData.getXml() will be used as the value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_STRING and the value and return it.

13. In case that the format of the data is FORMAT_UNSIGNED_INTEGER,

a. DmtData.toString() or DmtData.getUnsignedInteger() will be used as the value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_UNSIGNED_INT and the value, and return it.

14. In case that the format of the data is FORMAT_UNSIGNED_LONG,

a. DmtData.toString() or DmtData.getUnsignedLong() will be used as the value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_UNSIGNED_LONG and the value and return it.

15. In case that the format of the data is FORMAT_DATETIME,

a. DmtData.toString() or DmtData.getDateTime() will be used as the value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_DATETIME and the value, and return it.

16. In case that the format of the data is FORMAT_NODE_URI,

a. The returned String by DmtData.toString() or DmtData.getNodeUri() will be translated into TR-069Path. The result of the translation will be used as value.

b. Construct new TR069ParameterValue with TR069ParameterValue.TR069_TYPE_STRING and the value, and return it.

Parameters:
`data` - DmtData retrieved from DMT.
Returns:
Converted TR069 value.

7.2 org.osgi.util.tr069
Class TR069URI

```
java.lang.Object
  org.osgi.util.tr069.TR069URI
```

```
public class TR069URIextends java.lang.Object
```

Utility class for translating between DMT URI and TR-069 Path.

Method Summary

`static java.lang.String`	**`getDmtUri`**`(java.lang.String tr069Path)` Get the URI in DMT corresponding to the specified path in TR-069.
`static java.lang.String`	**`getTR069Path`**`(java.lang.String uri)` Get the TR069 path corresponding to the specified URI in DMT.
`static boolean`	**`isValidTR069Path`**`(java.lang.String tr069Path)` Validate whether the specified path is in the format of TR069 path.

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Method Detail

7.2.1 getDmtUri

```
public static final java.lang.String getDmtUri(java.lang.String tr069Path)
```

Get the URI in DMT corresponding to the specified path in TR-069. The specified TR-069 path should be valid. This method will not validate it. If the specified path ends with period ".", it means a Partial path. In that case, the conversion will be done after the last period is removed.

Parameters:
`tr069Path` - TR-069 path
Returns:
URI in DMT, which corresponds to the specified TR-069 path.

Throws:
`java.lang.IllegalArgumetException` - if tr069Path is null.

7.2.2 getTR069Path

```
public static final java.lang.String getTR069Path(java.lang.String uri)
```

Get the TR069 path corresponding to the specified URI in DMT. The specified URI should be valid absolute URI. This method will not validate it.

Parameters:
`uri` - URI in DMT.
Returns:
path in TR-069, which corresponds to the specified URI.

Throws:
`java.lang.IllegalArgumetException` - if uri is null.

7.2.3 isValidTR069Path

```
public static final boolean isValidTR069Path(java.lang.String tr069Path)
```

Validate whether the specified path is in the format of TR069 path.

Parameters:
`tr069Path` – TR-069 path.
Returns:
true if the specified path is in the format of TR069 path. Otherwise, false.

8 Considered Alternatives

8.1 Future work for notification handling

As described in Section 4.2.1, in order to realize TR-069 based notification especially regarding internal changes in OSGi, what Data Plugins and TR-069 protocol adapter, respectively, needs to do should be clarified. However, those are out of scope of this RFC, because the much further discussion will be required for the clarification.

8.2 Future work for forced inform parameters

Some data model specified by BBF, such as TR-098, defines some nodes as forced inform parameters. The values of the nodes must be included in the ParameterList argument of every Inform message to the ACS. How to realize it in OSGi should be clarified. However, those are out of scope of this RFC, because the much further discussion will be required for the clarification.

8.3 Future work for Set/GetParameterAttributes RPC

How a TR-069 protocol adapter should handle SetParameterAttributes RPC and GetParameterAttributes RPC is out of scope of this RFC, because the much further discussion will be required for the clarification.

9 Security Considerations

All security requirements follow the DMT Admin specification.

10Document Support

10.1 References

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

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

[3]. 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

[5]. CPE WAN Management Protocol, TR-069 amendment 2, December 2007. http://www.broadband-forum.org/technical/download/TR-069Amendment2.pdf

[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

[8]. OSGi Alliance, OSGi Service Platform Mobile Specification Release 4, Version 4.0, July 2006

[9]. XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

10.2 Author's Address

Name	Koya Mori
Company	NTT Corporation
Address	Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan
Voice	+81-46-859-3446
e-mail	mori.kouya@lab.ntt.co.jp

Name	Ikuo Yamasaki
Company	NTT Corporation
Address	Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan
Voice	+81-46-859-8537
e-mail	yamasaki.ikuo@lab.ntt.co.jp

10.3 Acronyms and Abbreviations

10.4 End of Document