RFC 85 Device Management

Confidential, Draft

77 Pages

Abstract

0 Document Information

0.1 Table of Contents

All Page Within This Box

0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

`Source code is shown in this typeface.`

0.3 Revision History

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

Revision	Date	Comments
Initial few versions	2004	Vadim Draluk, Motorola, vdraluk@motorola.com
V0.6	2004.11.20	Added Javadoc. Major editorial changes in chapter 5. Balazs Godeny, Nokia, balazs.godeny@nokia.com
V0.7	2004.12.07	Comments from Vadim Draluk and Bill Gengler addressed. Major changes in chapter 5 and 6. Balazs Godeny

All Page Within This Box

Revision	Date	Comments
V0.8	2004.12.15	Additions in chapter 2 and 3 Overlappping plugins are allowed, chapter 5 Dmt Event descriptions added. Moved here from rfc 97.
V0.9	2005.02.10	Corrections based on Peter Kriens comment after first version of spec. Major changes: - DmtDataType removed, fields moved to DmtData - MIME info from DmtData moved to Dmt and DmtReadOnly - DmtMetaNode. - referential integrity related methods removed - getNameRegExp added - getValidNames added - Exceptions specified for each method in Javadoc Balazs Godeny
V0.10	2005.02.25	More corrections after reviewing the draft spec. Major changes: - DmtSession - state constants and getState added - getEffectiveNodeAcl added - DmtException - constructor to sign fatal state and corresponding getter added - DmtAcl is made immutable, added constructor to receive array of principals and permissions - defined canonical string representation - DmtAlertItem - getMark removed - DmtAlertSender interface removed, alert sending methods grouped into a single form and moved to DmtAdmin - Dmt: added commit() - Clarified that ACL checking is done in the Admin Balazs Godeny

1 Introduction

OSGi is a generic, service-centric execution environment. It specifies a generic framework and a core set of service interfaces that enable delivery of multiple value added service implementations, potentially from different vendors.

All Page Within This Box

From its very inception OSGi had an architecture for remote management in place. It was expected to be encapsulated in the Management Agent contained in and deployed as the Management Bundle. The management of an OSGi-based environment is performed by a Remote Manager, communicating with the Management Agent over a mutually agreed protocol.

This capability always was and still remains pivotal for an environment as dynamic and as flexible as OSGi, allowing provisioning of new services and replacement of existing ones practically on the fly. The proprietary nature of the protocol between the Agent and the Remote manager was not an issue for verticals and business models where the devices in the field were likely to be associated with a single service center, which was in full control of all devices' configuration. Nor was the absence of standardized management data structures and APIs describing how such information is presented and manipulated.

With OSGi entering the mobile devices arena this situation changes dramatically. Now same server products can be expected to be deployed by multiple carriers, and work with mobile devices from several manufacturers. The software on the devices can be sourced not only from manufacturers and carriers, but increasingly from third-party developers. This software, unlike early days of Java on mobile phones, can go far beyond simple games, and can constitute a part of the device's own infrastructure. Such software is likely to require uniform access to the device management information over a simple and consistent API.

This document introduces such an API, as well as an infrastructure of plug-ins necessary to make device management highly dynamic and extensible.

2 Application Domain

With the capacity of handsets growing to match that of desktop computers from 7-8 years ago, the software that runs on these devices is evolving in several significant ways:

- Instead of monolithic firmware it is now increasingly based on more general and componentized operating systems, such as Linux, Symbian and Windows CE, featuring downloadable and replaceable applications
- The life cycle of handset software is becoming more complex, with both basic and application components subject to:
  - Dynamic download and installation
  - Updates
  - Repairs
  - Re-configuration
  - Inter-dependencies
- Multiple players are now involved in administration and configuration of software:
  - Manufacturer
  - Distributor
  - Operator
  - Third-party software vendor
  - Enterprise
  - Handset owner

This change in the handset software paradigm is driving the need to address the architecture of device management (DM) solutions in a way that accounts for the shift, and provides for a complete architecture encompassing:

- A comprehensive meta-data model to accommodate all types of DM-related information
- A generic DM engine, responsible for all manipulations of configuration and management data
- A set of APIs, both Java and native, to be used by all components accessing and manipulating DM and provisioning data

The most important decision in determining any fundamentally new architecture is choosing a single meta-data model that expresses a common conceptual and semantic view for all consumers of that architecture. In the case of systems management, of which mobile device management is a specific case, there are a number of meta-data models to choose from:

- SNMP [3], the best-established and most ubiquitous model for network management
- JMX [4], a generic systems management model for Java, a de-facto standard in J2EE management
- Federated Management Architecture (FMA) [5], another Java standard originating in storage management
- Common Information Model and Web-Based Enterprise Management (CIM/WEBM) [6], a rich and extensible management meta-model developed by DMTF

For various reasons none of these models enjoy any significant mindshare within the mobile device community. Some, like SNMP, are primitive and very limited in functionality. Some, such as JMX and FMA, are too Java-centric and not well-suited for mobile devices.

One model that appears to have gained an almost universal acceptance is the Device Management Tree (DMT) [8,9], introduced in support of the SyncML DM (now OMA DM) protocol [7]. Hierarchical like SNMP, it is more sophisticated in the kinds of operations and data structures it can support, as well as designed with the restrictions and specifics of mobile devices in mind. The standard-based features of the DMT model are:

- The DMT is a tree of interior and leaf nodes
- All nodes in the data tree have names
- Nodes in the meta-tree can be defined as children of a particular node, with no specific names defined
- Only leaf nodes have values
- Base value types (called “formats” in the standard) are
    - Integer
    - String
    - Boolean
    - Binary
    - XML
- Leaf nodes in the meta-tree can have default values
- Nodes in the meta-tree have allowed access operations defined (GET, ADD, REPLACE, DELETE and EXEC)
- Nodes in the data tree can have ACLs defined, associating operations generally allowed for nodes with particular identities

The DMT model as described above is rather primitive, and does not allow for any meaningful enforcement of the data integrity. To overcome this difficulty, additional meta-attributes refining semantics of both interior and leaf nodes have been introduced:

- Max/min values for integers
- Max length for string values and binary values
- Valid values lists

- Regular expressions to be matched for string values
- Similar constraints (max length, valid values list and regular expression) for node names

Based on its industry acceptance and technical features, DMT was chosen as the central and uniform meta-data and operational model for MEG DM. In this capacity it is considered separately and independently from OMA DM or any other provisioning protocol, and underlies all local and remote DM operations on the device.

3 Problem Description

The choice of a meta-data model further drives the programmatic interfaces and features used to access the DM data and to integrate with other services:

- Read-write management data access for applications and agents
- Read-only meta-data access for applications, allowing front-end testing of data validity
- Privileged ability to specify the principal on whose behalf DM data access is performed
- Access control management
- Parallel access to the DM data by several applications
- Events posted in case of an update of DM data
- User-initiated management server session support for monitoring and data delivery

The interfaces and features listed are defined and discussed in the current document.

4 Requirements

5 Technical Solution

5.1 Entities

The following figure shows the entities that have a role in the device management solution.

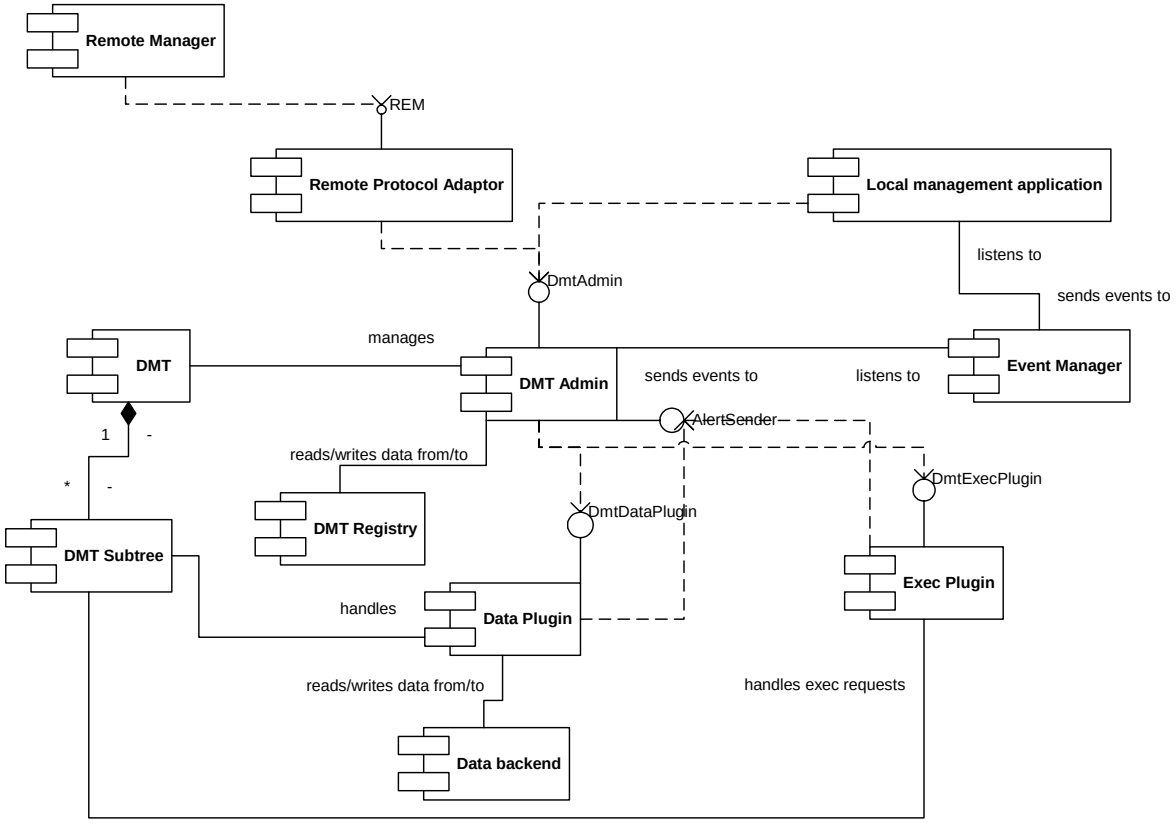

Figure 1 Entities of Device Management

A short description of the entities follows. The modules and interfaces which are specified in this document are highlighted in bold font. Detailed description of other entities can be found in [10].

DMT	The Device Management Tree, a logical entity representing all remotely manageable aspects of the device. The structure of the tree is specified in [14].
Dmt Admin	An OSGi service through which the DMT can be manipulated. It offers the **DmtAdmin** Java interface which is used by local administrator services and also by the protocol agents to issue DMT manipulation requests and queries. It also offers the **AlertSender** interface which can be used by the Dmt Plugins and by the local management applications to send a device initiated notification to the remote

All Page Within This Box

	management server. This flow of data is not shown on the picture.
Data plugins and Exec plugins	OSGi services which take the responsibility over a given subtree of the DMT. The Dmt Admin forwards DMT manipulation calls to Data plugins and Exec requests to Exec plugins. The Dmt Admin finds the appropriate plugins through the OSGi Service Registry (not shown on the picture). Note that the usage of plugins is not mandated, the Admin might be able to handle certain DMT calls on its own without forwarding them to plugins. Plugins can be built on top of applications, flat files or databases, to name only a few examples.
Data backend	The DMT is not necessarily a database, it is only a view of the management state of the device. The actual data is stored either in a plugin or in a backend service or in the Admin's own storage. An example for a data backend service is the Configuration Admin service: the data in the configuration data store is made available for reading and writing through the DMT.
DMT Registry	Alternatively (or in addition to) to using plugins to Dmt Admin might have it's own data storage, called DMT Registry or DMTR. In this case the Admin does not delegate certain calls to plugins, but it is able to provide the data on it's own. The work distribution between DMTR and plugins is implementation specific, in certain implementations DMTR might not exist at all, in others it may be the only single source of DMT data. There is no entity on the API which corresponds to the notion of DMTR.
Local management application	A bundle which uses the DmtAdmin interface to read or manipulate the data in the DMT. Note that that most of the management features mapped to the DMT are also available using the API of some OSGi service, which abstracts the callers from the underlying details of the DMT structure. As an example, it is more straightforward to use the MonitorAdmin service than manipulate the monitoring subtree. The local management application might listen to Dmt Events if it is interested in updates in the tree made by other entities.
Remote manager, Remote Protocol Adaptor	A management server external to the device can issue DMT manipulation commands over some management protocol. The protocol to be used is not specified by OSGi, the most likely ones used in practice will be OMA DM and IOTA. The commands reach the service platform through the REM interface where the Remote Protocol Adaptor forwards the calls to the DmtAdmin through the interface defined in this document. The typical client of the DmtAdmin interface is a Remote Protocol Adaptor. The flow of events can also go the other way round. Plugins or local administrator applications can send an instant notification to the DMT Admin which will send the notification (alert in OMA DM terminology) to the protocol adaptor which will forward it to the remote manager. This document does not specify how the Dmt Admin finds the Protocol Adaptor to forward the alert.

5.2 Basics

Users of this specification should be familiar with the concept of the Device Management Tree (DMT) and its properties and operations as defined by OMA DM specifications [8].

This specification is based on the concept of a Dmt Admin service which provides a Java API for manipulating data in the DMT of a mobile device. Manipulation means issuing commands specified by the OMA DM protocol: the Get, Replace, Add, Delete and Exec commands are available. Access to the DMT is session based. The sessions are concurrent and transactional with the following limitations:

- Two or more sessions can not update the same part of the tree simultaneously. An updating session acquires an exclusive lock on the subtree which blocks the creation of other sessions within the same subtree.
- There can be any number of concurrent read only sessions, but the ongoing read only sessions will block the creation of an updating session within the same subtree.
- Supporting a two phased transactional model is postponed to a future release of the specification. The lack of this support can lead to error situations which are described later in this document 5.7.3.

Although the DMT is a persistent data store with transactional access and without size limitations, the notion of DMT and DMT Registry should not be confused with a general purpose database. The intended purpose of the DMT is to provide a view of the management state of the device, this is what the DMT and the DMT API are designed for. Other kind of usage, like storing and sharing generic application specific data, is discouraged.

The API of the Dmt Admin service is based on the following interfaces and classes:

- The *DmtAdmin*, produces objects implementing the *DmtSession* interface, and offers a method for sending alerts.
- The *DmtSession*. Objects implementing this interface are produced by the *DmtAdmin*, and are associated with the principal on whose behalf the DMT access is performed. *DmtSession* objects are also responsible for concurrent DMT access.
- The *DmtReadOnly* and *Dmt* interfaces are providing DMT navigation and manipulation functionality. These are superinterfaces of the *DmtSession*, the end user does not directly interact with them.
- The *DmtMetaNode.* The interface encapsulating access to both OMA-DM-standard meta data information and its extensions introduced by MEG.
- The *DmtData* class encapsulates various types of leaf node values.
- The *DmtAcl* class represents DMT ACLs in a structured format.
- The *DmtDataPlugin, DmtReadOnlyDataPlugin* and *DmtExecPlugin* interfaces are abstracting away specific data structures underlying the DMT and operations that can be associated with it.
- The *DmtAlertSender* interface is used to send device initiated notifications (alerts) to the remote managers.
- The *DmtAlertItem* encapsulates the data sent in an alert.
- The *DmtException* is thrown whenever something goes wrong while executing a DMT manipulation command. It encapsulates various error codes defined by OMA and extensions defined in MEG.
- The *DmtPermission* guards against unauthorized local usage of the API. *DmtPrincipalPermission* guards against unauthorized creation of *DmtSessions* on behalf of remote servers.

All Page Within This Box

A class diagram for the DmtAdmin service is shown below.

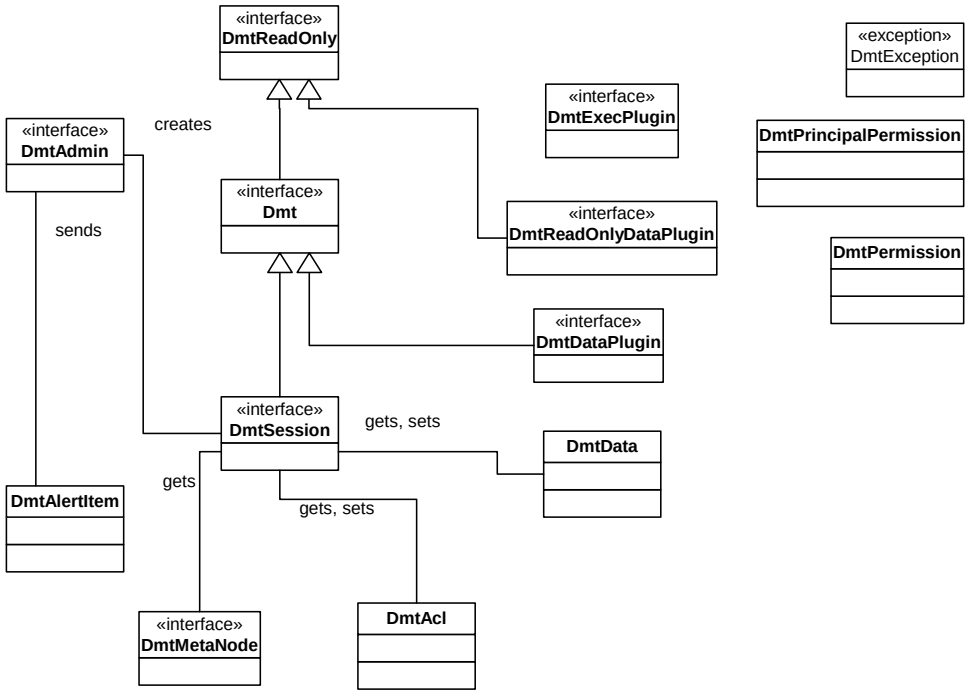

Figure 2 Class diagram

A short introduction to the interfaces and classes follows. For details see the Javadoc section (chapter 6).

5.3 The DmtAdmin interface

The *DmtAdmin* interface is the main entry point into the Device Management API, it is used to create *DmtSession* objects and to send alerts. The client of this API must obtain an instance of *DmtAdmin* from the OSGi service registry.

5.3.1 Creating sessions

There are three *getSession()* methods defined for the class. In its most complex form the method can take a String as a parameter which determines the principal on whose behalf the data manipulation is performed. See details later.

Another parameter is the subtreeUri string. If given as "./", it is assumed to mean the root of the DMT. In principle all nodes can be reached from the root, so specifying sub-tree is not necessary. However the implementation can use this information to optimize the concurrent tree access. It is allowed to open a session using a leaf node as session root.

The final parameter sets the locking mode for the tree. The constants for locking mode values are defined as part of the *DmtSession* interface, and are described later in this document.

```
DmtSession getSession(String subtreeUri) throws DmtException;
DmtSession getSession(String subtreeUri, int lockMode) throws DmtException;
DmtSession getSession(String principal, String subtreeUri, int lockMode)
    throws DmtException;
```

All Page Within This Box

These calls can block depending on the presence of other opened sessions within the same subtree. Because of the blocks the methods can time out. See details later.

5.3.2 Determining the principal

Remote Protocol Adaptors must use the third form of the *getSession* method which features a *principal* parameter. As can be seen on the sequence diagram in 5.13.1, in this call they can forward the server identification string to the DmtAdmin. This form of the *getSession()* method is guarded with *DmtPrincipalPermission,* this permission class is used to enforce that only trusted entities can act on behalf of Remote Management Servers. The server identification string is determined during the authentication process in a management protocol specific way. For example, the identity of the OMA DM server can be established during the handshake between the OMA DM agent and the server. In the simpler case of OMA CP protocol, which is a one-way protocol based on WAP Push, the identity of the agent can be fixed.

5.3.3 Locally initiated session

The other two forms of the *getSession* call are meant for local management applications where no *principal* string can be used. No special permission is defined to restrict the usage of these methods, however, as depicted in 5.13.2 the entities wanting to execute device management commands need to have the appropriate *DmtPermission*.

5.3.4 Concurrent sessions, locking

The *DmtAdmin* implementation is responsible for the tree or sub-tree locking. If the requested subtree is not accessible, the *getSession* call will get blocked till it becomes available again. In case the requested subtree is not available for an extended period of time, the method times out throwing an exception. The length of the period before time out exception is thrown is not specified by this document. Once *getSession* returns a session object, data access method calls can be performed on it, with the appropriate locking in place.

DmtAdmin implementations are allowed to lock the entire tree irrespective of a specific subtree URI specified in the *getSession* method. They are encouraged however to provide more fine-grained locking in order to improve overall system performance.

Although concurrent sessions are allowed on different subtrees, the *DmtAdmin* must make sure that concurrent updating sessions are not opened within the same plugin. Otherwise the plugin would be unable to determine on receiving a *commit(), rollback()* or *close()* call which session these calls relate to.

5.3.5 Sending alerts

DmtAdmin offers a feature of sending notifications to a remote management server. The typical client of this service is a Dmt Plugin which send an alert asynchronously after an exec operation, however, this feature might be used directly also by other clients such as local administrator applications. Examples for the latter are a repair package download application kicking off the repair package search (like in the case of the client-initiated FOTA CR), or the FOTA agent reporting the end of a repair operation, or a monitoring agent delivering some info to the server.

The

```
void sendAlert(String principal, int code, DmtAlertItem[] items) throws
DmtException;
```

call features an array of alert items to be sent,an alert code associated with the alert and a principal name representing the addressee of the alert. It is the Dmt Admin's responsibility to route the alert to the appropriate principal through the appropriate protocol adaptor. If the client can provide the principal name, it must do so.

Otherwise they can try calling the method without providing any hint for routing. It might still succeed for example in the case when the device has only one protocol adaptor which is connected to only one remote server.

5.4 The DmtSession, DmtReadOnly and Dmt interfaces

The *DmtSession* interface feature the following functionality:

- Node value access
- Tree manipulation through creation, renaming and deletion and execution of nodes
- Concurrent access control
- Transactionality

DmtSession extends *Dmt* and *DmtReadOnly*, the API client only interacts with *DmtSession*. The methods of *Dmt* and *DmtReadOnly* are used also in the plugin interfaces, see later. *DmtReadOnly* is separated out to allow for simpler implementation of read only plugins which have to implement only this interface.

5.4.1 Tree manipulation

Most tree manipulation methods accept a node URI as their first parameters, like in the following examples:

```
boolean     isLeafNode(String nodeUri) throws DmtException;
DmtAcl      getNodeAcl(String nodeUri) throws DmtException;
DmtMetaNode getMetaNode(String nodeUri) throws DmtException;
void        setNodeTitle(String nodeUri, String title) throws DmtException;
void        deleteNode(String nodeUri) throws DmtException;
```

Since a *DmtSession* object can represent both the entire tree and a sub-tree, the URI is either absolute (then starting with "./") or relative to the root of the sub-tree. This approach is different from one described by the OMA DM protocol spec, which allows to omit "./" at the beginning of the URIs, but it does not affect the interoperability in any way.

If the URI specified does not correspond to a legitimate node in the tree, typically an exception is thrown. The only exception is the *isNodeUri()* method which returns *false* in case of an invalid URI.

Separate methods are used to create interior and leaf nodes.

```
void createInteriorNode(String nodeUri) throws DmtException;
void createInteriorNode(String nodeUri, String type) throws DmtException;
void createLeafNode(String nodeUri) throws DmtException;
void createLeafNode(String nodeUri, DmtData value) throws DmtException;
void createLeafNode(String nodeUri, DmtData value, String mimetype)
     throws DmtException;
```

For a node to be created following conditions must be fulfilled:

- The URI of the new node has to be valid
- In the remote management case the principal of the *DmtSession* must have ACL-level permission to add the node. In the local case the entity issuing the call must have the necessary *DmtPermission*.
- The parent of the node must exist

All Page Within This Box

- All constraints must be obeyed

In case of leaf node creation the value of the new node is defined by the *DmtData* object can be passed as the second parameter. This class is described later. Omitting the DmtData argument means that creating a leaf node with default value is requested. If a node does not have a default value specified by it's meta data, this form of the *createLeafNode()* method will throw an exception.

5.4.2 Concurrent access and transactionality

A *DmtSession* can be created using the following lock modes:

- LOCK_TYPE_SHARED allows read-only access to the tree, but can be shared between multiple readers. That means that any number of simultaneous read only session can coexist.

- LOCK_TYPE_EXCLUSIVE lock guarantees full access to the tree, but can not be shared with any other locks. Not even read only sessions can coexist with a session opened using this lock mode.

- LOCK_TYPE_ATOMIC is an exclusive lock with transactional functionality. It means that operations in a session opened using this lock mode will either succeed together or fail together. See 5.7.3 for the details of transaction support in plugins.

Persisting the changes works differently for exclusive and atomic lock. Changes to the tree in an atomic session are not persisted until the *commit()* or *close()* method is executed. Changes in an exclusively locked session are persisted immediately. For the atomic sessions once an error is encountered, all successful changes are rolled back. *Close()* is responsible also for releasing the locks associated with the *DmtSession*. A failed *close()* operation also releases the session's locks. *Close()* must be called also at the end of read only sessions. No DMT manipulation operation is allowed on the session after a *close().*

The *close()* and *commit()* methods can be expected to fail even if all or some of the individual updates were successful. This can happen due to some multi-node semantic constraints defined by a specific implementation. For example, node A can be required to always have children A.B, A.C and A.D. If this condition is broken when *close()* or *commit()* is executed, the method can fail, and throw an exception. The details of how an implementation specifies such constraints is outside the scope of this specification.

The *commit()* call is a convenience method with which a series of calls can be committed in an atomic session without closing the session. It commits the changes made since the last commit call or since the session was opened.

5.4.3 Session time out and session state

A session times out and it is invalidated after an extended period of inactivity. The exact length of this period is not specified, it is recommended to be at least 1 minute and at most 24 hours. All methods of an invalidated session must throw InvalidStateException.

A session's state is one of the following: open, closed or invalid, as can be queried by the *getState()* call. The invalid state is reached either after a fatal error case is encountered or after the session is timed out. When an atomic session is invalidated, it is automatically rolled back.

5.5 The DmtMetaNode interface

The *DmtMetaNode* interface supports retrieval of meta-data both standard for OMA DM and defined by MEG (without breaking the compatibility) to provide for better DMT data quality in an environment where many software components manipulate this data. Setting meta data is not supported.

All Page Within This Box

The *DmtMetaNode* interface has two types of functions. One is used to retrieve OMA-DM-standard metadata, like in the case of the following methods.

```
boolean   can(int operation);
int       getScope();
String    getDescription();
```

The other type is used for meta-data extensions defined by MEG, such as valid values, regular expressions for node names and node values. Examples belonging to this category:

```
String    getPattern();
DmtData[] getValidValues();
```

The *getFormat()* method returns the node type (Format, in OMA DM terms) expressed in terms of type constants defined by the *DmtData*, described later in this document. If the node can take one of multiple possible formats at run time then the return value is the logical OR of these type constants.

The *getValidValues()* method returns an array of DmtData objects if valid values are defined for the node, or *null* otherwise.

As the meta data of a node in OSGi is richer than what is mandated by OMA DM, the OSGi managed objects can not be fully described by OMA DM's DDF (Device Description Framework). How the management server learns the OSGi management object structure is out of the scope of OSGi specifications.

5.6 The DmtData class

The *DmtData* class is used to create and update values of nodes in the DMT. This data structure represents only the value and the format property of the node, all other properties of the node (like MIME type) can be set and read using the *Dmt* and *DmtReadOnly* interfaces.

Different constructors are available to create nodes with different formats. Nodes of *null* format can be created using the static *DmtData.NULL_VALUE* constant instance of this class.

The *DmtData* class contains an enumeration of the formats a node in the DMT can have. An alternative to this approach would be to use the OSGi *MetaType* service facilities, but in order to preserve the possibility of applying this API to non-OSGi CDC-based environments we decided not to build on MetaTyping.

5.7 The DmtDataPlugin, DmtReadOnlyDataPlugin and DmtExecPlugin interfaces

The Dmt plugins take the responsibility of handling DMT manipulation commands within certain subtrees of the DMT. It is the responsibility of the Dmt Admin to forward the calls to the appropriate plugin. The only exception is the ACL manipulation commands: as ACLs are enforced on the Admin level and not on the plugin level, these commands are not forwarded to plugins.

Although the implementation of this specification must support plugins, their usage in handling the OSGi DMT ([14]) is not mandatory, it is possible to implement a Dmt Admin which handles the whole DMT. The plugins are deployed as OSGi services, the Dmt Admin must dynamically take the plugins into use. An example of a plugin can be one that handles the subtree which stores configuration data. When the client issues a DMT manipulation command which relates to this subtree (e.g. creation of a new node representing an item in a configuration dictionary), the Dmt Admin forwards the call to the plugin which translates the request to a method call on the ConfigurationAdmin service. In this example the plugin is a simple proxy to the configuration service, it provides the DMT view of the configuration data store.

As described before the Dmt Admin might have it's own data store (DMT Registry). It is implementation specific whether the Dmt Admin uses the plugin mechanism to implement the DMT Registry or not.

Plugins provide SPI-like features, the clients of the Device Management API do not interact with Dmt plugins directly and do not need to understand how they work. Access to the plugins must be available only for the *DmtSession* implementation. This can be enforced by the standard OSGi *ServicePermissions*.

There are two type of Dmt plugins: data and exec plugin. A data plugin is responsible for handling the node retrieval, addition and deletion commands, while the exec plugin handles the node execution commands. Data plugins also come in two flavours: the *DmtReadOnlyDataPlugin* supports only data retrieval while the *DmtDataPlugin* supports also modification of node values and tree structure. The data plugins contain most of the methods the *DmtSession* does. The *DmtReadOnlyDataPlugin* inherits them from *DmtReadOnly*, the *DmtDataPlugin* inherits from *Dmt*.

5.7.1 Subtrees handled by plugins

Each data plugin is associated with a DMT subtree, or a set of subtrees. The plugin to subtree association is described by the service registration property `dataRootURIs`, so that multiple URIs are represented by an array of string values. The plugins can therefore be found by the DmtAdmin using OSGi filtering mechanism. Certain implementations may choose to start all bundles containing plugins at startup time, or they can choose to start them on demand later.

Each EXEC plugin is also associated with one or several DMT subtrees. The plugin to subtree association is set by the registration property `execRootURIs`.

Only nodes having occurrence=1 in their metadata can be plugin roots. That is if a given type of node can occur in multiple instances with different names on the same level then a plugin can not be rooted at any of these nodes, it must be rooted at the common parent of these nodes.

Overlapping plugins are allowed, i.e. it is possible for PluginA to control the ./A subtree while PluginB controls ./A/B. It is the responsibility of the Dmt Admin to forward the calls to the appropriate plugin. If the call addresses a given `nodeUri` then the Admin must forward the call to that plugin which is registered to handle a subtree where `nodeUri` belongs, i.e. the subtree root URI is a prefix of `nodeUri`. If there are more than one such plugins then the Admin must choose the one with the longest subtree root URI that is the one which is the closest in the tree to `nodeUri`. In the example above a call to ./A/B/C will be handled by PluginB, a call to ./A/X will be handled by PluginA.

5.7.2 Meta data

Meta data can be supported at the engine level (i.e. in Dmt Admin), and its support by plugins is an optional (and advanced) feature. It can be used, for example, when a data plugin is implemented on top of a data store or another API that have their own metadata, such as a relational database, in order to avoid metadata duplication and inconsistency. The meta data specific to the plugin returned by the plugin's *getMetaNode()* method is complemented by the engine level meta data. If there are differences in the meta data elements known by the plugin and the Dmt Admin then the plugin specific elements take precedence.

5.7.3 Plugins and transactions

As the Device Management API is transactional, it has to be supported also in the data plugins. Exec plugins need not be transaction aware. When a *DmtSession* is created, committed or rolled back, the Dmt Admin forwards the transaction demarcation calls to the plugins which participate in the session. Both types of data plugins have an *open()* and *close()* method, the *DmtDataPlugin* also has *commit()* and *rollback()* methods. The *DmtDataPlugin* might or might not support atomic transactions.

Any operation in an atomic session will fail and the session will be rolled back if the underlying plugin does not support atomic transcations. This scenario can occur if an atomic session is successfully established but later an updating operation is attempted on a node which is served by a plugin with no atomicity support.

When the session is closed the *close()* operations will be forwarded to the plugins that participated in the session in the reverse order of that of the *open()* calls. The admin forwards the *close()* call to atomic plugins even if the last method was a *commit()* or *rollback().*

As mentioned before this transactional model will be improved in further releases of this specification by supporting a two phased commit model. The current design can lead to errors like in the following scenario:

PluginA and PluginB participate in the session. When the session is committed, PluginA commits successfully, but PluginB encounters an error and rolls back. Now the whole session should be rolled back but PluginA has already committed its changes.

5.8 The DmtAcl class

DmtAcl is an immutable class representing structured access to DMT ACLs which under OMA DM are defined as strings with an internal syntax. Instances of this class can be created by supplying a valid OMA DM ACL string as its parameter. The class features methods for getting, setting and modifying permissions for given principals as well as checking whether a given operation is permitted for a principal. The *toString()* method of this class gives a string representation where operations are in the following order: {Add, Delete, Exec, Get, Replace} and principal names are sorted alphabetically.

5.8.1 Maintaining the ACLs

As mentioned in 5.7 the ACLs are not maintained and not enforced at the plugin level. In cases where the data is provided by a plugin which proxies a service which has another API to manipulate the data, this may lead to inconsistency between the Admin's view of the ACLs and the underlying data.

As an example, consider a configuration dictionary which is mapped to a DMT node and this node has an ACL associated with it. It the configuration dictionary is deleted using the ConfigurationAdmin service, the data disappears but the ACL entry in the Dmt Admin remains. Even worse, if the data is recreated again, it will inherit the old ACL, which might not be the intended behaviour.

The current version of this specification does not solve this problem. Implementation specific workarounds exist, one possibility is to use some callback mechanism from the data backend to notify the Admin to clean up the ACLs.

5.9 The DmtAlertItem class

The *DmtAlertItem* class is a data structure carried in an alert (client initiated notification).It describes details of various alerts that can be sent by the client of the OMA DM protocol. The use cases include the client sending a session request to the server (alert 1201), the client notifying the server of completion of a software update operation (alert 1226) or sending back results in response to an asynchronous EXEC command. The latter technique is used in MEG to return data from a device log query and also for reporting KPI changes.

The data structure contains the following fields: source, type, format, data.

The data syntax and semantics varies widely between various alerts, so does the optionality of particular parameters of an alert item. If an item, such as source or type, is not defined, the corresponding getter method returns *null*. For example, for alert 1201 (client-initiated session) all elements will be *null*.

5.10 DmtException class

Most of the methods of this API throw *DmtException* whenever something goes wrong. The entity throwing the exception can indicate that the error is a fatal one which causes an ongoing atomic session to be rolled back automatically.

The exception class contains numeric error codes which describe the cause of the error. Some of the error codes correspond to the codes described by the OMA DM spec, some are introduced by MEG. Examples of both categories:

```
public static final int NODE_NOT_FOUND        = 404;
public static final int COMMAND_NOT_ALLOWED   = 405;
public static final int FEATURE_NOT_SUPPORTED = 406;
public static final int FORMAT_NOT_SUPPORTED  = 415;
```

and

```
public static final int METADATA_MISMATCH = 2;
public static final int CONCURRENT_ACCESS = 3;
public static final int INVALID_DATA      = 4;
public static final int ALERT_NOT_ROUTED  = 5;
```

5.11 The DmtPrincipalPermission class

Execution of the *getSession()* methods of the *DmtAdmin* featuring an explicit principal name is guarded by this permission so that only authorized entities can open a *DmtSession*. This permission must be granted only to management protocol adaptors who open *DmtSessions* on behalf of remote management servers. The target string of the permission is the name of the principal on whose behalf the session is created.

5.12 The DmtPermission class

The *DmtPermission* controls access to management objects in the Device Management Tree. It is intended to control only the local access to the DMT. *DmtPermission* target string identifies the management object URI and the action field lists the management commands that are permitted on the management object

5.13 Use cases

The following sequence diagrams describe the operation of the most typical usage of the Device Management API, with an emphasis on its interworking with the security subsystem.

5.13.1 Session initiated by a Remote Management Server

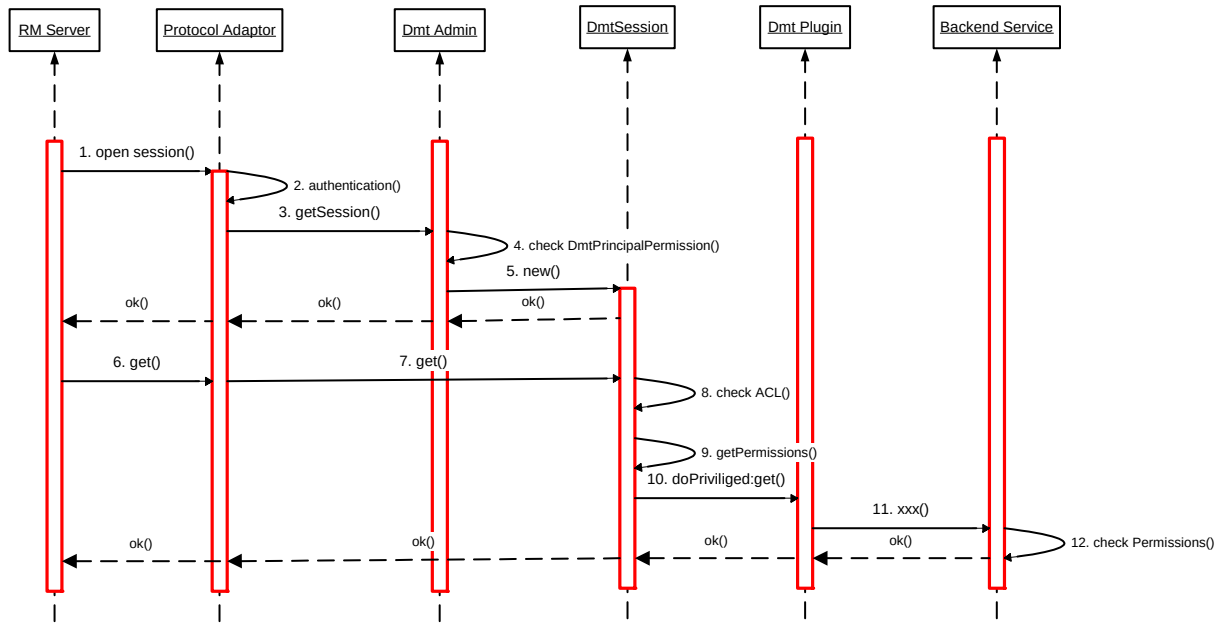

Figure 3 Remote session

1. The Remote Management Server initiates a management session on some management protocol.

2. The Remote Protocol Adaptor authenticates the server and determines it's identity. If authentication fails the server is notified in some protocol specific way.

3. The Remote Protocol Adaptor calls the DmtAdmin's getSession() method, supplying the server identification (principal) string as a parameter. (This is a slight simplification, actually first the Adaptor obtains the DmtAdmin service from the registry and calls it's getSession method. For brevity these kind of steps are not shown in these diagrams.)

4. The DmtAdmin checks whether the Remote Protocol Adaptor has the required DmtPrincipalPermission to issue the getSession() method. If not, a SecurityException is thrown.

5. The DmtAdmin creates a new DmtSession and returns it to the Protocol Adaptor. The Protocol Adaptor signals the successful opening of the session to the Remote Server in some protocol specific way.

6. The server issues a management command on a DMT node, in the figure a SyncML get is shown.

7. The Protocol Adaptor calls the get() method with the given node URI on its DmtSession instance.

8. The DmtSession checks the ACLs, whether the server has rights to access the given node. If this fails the server is notified in some protocol specific way.

9. The DmtSession collects the permissions associated with the given principal string. It's implementation specific how it is done: it can read the DMT subtree directly which stores the server permissions, or it can use a proprietary API offered by some 'DmtPrincipalAdmin' service.

10. Limiting it's permissions to the set obtained in the previous step the DmtSession forwards the call to the appropriate plugin in a doPriviliged call.

11. The plugin translates the DMT operation to the appropriate management call, e.g. a read request in a configuration database.

12. The backend service checks that everybody on the stack has the required permission to execute the management command. If yes, it executes and gives back the result. If not, a SecurityException is thrown. Note that here 'everybody on the stack' means:

    a. the plugin which needs to have the appropriate permissions (e.g. to modify the configuration database)

    b. the DmtSession implementation which is trusted system code

    c. the collection of permissions associated with the remote principal gathered in step 9.

13. The plugin translates the result back to a Device Management API result and forwards it the Protocol Adaptor. Eventually the result reaches the remote server in some protocol specific way.

5.13.2 Locally initiated session

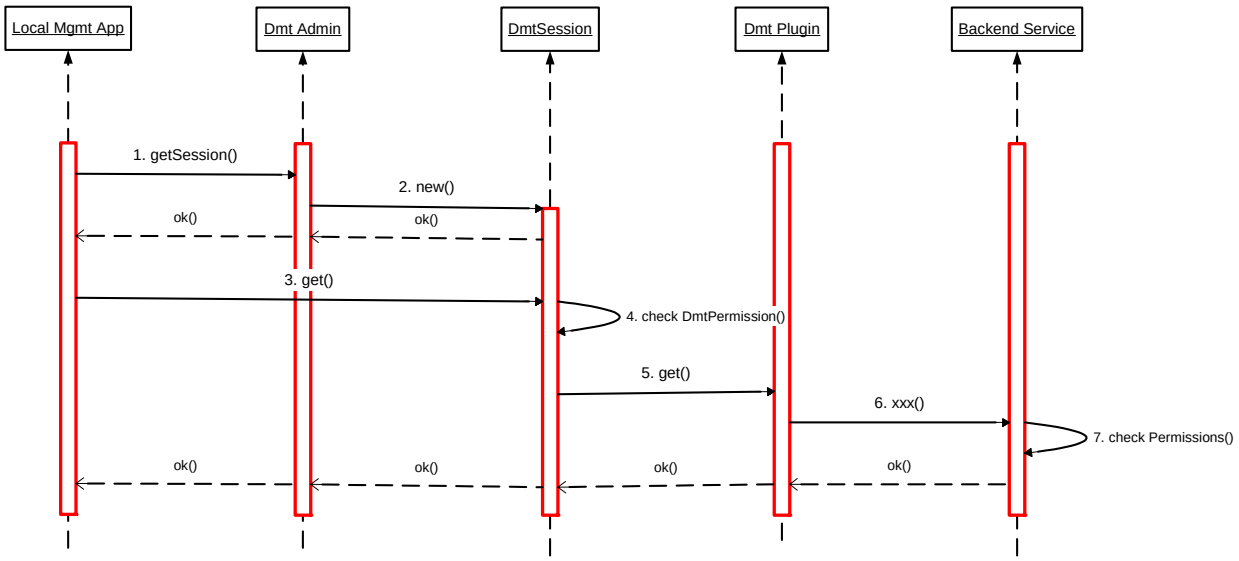

Figure 4 Local session

1. The local management application calls the getSession() method of the DmtAdmin to open a new session.

2. The DmtAdmin creates a new DmtSession and returnes it to the management application.

3. The application issues a DMT management call on the DmtSession.

4. The DmtSession checks whether the application has the required DmtPermission to issue this call. If not, a SecurityException is thrown.

All Page Within This Box

5. The DmtSession forwards the call to the appropriate plugin.

6. The plugin translates the DMT operation to the appropriate management call, e.g. a read request in a configuration database.

7. The backend service checks that everybody on the stack has the required permission to execute the management command. If yes, it executes and gives back the result. If not, a SecurityException is thrown.

8. The plugin translates the result back to a Device Management API result and forwards it to the DmtSession. Eventually it reaches the management application.

5.14 Dmt Events

When the DmtSession is closed, it generates up to 5 events containing the nodes that were added, replaced, deleted, renamed or copied.

Events for the device management tree are sent to one of these topics:

Topic	Notes
org/osgi/service/dmt/ADDED	New nodes were added.
org/osgi/service/dmt/DELETED	Existing nodes were removed.
org/osgi/service/dmt/REPLACED	Existing nodes were changed.
org/osgi/service/dmt/RENAMED	Existing nodes were renamed.
org/osgi/service/dmt/COPIED	Existing nodes were copied.

The order of the events is the same as in the table above, that is the ADDED events are sent first, the COPIED events are sent last.

DMT events have the following properties:

Property	Type	Notes
session.id	Integer	A unique identifier for the session that triggered the event.
nodes	String[]	The URI of each affected node. This is the `nodeUri` parameter of the Dmt API methods. The order of the URIs in the array corresponds to the chronological order of the operations. In case of a recursive delete, only the root URI is present in the array.
newnodes	String[]	Only the RENAMED and COPIED events have this property. In case of a rename newnodes[i] contains the new name of nodes[i], and in case of a copy newnodes[i] is the uri where nodes[i] was copied to.

5.14.1 Examples

In a given session, the node "a/b/c" and "a/b/c/d" is created, the value in the node "x/y/z" is updated, the node "p/q/r" (which has children "p/q/r/s1" and "p/q/r/s2") is deleted, the node “m/n1” is renamed to “m/n2” and the subtree under node “m/c1” is recursively copied to “m/c2”. When the session is closed the following events are published by the DMT in an arbitrary order:

All Page Within This Box

Event 1:
topic = "org/osgi/service/dmt/ADDED"
nodes = { "a/b/c", "a/b/c/d" }
session.id = 42

Event 2:
topic = "org/osgi/service/dmt/REPLACED"
nodes = { "x/y/z" }
session.id = 42

Event 3:
topic = "org/osgi/service/dmt/DELETED"
nodes = { "p/q/r" }
session.id = 42

Event 4:
topic = "org/osgi/service/dmt/RENAMED"
nodes = { "m/n1" }
newnodes = { "m/n2" }
session.id = 42

Event 5:
topic = "org/osgi/service/dmt/COPIED"
nodes = { "m/c1" }
newnodes = { "m/c2" }
session.id = 42

To receive notification when the node "a/b/c/d" is created, register a listener with the following properties:

topic = { "org/osgi/service/dmt/ADDED" }
filter = "(node=a/b/c/d)"

To receive notification when any node in the "a/b/c/*" sub-tree is replaced or removed, register a listener with the following properties:

topic = { "org/osgi/service/dmt/REPLACED", "org/osgi/service/dmt/REMOVED" }
filter = "(node=a/b/c/*)"

To receive notification when both the node "a/b/c/d1" and "a/b/c/d2", or just the node "a/b/c/d3" are touched in any way, register a listener with the following properties:

topic = { "org/osgi/service/dmt/*" }
filter = "(|(&(node=a/b/c/d1)(node=a/b/c/d2))(node=a/b/c/d3))"

6 Java API

All classes and interfaces are in the org.osgi.service.dmt package.

6.1 Dmt interface

public interface **Dmt**
extends DmtReadOnly

A collection of DMT manipulation methods. The application programmers use these methods when they are interacting with a `DmtSession` which inherits from this interface. Data plugins also implement this interface.

Method Summary

`void`	**commit**`()` Commits a series of DMT operations issued in the current atomic session since it was opened or since rollback was called the last time.
`void`	**copy**`(java.lang.String nodeUri, java.lang.String newNodeUri, boolean recursive)` Create a deep copy of a node.
`void`	**createInteriorNode**`(java.lang.String nodeUri)` Create an interior node
`void`	**createInteriorNode**`(java.lang.String nodeUri, java.lang.String type)` Create an interior node with a given type.
`void`	**createLeafNode**`(java.lang.String nodeUri)` Create a leaf node with default value.
`void`	**createLeafNode**`(java.lang.String nodeUri, DmtData value)` Create a leaf node with a given value.
`void`	**createLeafNode**`(java.lang.String nodeUri, DmtData value, java.lang.String mimeType)` Create a leaf node with a given value and MIME type.
`void`	**deleteNode**`(java.lang.String nodeUri)` Delete the given node.
`void`	**renameNode**`(java.lang.String nodeUri, java.lang.String newName)` Rename a node.
`void`	**rollback**`()` Rolls back a series of DMT operations issued in the current atomic session since it was opened.
`void`	**setDefaultNodeValue**`(java.lang.String nodeUri)` Set the value of a leaf node to it's default as defined by the node's meta data.
`void`	**setNodeTitle**`(java.lang.String nodeUri, java.lang.String title)` Set the title property of a node.
`void`	**setNodeType**`(java.lang.String nodeUri, java.lang.String type)` Set the type of a node.

All Page Within This Box

void	**setNodeValue**(java.lang.String nodeUri, DmtData data) Set the value of a leaf node.

Methods inherited from interface org.osgi.service.dmt.DmtReadOnly

close, getChildNodeNames, getMetaNode, getNodeSize, getNodeTimestamp, getNodeTitle, getNodeType, getNodeValue, getNodeVersion, isNodeUri

Method Detail

6.1.1 rollback

```
public void rollback()
            throws DmtException
```

Rolls back a series of DMT operations issued in the current atomic session since it was opened.

Throws:

DmtException - with the following possible error codes

- ROLLBACK_FAILED in case the rollback did not succeed
- FEATURE_NOT_SUPPORTED in case the session was not created using the LOCK_TYPE_ATOMIC lock type.

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.

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

---

6.1.2 commit

```
public void commit()
          throws DmtException
```

Commits a series of DMT operations issued in the current atomic session since it was opened or since rollback was called the last time.

This method can fail even if all operations were successful. This can happen due to some multi-node semantic constraints defined by a specific implementation. For example, node A can be required to always have children A.B, A.C and A.D. If this condition is broken when commit() is executed, the method will fail, and throw an exception.

Throws:

DmtException - with the following possible error codes

- TRANSACTION_ERROR if error occured because of lack of two phase commit in the underlying plugins. An example: plugin A has committed successfully but plugin B failed, so the whole session must fail, but A can not undo the commit
- FEATURE_NOT_SUPPORTED in case the session was not created using the LOCK_TYPE_ATOMIC lock type.
- COMMAND_FAILED if an underlying plugin failed to commit
- DATA_STORE_FAILURE
- METADATA_MISMATCH

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.

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

---

All Page Within This Box

6.1.3 setNodeTitle

```
public void setNodeTitle(java.lang.String nodeUri,
                         java.lang.String title)
                  throws DmtException
```

Set the title property of a node.

Parameters:

`nodeUri` - The URI of the node

`title` - The title text of the node

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `FEATURE_NOT_SUPPORTED`
- `COMMAND_FAILED` if the title string is too long or contains not allowed characters
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED`
- `METADATA_MISMATCH`
- `DATA_STORE_FAILURE`
- `TRANSACTION_ERROR`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

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

---

6.1.4 setNodeValue

```
public void setNodeValue(java.lang.String nodeUri,
                         DmtData data)
                  throws DmtException
```

Set the value of a leaf node.

Parameters:

`nodeUri` - The URI of the node

`data` - The data to be set. The format of the node is contained in the DmtData. Nodes of `null` format can be set by using `DmtData.NULL_VALUE` as second argument.

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `COMMAND_FAILED` if the data is null
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED` if the specified node is not a leaf node
- `METADATA_MISMATCH`
- `DATA_STORE_FAILURE`
- `FORMAT_NOT_SUPPORTED`
- `TRANSACTION_ERROR`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

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

---

6.1.5 setDefaultNodeValue

```
public void setDefaultNodeValue(java.lang.String nodeUri)
                         throws DmtException
```

Set the value of a leaf node to it's default as defined by the node's meta data. The method throws exception if there is no default defined.

Parameters:

`nodeUri` - The URI of the node

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `COMMAND_FAILED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED` if the specified node is not a leaf node
- `METADATA_MISMATCH`
- `DATA_STORE_FAILURE`
- `TRANSACTION_ERROR`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

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

---

6.1.6 setNodeType

```
public void setNodeType(java.lang.String nodeUri,
                        java.lang.String type)
                 throws DmtException
```

Set the type of a node. The type of leaf node is the MIME type of the data it contains. The type of interior node is an URL pointing to a DDF document.

Parameters:

`nodeUri` - The URI of the node

`type` - The type of the node

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `FEATURE_NOT_SUPPORTED`
- `COMMAND_FAILED` if the type string is null or invalid
- `COMMAND_NOT_ALLOWED`
- `METADATA_MISMATCH`
- `DATA_STORE_FAILURE`
- `FORMAT_NOT_SUPPORTED`

- TRANSACTION_ERROR

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.1.7 deleteNode

```
public void deleteNode(java.lang.String nodeUri)
                throws DmtException
```

Delete the given node. Deleting interior nodes is recursive, the whole subtree under the given node is deleted.

Parameters:

nodeUri - The URI of the node

Throws:

DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_NOT_ALLOWED if the node is permanent or non-deletable
- METADATA_MISMATCH
- DATA_STORE_FAILURE
- TRANSACTION_ERROR

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.1.8 createInteriorNode

```
public void createInteriorNode(java.lang.String nodeUri)
                        throws DmtException
```

Create an interior node

Parameters:

nodeUri - The URI of the node

Throws:

DmtException - with the following possible error codes

- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED
- NODE_ALREADY_EXISTS
- OTHER_ERROR if the URI is not within the current session's subtree
- METADATA_MISMATCH
- DATA_STORE_FAILURE
- TRANSACTION_ERROR

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.1.9 createInteriorNode

```
public void createInteriorNode(java.lang.String nodeUri,
                                java.lang.String type)
                         throws DmtException
```

Create an interior node with a given type. The type of interior node is an URL pointing to a DDF document.

Parameters:

`nodeUri` - The URI of the node

`type` - The type URL of the interior node

Throws:

DmtException - with the following possible error codes

- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `NODE_ALREADY_EXISTS`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_FAILED` if the type string is invalid
- `METADATA_MISMATCH`
- `DATA_STORE_FAILURE`
- `TRANSACTION_ERROR`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

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

---

6.1.10 createLeafNode

```
public void createLeafNode(java.lang.String nodeUri)
                    throws DmtException
```

Create a leaf node with default value. If a node does not have a default value defined by it's meta data, this method will throw a DmtException with error code `METADATA_MISMATCH`. The MIME type of the default node should also be specified by the meta data.

Parameters:

`nodeUri` - The URI of the node

Throws:

DmtException - with the following possible error codes

- `NODE_ALREADY_EXISTS`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED`
- `METADATA_MISMATCH`
- `DATA_STORE_FAILURE`
- `TRANSACTION_ERROR`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

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

---

All Page Within This Box

6.1.11 createLeafNode

```
public void createLeafNode(java.lang.String nodeUri,
                           DmtData value)
                    throws DmtException
```

Create a leaf node with a given value. The node's MIME type is not explicitely specified, it will be derived from the meta data associated with this node. The meta data defining the possible type (if any) should allow only one MIME type, otherwise this method will fail with `METADATA_MISMATCH`. Nodes of `null` format can be created by using `DmtData.NULL_VALUE` as second argument.

Parameters:

`nodeUri` - The URI of the node

`value` - The value to be given to the new node, can not be null

Throws:

DmtException - with the following possible error codes

- `NODE_ALREADY_EXISTS`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_FAILED` if the data is null
- `COMMAND_NOT_ALLOWED`
- `METADATA_MISMATCH`
- `DATA_STORE_FAILURE`
- `FORMAT_NOT_SUPPORTED`
- `TRANSACTION_ERROR`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

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

---

6.1.12 createLeafNode

```
public void createLeafNode(java.lang.String nodeUri,
                           DmtData value,
                           java.lang.String mimeType)
                    throws DmtException
```

Create a leaf node with a given value and MIME type.

Parameters:

`nodeUri` - The URI of the node

`value` - The value to be given to the new node, can not be null

`mimeType` - The MIME type to be given to the new node. It can be null.

Throws:

DmtException - with the following possible error codes

- `NODE_ALREADY_EXISTS`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED`
- `COMMAND_FAILED` if the data is null
- `METADATA_MISMATCH`
- `DATA_STORE_FAILURE`

- FORMAT_NOT_SUPPORTED
- TRANSACTION_ERROR

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.1.13 copy

```
public void copy(java.lang.String nodeUri,
                 java.lang.String newNodeUri,
                 boolean recursive)
          throws DmtException
```

Create a deep copy of a node. All properties and values will be copied. The command works for single nodes and recursively for whole subtrees.

Parameters:

nodeUri - The node or root of a subtree to be copied
newNodeUri - The URI of the new node or root of a subtree
recursive - false if only a single node is copied, true if the whole subtree is copied.

Throws:

DmtException - with the following possible error codes

- NODE_ALREADY_EXISTS if newNodeUri already exists
- NODE_NOT_FOUND if nodeUri does not exist
- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if either URI is not within the current session's subtree
- COMMAND_NOT_ALLOWED if any of the implied Get or Add commands are not allowed, or if nodeUri is an ancestor of newNodeUri
- METADATA_MISMATCH
- DATA_STORE_FAILURE
- FORMAT_NOT_SUPPORTED
- TRANSACTION_ERROR

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.1.14 renameNode

```
public void renameNode(java.lang.String nodeUri,
                       java.lang.String newName)
                throws DmtException
```

Rename a node. The value and the other properties of the node does not change.

Parameters:

nodeUri - The URI of the node to rename
newName - The new name property of the node. This is not the new URI of the node, the new URI is constructed from the old URI and the new name.

Throws:

DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG

- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED if the newName string is invalid
- COMMAND_NOT_ALLOWED
- METADATA_MISMATCH
- DATA_STORE_FAILURE
- TRANSACTION_ERROR

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.
`java.lang.SecurityException` - if the caller does not have the necessary permissions to execute the underlying management operation

6.2 DmtAcl class

public class **DmtAcl**
extends java.lang.Object
implements java.lang.Cloneable

DmtAcl is an immutable class representing structured access to DMT ACLs. Under OMA DM the ACLs are defined as strings with an internal syntax.
The methods of this class taking a principal as parameter accept remote server IDs (as passed to `DmtAdmin.getSession`), as well as "`*`" indicating any principal.

Field Summary

`static int`	**ADD** Principals holding this permission can issue ADD commands on the node having this ACL.
`static int`	**ALL_PERMISSION** Principals holding this permission can issue any command on the node having this ACL.
`static int`	**DELETE** Principals holding this permission can issue DELETE commands on the node having this ACL.
`static int`	**EXEC** Principals holding this permission can issue EXEC commands on the node having this ACL.
`static int`	**GET** Principals holding this permission can issue GET command on the node having this ACL.
`static int`	**REPLACE** Principals holding this permission can issue REPLACE commands on the node having this ACL.

Constructor Summary

DmtAcl() Create an instance of the ACL that represents an empty list of principals with no permissions.
DmtAcl(`java.lang.String acl`) Create an instance of the ACL from its canonic string representation.
DmtAcl(`java.lang.String[] principals, int[] permissions`) Creates an instance with specifying the list of principals and the permissions they hold.

All Page Within This Box

Method Summary

DmtAcl	**addPermission**(java.lang.String principal, int permissions) Create a new DmtAcl instance by adding a specific permission to a given principal.
java.lang.Object	**clone**() Creates a copy of this ACL object.
DmtAcl	**deletePermission**(java.lang.String principal, int permissions) Create a new DmtAcl instance by revoking a specific permission from a given principal.
int	**getPermissions**(java.lang.String principal) Get the permissions associated to a given principal.
java.lang.String[]	**getPrincipals**() Get the list of principals who have any kind of permissions on this node.
boolean	**isPermitted**(java.lang.String principal, int permissions) Check whether the given permissions are granted to a certain principal.
DmtAcl	**setPermission**(java.lang.String principal, int permissions) Create a new DmtAcl instance by setting the list of permissions a given principal has.
java.lang.String	**toString**() Give the canonic string representation of this ACL.

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

Field Detail

6.2.1 GET

`public static final int` **`GET`**

Principals holding this permission can issue GET command on the node having this ACL.

See Also:

Constant Field Values

6.2.2 ADD

`public static final int` **`ADD`**

Principals holding this permission can issue ADD commands on the node having this ACL.

See Also:

Constant Field Values

6.2.3 REPLACE

`public static final int` **`REPLACE`**

Principals holding this permission can issue REPLACE commands on the node having this ACL.

See Also:

Constant Field Values

All Page Within This Box

6.2.4 DELETE

`public static final int` **`DELETE`**

Principals holding this permission can issue DELETE commands on the node having this ACL.

See Also:

Constant Field Values

6.2.5 EXEC

`public static final int` **`EXEC`**

Principals holding this permission can issue EXEC commands on the node having this ACL.

See Also:

Constant Field Values

6.2.6 ALL_PERMISSION

`public static final int` **`ALL_PERMISSION`**

Principals holding this permission can issue any command on the node having this ACL. This permission is the logical OR of ADD, DELETE, EXEC, GET and REPLACE permissions.

See Also:

Constant Field Values

Constructor Detail

6.2.7 DmtAcl

`public` **`DmtAcl`**`()`

Create an instance of the ACL that represents an empty list of principals with no permissions.

6.2.8 DmtAcl

`public` **`DmtAcl`**`(java.lang.String acl)`

Create an instance of the ACL from its canonic string representation.

Parameters:

`acl` - The string representation of the ACL as defined in OMA DM. If `null` then it represents an empty list of principals with no permissions.

Throws:

`java.lang.IllegalArgumentException` - if acl is not a valid OMA DM ACL string

6.2.9 DmtAcl

`public` **`DmtAcl`**`(java.lang.String[] principals, int[] permissions)`

Creates an instance with specifying the list of principals and the permissions they hold. The two arrays run in parallel, that is `principals[i]` will hold `permissions[i]` in the ACL.

A principal name may not appear multiple times in the 'principals' argument. If the "*" principal appears in the array, the corresponding permissions will be granted to all principals (regardless of whether they appear in the array or not).

Parameters:

`principals` - The array of principals

`permissions` - The array of permissions

Throws:

`java.lang.IllegalArgumentException` - if the length of the two arrays are not the same or any array element is invalid

All Page Within This Box

Method Detail

6.2.10 clone

`public java.lang.Object clone()`

Creates a copy of this ACL object.

Returns:

a DmtAcl instance descibing the same permissions as this instance

---

6.2.11 addPermission

```
public DmtAcl addPermission(java.lang.String principal,
                            int permissions)
```

Create a new DmtAcl instance by adding a specific permission to a given principal. The already existing permissions of the principal are not affected.

Parameters:

`principal` - The entity to which permissions should be granted, or "*" to grant permissions to all principals.

`permissions` - The permissions to be given. The parameter can be a logical `or` of more permission constants defined in this class.

Returns:

a new DmtAcl instance

Throws:

`java.lang.IllegalArgumentException` - if `principal` is not a valid principal name or if `permissions` is not a valid combination of the permission constants defined in this class

---

6.2.12 deletePermission

```
public DmtAcl deletePermission(java.lang.String principal,
                               int permissions)
```

Create a new DmtAcl instance by revoking a specific permission from a given principal. Other permissions of the principal are not affected.

Note, that it is not valid to revoke a permission from a specific principal if that permission is granted globally to all principals.

Parameters:

`principal` - The entity from which permissions should be revoked, or "*" to revoke permissions from all principals.

`permissions` - The permissions to be revoked. The parameter can be a logical `or` of more permission constants defined in this class.

Returns:

a new DmtAcl instance

Throws:

`java.lang.IllegalArgumentException` - if `principal` is not a valid principal name, if `permissions` is not a valid combination of the permission constants defined in this class, or if a globally granted permission would have been revoked from a specific principal

---

6.2.13 getPermissions

`public int getPermissions(java.lang.String principal)`

Get the permissions associated to a given principal.

Parameters:

`principal` - The entity whose permissions to query, or "*" to query the permissions that are granted globally, to all principals

All Page Within This Box

Returns:
The permissions which the given principal has. The returned `int` is the logical `or` of the permission constants defined in this class.
Throws:
`java.lang.IllegalArgumentException` - if `principal` is not a valid principal name

6.2.14 isPermitted

```
public boolean isPermitted(java.lang.String principal,
                           int permissions)
```

Check whether the given permissions are granted to a certain principal.
Parameters:
`principal` - The entity to check, or "*" to check whether the given permissions are granted globally, to all principals
`permissions` - The permission to check
Returns:
`true` if the principal holds the given permission
Throws:
`java.lang.IllegalArgumentException` - if `principal` is not a valid principal name or if `permissions` is not a valid combination of the permission constants defined in this class

---

6.2.15 setPermission

```
public DmtAcl setPermission(java.lang.String principal,
                            int permissions)
```

Create a new DmtAcl instance by setting the list of permissions a given principal has. All permissions the principal had will be overwritten.
Note, that when changing the permissions of a specific principal, it is not allowed to specify a set of permissions stricter than the global set of permissions (that apply to all principals).
Parameters:
`principal` - The entity to which permissions should be granted, or "*" to grant permissions globally, to all principals.
`permissions` - The set of permissions to be given. The parameter can be a logical `or` of the permission constants defined in this class.
Returns:
a new DmtAcl instance
Throws:
`java.lang.IllegalArgumentException` - if `principal` is not a valid principal name, if `permissions` is not a valid combination of the permission constants defined in this class, or if a globally granted permission would have been revoked from a specific principal

---

6.2.16 getPrincipals

```
public java.lang.String[] getPrincipals()
```

Get the list of principals who have any kind of permissions on this node. The list only includes those principals that have been explicitly assigned permissions, globally set permissions naturally apply to all other principals as well.
Returns:
The array of principals having permissions on this node.

6.2.17 toString

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

Give the canonic string representation of this ACL. The operations are in the following order: {Add, Delete, Exec, Get, Replace}, principal names are sorted alphabetically.

Returns:

The string representation as defined in OMA DM.

6.3 DmtAdmin interface

public interface **DmtAdmin**

The DmtAdmin interface is used to create DmtSession objects. The implementation of DmtAdmin should register itself in the OSGi service registry as a service. DmtAdmin is the entry point for applications to use the Dmt API. The `getSession` methods are used to open a session on a specified subtree of the DMT. A typical way of usage:

```
serviceRef = context.getServiceReference(DmtAdmin.class.getName());
DmtAdmin factory = (DmtAdmin) context.getService(serviceRef);
DmtSession session = factory.getSession("./OSGi/cfg");
session.createInteriorNode("./OSGi/cfg/mycfg");
```

Method Summary

DmtSession	**getSession**(java.lang.String subtreeUri) Opens a DmtSession for local usage on a given subtree of the DMT with non transactional write lock.
DmtSession	**getSession**(java.lang.String subtreeUri, int lockMode) Opens a DmtSession for local usage on a specific DMT subtree with a given locking mode.
DmtSession	**getSession**(java.lang.String principal, java.lang.String subtreeUri, int lockMode) Opens a DmtSession on a specific DMT subtree using a specific locking mode on behalf of a remote principal.
void	**sendAlert**(java.lang.String principal, int code, DmtAlertItem[] items) Sends an alert to a named principal.

Method Detail

6.3.1 getSession

```
public DmtSession getSession(java.lang.String subtreeUri)
                      throws DmtException
```

Opens a DmtSession for local usage on a given subtree of the DMT with non transactional write lock. This call is equivalent to the following: `getSession(null, subtreeUri, DmtSession.LOCK_TYPE_EXCLUSIVE)`

Parameters:

`subtreeUri` - The subtree on which DMT manipulations can be performed within the returned session. If you want to use the whole DMT then use "." as subtree URI.

Returns:

a DmtSession object on which DMT manipulations can be performed

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`

All Page Within This Box

- TIMEOUT

6.3.2 getSession

```
public DmtSession getSession(java.lang.String subtreeUri,
                             int lockMode)
                      throws DmtException
```

Opens a DmtSession for local usage on a specific DMT subtree with a given locking mode. This call is equivalent to the following: `getSession(null, subtreeUri, lockMode)`

Parameters:

`subtreeUri` - The subtree on which DMT manipulations can be performed within the returned session. If you want to use the whole DMT then use "." as subtree URI.

`lockMode` - One of the locking modes specified in DmtSession

Returns:

a DmtSession object on which DMT manipulations can be performed

Throws:

DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- OTHER_ERROR if the lockMode is unknown
- TIMEOUT

6.3.3 getSession

```
public DmtSession getSession(java.lang.String principal,
                             java.lang.String subtreeUri,
                             int lockMode)
                      throws DmtException
```

Opens a DmtSession on a specific DMT subtree using a specific locking mode on behalf of a remote principal. If local management applications are using this method then they should provide `null` as the first parameter. Alternatively they can use other forms of this method without providing a principal string. This method is guarded by DmtPrincipalPermission.

Parameters:

`principal` - the identifier of the remote server on whose behalf the data manipulation is performed, or `null` for local sessions

`subtreeUri` - The subtree on which DMT manipulations can be performed within the returned session. If you want to use the whole DMT then use "." as subtree URI.

`lockMode` - One of the locking modes specified in DmtSession

Returns:

a DmtSession object on which DMT manipulations can be performed

Throws:

DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- OTHER_ERROR if the lockMode is unknown
- TIMEOUT

`java.lang.SecurityException` - if the caller does not have the required DmtPrincipalPermission

---

6.3.4 sendAlert

```
public void sendAlert(java.lang.String principal,
```

All Page Within This Box

```
                int code,
                DmtAlertItem[] items)
        throws DmtException
```

Sends an alert to a named principal. If OMA DM is used as a management protocol the principal name is server ID that corresponds to a DMT node value in `./SyncML/DMAcc/x/ServerId`. It is the DmtAdmin's responsibility to route the alert to the given principal.

Parameters:

`principal` - The principal name which is the recepient of this alert. In remotely initiated session it corresponds to a remote server ID, which can be obtained using the session's getPrincipal call.
It can be `null` when the client does not know the principal name. Even in this case the routing might be possible if the DmtAdmin is connected to only one protocol adapter which is connected to only one remote server.
`code` - Alert code. Can be 0 if not needed.
`items` - The data of the alert items carried in this alert. Can be `null` if not needed.

Throws:

DmtException - with the following possible error codes

- `ALERT_NOT_ROUTED` when the alert can not be routed to the server
- `FEATURE_NOT_SUPPORTED` in case of locally initiated sessions
- `REMOTE_ERROR` in case of communication problems between the device and the server

6.4 DmtAlertItem class

public class **DmtAlertItem**
extends java.lang.Object

Data structure carried in an alert (client initiated notification). The DmtAlertItem describes details of various alerts that can be sent by the client of the OMA DM protocol. The use cases include the client sending a session request to the server (alert 1201), the client notifying the server of completion of a software update operation (alert 1226) or sending back results in response to an asynchronous EXEC command.
The data syntax and semantics varies widely between various alerts, so does the optionality of particular parameters of an alert item. If an item, such as source or type, is not defined, the corresponding getter method returns `null`. For example, for alert 1201 (client-initiated session) all elements will be `null`.

Constructor Summary

DmtAlertItem(java.lang.String source, java.lang.String type, java.lang.String format, java.lang.String data) Create an instance of the alert item.	

Method Summary

java.lang.String	**getData**() Get the data associated with the alert item.
java.lang.String	**getFormat**() Get the format associated with the alert item.
java.lang.String	**getSource**() Get the node which is the source of the alert.
java.lang.String	**getType**() Get the type associated with the alert item.

All Page Within This Box

`java.lang.String`	**toString**()

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

Constructor Detail

6.4.1 DmtAlertItem

```
public DmtAlertItem(java.lang.String source,
                    java.lang.String type,
                    java.lang.String format,
                    java.lang.String data)
```

Create an instance of the alert item. The constructor takes all possible data entries as parameters. Any of these parameters can be `null`

Parameters:

`source` - The URI of the node which is the source of the alert item
`type` - The type of the alert item
`format` - The format of the alert item
`data` - The data of the alert item

Method Detail

6.4.2 getSource

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

Get the node which is the source of the alert. There might be no source associated with the alert item.

Returns:

The URI of the node which is the source of this alert. Can be `null`.

6.4.3 getType

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

Get the type associated with the alert item. There might be no format associated with the alert item.

Returns:

The type type associated with the alert item. Can be `null`.

6.4.4 getFormat

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

Get the format associated with the alert item. There might be no format associated with the alert item.

Returns:

The format associated with the alert item. Can be `null`.

6.4.5 getData

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

Get the data associated with the alert item. There might be no data associated with the alert item.

Returns:

The data associated with the alert item. Can be `null`.

6.4.6 toString

`public java.lang.String toString()`

6.5 DmtData class

public class **DmtData**
extends java.lang.Object

A data structure representing a leaf node. This structure represents only the value and the format property of the node, all other properties of the node (like MIME type) can be set and read using the `Dmt` and `DmtReadOnly` interfaces.

Different constructors are available to create nodes with different formats. Nodes of `null` format can be created using the static `DmtData.NULL_VALUE` constant instance of this class.

Field Summary

static int	**FORMAT_BINARY** The node holds an OMA DM `binary` value.
static int	**FORMAT_BOOLEAN** The node holds an OMA DM `bool` value.
static int	**FORMAT_INTEGER** The node holds an integer value.
static int	**FORMAT_NODE** Format specifier of an internal node.
static int	**FORMAT_NULL** The node holds an OMA DM `null` value.
static int	**FORMAT_STRING** The node holds an OMA DM `chr` value.
static int	**FORMAT_XML** The node holds an OMA DM `xml` value.
static DmtData	**NULL_VALUE** Constant instance representing a leaf node of `null` format.

Constructor Summary

DmtData(boolean bool) Create a DmtData instance of `bool` format and set its value.
DmtData(byte[] bytes) Create a DmtData instance of `bin` format and set its value.
DmtData(int integer) Create a DmtData instance of `int` format and set its value.
DmtData(java.lang.String str) Create a DmtData instance of `chr` format with the given string value.
DmtData(java.lang.String str, boolean xml) Create a DmtData instance of `xml` format and set its value.

All Page Within This Box

Method Summary

boolean	**equals**(java.lang.Object obj)
byte[]	**getBinary**() Gets the value of a node with binary format
boolean	**getBoolean**() Gets the value of a node with boolean format
int	**getFormat**() Get the node's format, expressed in terms of type constants defined in this class.
int	**getInt**() Gets the value of a node with integer format
java.lang.String	**getString**() Gets the value of a node with chr format
java.lang.String	**getXml**() Gets the value of a node with xml format
int	**hashCode**()
java.lang.String	**toString**() Gets the string representation of the DmtNode.

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

Field Detail

6.5.1 FORMAT_INTEGER

```
public static final int FORMAT_INTEGER
```

The node holds an integer value. Note that this does not correspond to the Java `int` type, OMA DM integers are unsigned.

See Also:

Constant Field Values

6.5.2 FORMAT_STRING

```
public static final int FORMAT_STRING
```

The node holds an OMA DM `chr` value.

See Also:

Constant Field Values

6.5.3 FORMAT_BOOLEAN

```
public static final int FORMAT_BOOLEAN
```

The node holds an OMA DM `bool` value.

See Also:

Constant Field Values

All Page Within This Box

6.5.4 FORMAT_BINARY

`public static final int` **`FORMAT_BINARY`**

The node holds an OMA DM `binary` value. The value of the node corresponds to the Java `byte[]` type.

See Also:

Constant Field Values

6.5.5 FORMAT_XML

`public static final int` **`FORMAT_XML`**

The node holds an OMA DM `xml` value.

See Also:

Constant Field Values

6.5.6 FORMAT_NULL

`public static final int` **`FORMAT_NULL`**

The node holds an OMA DM `null` value. This corresponds to the Java `null` type.

See Also:

Constant Field Values

6.5.7 FORMAT_NODE

`public static final int` **`FORMAT_NODE`**

Format specifier of an internal node. A DmtData instance can not have this value. This is used only as a return value of the `DmtMetaNode.getFormat()` method.

See Also:

Constant Field Values

6.5.8 NULL_VALUE

`public static DmtData` **`NULL_VALUE`**

Constant instance representing a leaf node of `null` format.

Constructor Detail

6.5.9 DmtData

`public` **`DmtData`**`(java.lang.String str)`

Create a DmtData instance of `chr` format with the given string value.

Parameters:

`str` - The string value to set

6.5.10 DmtData

```
public DmtData(java.lang.String str,
               boolean xml)
```

Create a DmtData instance of `xml` format and set its value.

Parameters:

`str` - The string or xml value to set

`xml` - If `true` then a node of `xml` is created otherwise this constructor behaves the same as `DmtData(String)`.

6.5.11 DmtData

`public` **`DmtData`**`(int integer)`

All Page Within This Box

Create a DmtData instance of `int` format and set its value.

Parameters:

`integer` - The integer value to set

6.5.12 DmtData

```
public DmtData(boolean bool)
```

Create a DmtData instance of `bool` format and set its value.

Parameters:

`bool` - The boolean value to set

6.5.13 DmtData

```
public DmtData(byte[] bytes)
```

Create a DmtData instance of `bin` format and set its value.

Parameters:

`bytes` - The byte array to set

Method Detail

6.5.14 getString

```
public java.lang.String getString()
                               throws DmtException
```

Gets the value of a node with chr format

Returns:

The string value

Throws:

DmtException - with the error code `OTHER_ERROR` if the format of the node is not chr

---

6.5.15 getXml

```
public java.lang.String getXml()
                        throws DmtException
```

Gets the value of a node with xml format

Returns:

The xml value

Throws:

DmtException - with the error code `OTHER_ERROR` if the format of the node is not xml

---

6.5.16 getInt

```
public int getInt()
           throws DmtException
```

Gets the value of a node with integer format

Returns:

The integer value

Throws:

DmtException - with the error code `OTHER_ERROR` if the format of the node is not integer

---

6.5.17 getBoolean

```
public boolean getBoolean()
                    throws DmtException
```

Gets the value of a node with boolean format

Returns:
The boolean value
Throws:
DmtException - with the error code OTHER_ERROR if the format of the node is not boolean

---

6.5.18 getBinary

```
public byte[] getBinary()
                  throws DmtException
```

Gets the value of a node with binary format
Returns:
The binary value
Throws:
DmtException - with the error code OTHER_ERROR if the format of the node is not binary

---

6.5.19 getFormat

```
public int getFormat()
```

Get the node's format, expressed in terms of type constants defined in this class. Note that the 'format' term is a legacy from OMA DM, it is more customary to think of this as 'type'.
Returns:
The format of the node.

6.5.20 toString

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

Gets the string representation of the DmtNode. This method works for all formats. [TODO specify for all formats. what does it mean if binary]
Returns:
The string value of the DmtData

6.5.21 equals

```
public boolean equals(java.lang.Object obj)
```

6.5.22 hashCode

```
public int hashCode()
```

6.6 DmtDataPlugin interface

public interface **DmtDataPlugin**
extends Dmt

An implementation of this interface takes the responsibility over a modifiable subtree of the DMT. If the subtree is non modifiable then the `DmtReadOnlyDataPlugin` interface should be used instead.
The plugin might support transactionality, in this case it has to implemenent commit and rollback functionality.
In an OSGi environment such implementations should be registered at the OSGi service registry specifying the list of root node URIs in a String array in the `dataRootURIs` registration parameter.

Method Summary

void	**open**(java.lang.String subtreeUri, int lockMode, DmtSession session)

	This method is called to signal the start of a transaction when the first reference is made within a DmtSession to a node which is handled by this plugin.
`boolean`	**supportsAtomic**() Tells whether the plugin can handle atomic transactions.

Methods inherited from interface org.osgi.service.dmt.Dmt
commit, copy, createInteriorNode, createInteriorNode, createLeafNode, createLeafNode, createLeafNode, deleteNode, renameNode, rollback, setDefaultNodeValue, setNodeTitle, setNodeType, setNodeValue

Methods inherited from interface org.osgi.service.dmt.DmtReadOnly
close, getChildNodeNames, getMetaNode, getNodeSize, getNodeTimestamp, getNodeTitle, getNodeType, getNodeValue, getNodeVersion, isNodeUri

Method Detail

6.6.1 open

```
public void open(java.lang.String subtreeUri,
                 int lockMode,
                 DmtSession session)
          throws DmtException
```

This method is called to signal the start of a transaction when the first reference is made within a DmtSession to a node which is handled by this plugin.

Parameters:

`subtreeUri` - The subtree which is locked in the current session

`lockMode` - One of the lock type constants specified in `DmtSession`

`session` - The session from which this plugin instance is accessed

Throws:

DmtException

6.6.2 supportsAtomic

```
public boolean supportsAtomic()
```

Tells whether the plugin can handle atomic transactions. If a session is created using `DmtSession.LOCK_TYPE_ATOMIC` locking and the plugin supports it then it is possible to roll back operations in the session.

Returns:

`true` if the plugin can handle atomic transactions

6.7 DmtException class

public class **DmtException**
extends java.lang.Exception

Checked exception received when a DMT operation fails. Beside the exception message, a `DmtException` always contains an error code (one of the constants specified in this class), and may optionally contain the URI of the related node, and information about the cause of the exception.

All Page Within This Box

The cause (if specified) can either be a single `Throwable` instance, or a list of such instances if several problems occured during the execution of a method. An example for the latter is the `close` method of `DmtSession` that tries to close multiple plugins, and has to report the exceptions of all failures.
Getter methods are provided to retrieve the values of the additional parameters, and the `printStackTrace` methods are extended to print the stack trace of all causing throwables as well.

See Also:

Serialized Form

Field Summary

`static int`	**ALERT_NOT_ROUTED** An alert can not be sent from the device to the Remote Management Server because of missing routing information.
`static int`	**COMMAND_FAILED** The recipient encountered an unexpected condition which prevented it from fulfilling the request.
`static int`	**COMMAND_NOT_ALLOWED** The requested command is not allowed on the target node.
`static int`	**CONCURRENT_ACCESS** An error occurred related to concurrent access of nodes.
`static int`	**DATA_STORE_FAILURE** An error related to the recipient data store occurred while processing the request.
`static int`	**FEATURE_NOT_SUPPORTED** The requested command failed because an optional feature in the request was not supported.
`static int`	**FORMAT_NOT_SUPPORTED** Unsupported media type or format.
`static int`	**INVALID_URI** The received URI string was `null`, contained not allowed characters or was not parseable to a valid URI.
`static int`	**METADATA_MISMATCH** Invalid data, operation failed bacause of meta data restrictions.
`static int`	**NODE_ALREADY_EXISTS** The requested `Add` or `Copy` command failed because the target already exists.
`static int`	**NODE_NOT_FOUND** The requested target node was not found.
`static int`	**OTHER_ERROR** An error occurred that does not fit naturally into any of the other error categories.
`static int`	**PERMISSION_DENIED** The requested command failed because the sender does not have adequate access control permissions (ACL) on the target.
`static int`	**REMOTE_ERROR** A device initiated remote operation failed.
`static int`	**ROLLBACK_FAILED** The rollback command was not completed successfully.
`static int`	**TIMEOUT**

All Page Within This Box

	Creation of a session timed out because of another ongoing session.
`static int`	**TRANSACTION_ERROR** This error is caused by one of the following situations: An updating method within an atomic session can not be executed because the underlying plugin does not support atomic transactions.
`static int`	**URI_TOO_LONG** The requested command failed because the target URI is too long for what the recipient is able or willing to process.

Constructor Summary

DmtException(`java.lang.String uri, int code, java.lang.String message`) Create an instance of the exception.
DmtException(`java.lang.String uri, int code, java.lang.String message, java.lang.Throwable cause`) Create an instance of the exception, specifying the cause exception.
DmtException(`java.lang.String uri, int code, java.lang.String message, java.util.Vector causes`) Create an instance of the exception, specifying the list of cause exceptions.
DmtException(`java.lang.String uri, int code, java.lang.String message, java.util.Vector causes, boolean fatal`) Create an instance of the exception, specifying the list of cause exceptions and whether the exception is a fatal one.

Method Summary

`java.lang.Throwable`	**getCause**() Get the cause of this exception.
`java.util.Vector`	**getCauses**() Get all causes of this exception.
`int`	**getCode**() Get the error code associated with this exception.
`java.lang.String`	**getMessage**() Get the message associated with this exception.
`java.lang.String`	**getURI**() Get the node on which the failed DMT operation was issued.
`boolean`	**isFatal**() Check whether this exception is fatal in the session, meaning that it triggers an automatic rollback of atomic sessions.
`void`	**printStackTrace**(`java.io.PrintStream s`) Prints the exception and its backtrace to the specified print stream.
`void`	**printStackTrace**(`java.io.PrintWriter s`) Prints the exception and its backtrace to the specified print writer.

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, getStackTrace, initCause, printStackTrace,`

All Page Within This Box

setStackTrace, toString

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

Field Detail

6.7.1 NODE_NOT_FOUND

`public static final int` **`NODE_NOT_FOUND`**

The requested target node was not found. No indication is given as to whether this is a temporary or permanent condition. This error code corresponds to the OMA DM response status code 404.

See Also:

Constant Field Values

6.7.2 COMMAND_NOT_ALLOWED

`public static final int` **`COMMAND_NOT_ALLOWED`**

The requested command is not allowed on the target node. Examples are executing a non-executable node or trying to read the value of an interior node. This error code corresponds to the OMA DM response status code 405.

See Also:

Constant Field Values

6.7.3 FEATURE_NOT_SUPPORTED

`public static final int` **`FEATURE_NOT_SUPPORTED`**

The requested command failed because an optional feature in the request was not supported. This error code corresponds to the OMA DM response status code 406.

See Also:

Constant Field Values

6.7.4 URI_TOO_LONG

`public static final int` **`URI_TOO_LONG`**

The requested command failed because the target URI is too long for what the recipient is able or willing to process. This error code corresponds to the OMA DM response status code 414.

See Also:

Constant Field Values

6.7.5 FORMAT_NOT_SUPPORTED

`public static final int` **`FORMAT_NOT_SUPPORTED`**

Unsupported media type or format. This error code corresponds to the OMA DM response status code 415.

See Also:

Constant Field Values

6.7.6 NODE_ALREADY_EXISTS

`public static final int` **`NODE_ALREADY_EXISTS`**

The requested `Add` or `Copy` command failed because the target already exists. This error code corresponds to the OMA DM response status code 418.

See Also:
Constant Field Values

6.7.7 PERMISSION_DENIED

`public static final int` **`PERMISSION_DENIED`**

The requested command failed because the sender does not have adequate access control permissions (ACL) on the target. This error code corresponds to the OMA DM response status code 425.

See Also:
Constant Field Values

6.7.8 COMMAND_FAILED

`public static final int` **`COMMAND_FAILED`**

The recipient encountered an unexpected condition which prevented it from fulfilling the request. This error code corresponds to the OMA DM response status code 500.

See Also:
Constant Field Values

6.7.9 DATA_STORE_FAILURE

`public static final int` **`DATA_STORE_FAILURE`**

An error related to the recipient data store occurred while processing the request. This error code corresponds to the OMA DM response status code 510.

See Also:
Constant Field Values

6.7.10 ROLLBACK_FAILED

`public static final int` **`ROLLBACK_FAILED`**

The rollback command was not completed successfully. It should be tried to recover the client back into original state. This error code corresponds to the OMA DM response status code 516.

See Also:
Constant Field Values

6.7.11 OTHER_ERROR

`public static final int` **`OTHER_ERROR`**

An error occurred that does not fit naturally into any of the other error categories. This error code does not correspond to any OMA DM response status code.

See Also:
Constant Field Values

6.7.12 REMOTE_ERROR

`public static final int` **`REMOTE_ERROR`**

A device initiated remote operation failed. This error code does not correspond to any OMA DM response status code.

See Also:
Constant Field Values

6.7.13 METADATA_MISMATCH

`public static final int` **`METADATA_MISMATCH`**

Invalid data, operation failed bacause of meta data restrictions. Examples can be violating referential integrity constraints or exceeding maximum node value limits, etc. This error code does not correspond to any OMA DM response status code.

See Also:

Constant Field Values

6.7.14 INVALID_URI

`public static final int` **`INVALID_URI`**

The received URI string was `null`, contained not allowed characters or was not parseable to a valid URI. This error code does not correspond to any OMA DM response status code.

See Also:

Constant Field Values

6.7.15 CONCURRENT_ACCESS

`public static final int` **`CONCURRENT_ACCESS`**

An error occurred related to concurrent access of nodes. For example a configuration node was deleted through the Config Admin while the node was manipulated via DMT. This error code does not correspond to any OMA DM response status code.

See Also:

Constant Field Values

6.7.16 ALERT_NOT_ROUTED

`public static final int` **`ALERT_NOT_ROUTED`**

An alert can not be sent from the device to the Remote Management Server because of missing routing information. This error code does not correspond to any OMA DM response status code.

See Also:

Constant Field Values

6.7.17 TRANSACTION_ERROR

`public static final int` **`TRANSACTION_ERROR`**

This error is caused by one of the following situations:

- An updating method within an atomic session can not be executed because the underlying plugin does not support atomic transactions.
- A commit operation at the end of an atomic session failed because of lack of two phase commit in the underlying plugins. An example: plugin A has committed successfully but plugin B failed, so the whole session must fail, but A can not undo the commit This error code does not correspond to any OMA DM response status code.

See Also:

Constant Field Values

6.7.18 TIMEOUT

`public static final int` **`TIMEOUT`**

Creation of a session timed out because of another ongoing session. This error code does not correspond to any OMA DM response status code. OMA has several status codes related to timeout, but these are meant to be used when a request times out, not when a session can not be established.

See Also:

Constant Field Values

Constructor Detail

6.7.19 DmtException

```
public DmtException(java.lang.String uri,
                    int code,
                    java.lang.String message)
```

Create an instance of the exception. No originating exception is specified.

Parameters:

`uri` - The node on which the failed DMT operation was issued
`code` - The error code of the failure
`message` - Message associated with the exception

6.7.20 DmtException

```
public DmtException(java.lang.String uri,
                    int code,
                    java.lang.String message,
                    java.lang.Throwable cause)
```

Create an instance of the exception, specifying the cause exception.

Parameters:

`uri` - The node on which the failed DMT operation was issued
`code` - The error code of the failure
`message` - Message associated with the exception
`cause` - The originating exception

6.7.21 DmtException

```
public DmtException(java.lang.String uri,
                    int code,
                    java.lang.String message,
                    java.util.Vector causes)
```

Create an instance of the exception, specifying the list of cause exceptions.

Parameters:

`uri` - The node on which the failed DMT operation was issued
`code` - The error code of the failure
`message` - Message associated with the exception
`causes` - The list of originating exceptions

6.7.22 DmtException

```
public DmtException(java.lang.String uri,
                    int code,
                    java.lang.String message,
                    java.util.Vector causes,
                    boolean fatal)
```

Create an instance of the exception, specifying the list of cause exceptions and whether the exception is a fatal one. This constructor is meant to be used by plugins wishing to indicate that a serious error occurred which should invalidate the ongoing atomic session.

Parameters:

`uri` - The node on which the failed DMT operation was issued
`code` - The error code of the failure
`message` - Message associated with the exception. Can be null.
`causes` - The list of originating exceptions
`fatal` - Whether the exception is fatal that is it triggers the automatic rollback of an ongoing atomic session.

Method Detail

6.7.23 getURI

`public java.lang.String` **`getURI`**`()`

Get the node on which the failed DMT operation was issued. Some operations like `DmtSession.close()` don't require an URI, in this case this method returns `null`.

Returns:

the URI of the node, or `null`

6.7.24 getCode

`public int` **`getCode`**`()`

Get the error code associated with this exception. Most of the error codes (returned by `getCode()`) within this exception correspond to OMA DM error codes.

Returns:

the error code

6.7.25 getMessage

`public java.lang.String` **`getMessage`**`()`

Get the message associated with this exception. The message also contains the associated URI and the exception code, if specified.

Returns:

the error message, or `null` if not specified

6.7.26 getCause

`public java.lang.Throwable` **`getCause`**`()`

Get the cause of this exception. Returns non-`null`, if this exception is caused by one or more other exceptions (like a `NullPointerException` in a Dmt Plugin).

6.7.27 getCauses

`public java.util.Vector` **`getCauses`**`()`

Get all causes of this exception. Returns the causing exceptions in a vector. If no cause was specified, an empty vector is returned.

6.7.28 isFatal

`public boolean` **`isFatal`**`()`

Check whether this exception is fatal in the session, meaning that it triggers an automatic rollback of atomic sessions.

6.7.29 printStackTrace

`public void` **`printStackTrace`**`(java.io.PrintStream s)`

Prints the exception and its backtrace to the specified print stream. Any causes that were specified for this exception are also printed, together with their backtraces.

Parameters:

`s` - `PrintStream` to use for output

6.7.30 printStackTrace

`public void printStackTrace(java.io.PrintWriter s)`

Prints the exception and its backtrace to the specified print writer. Any causes that were specified for this exception are also printed, together with their backtraces.

Parameters:

`s` - `PrintWriter` to use for output

6.8 DmtExecPlugin interface

public interface **DmtExecPlugin**

An implementation of this interface takes the responsibility of handling EXEC requests in a subtree of the DMT.

In an OSGi environment such implementations should be registered at the OSGi service registry specifying the list of root node URIs in a String array in the `execRootURIs` registration parameter.

Method Summary

void	**execute**(DmtSession session, java.lang.String nodeUri, java.lang.String data) Execute the given node with the given data.

Method Detail

6.8.1 execute

```
public void execute(DmtSession session,
                    java.lang.String nodeUri,
                    java.lang.String data)
             throws DmtException
```

Execute the given node with the given data. The `execute()` method of the DmtSession is forwarded to the appropriate DmtExecPlugin which handles the request. This operation corresponds to the EXEC command in OMA DM. The semantics of an EXEC operation and the data parameters it takes depend on the definition of the managed object on which the command is issued.

Parameters:

`session` - A reference to the session in which the operation was issued. Session information is needed in case an alert should be sent back from the plugin.

`nodeUri` - The node to be executed.

`data` - The data of the EXEC operation. The format of the data is not specified, it depends on the definition of the managed object (the node). Can be `null`.

Throws:

DmtException

6.9 DmtMetaNode interface

public interface **DmtMetaNode**

The DmtMetaNode contains meta data both standard for SyncML DM and defined by OSGi MEG (without breaking the compatibility) to provide for better DMT data quality in an environment where many software components manipulate this data.

The interface has two types of functions to describe type of nodes in the DMT. One is used to retrieve standard OMA DM metadata, such as access mode, cardinality, default etc. Another is used for meta data extensions defined by OSGi MEG, such as valid values and regular expressions.

Most of the methods of this class return `null` if a certain piece of meta information is not defined for the node or providing this information is not supported. Methods of this class do not throw exceptions.

Field Summary

`static int`	**CMD_ADD**
`static int`	**CMD_DELETE**
`static int`	**CMD_EXECUTE**
`static int`	**CMD_GET**
`static int`	**CMD_REPLACE**
`static int`	**DYNAMIC**
`static int`	**PERMANENT**

Method Summary

`boolean`	**can**`(int operation)` Check whether the given operation is valid for this node.
DmtData	**getDefault**`()` Get the default value of this node if any.
`java.lang.String`	**getDescription**`()` Get the explanation string associated with this node
`int`	**getFormat**`()` Get the node's format, expressed in terms of type constants defined in `DmtData`.
`int`	**getMax**`()` Get the maximum allowed value associated with an integer node.
`int`	**getMaxOccurrence**`()` Get the number of maximum occurrences of this type of nodes on the same level in the DMT.
`java.lang.String[]`	**getMimeTypes**`()` Get the list of MIME types this node can hold.
`int`	**getMin**`()` Get the minimum allowed value associated with an integer node.
`java.lang.String`	**getNamePattern**`()` Get the regular expression associated with the name of this node if any.
`java.lang.String`	**getPattern**`()` Get the regular expression associated with the value of this node if any.
`int`	**getScope**`()`

All Page Within This Box

	Return the scope of the node.
`java.lang.String[]`	**getValidNames**() Return an array of Strings if valid names are defined for the node, or `null` if no valid name list is defined or if this piece of meta info is not supported
DmtData[]	**getValidValues**() Return an array of DmtData objects if valid values are defined for the node, or `null` otherwise
`boolean`	**isLeaf**() Check whether the node is a leaf node or an internal one
`boolean`	**isZeroOccurrenceAllowed**() Check whether zero occurrence of this node is valid

Field Detail

6.9.1 CMD_ADD
`public static final int` **`CMD_ADD`**
See Also:
Constant Field Values

6.9.2 CMD_DELETE
`public static final int` **`CMD_DELETE`**
See Also:
Constant Field Values

6.9.3 CMD_EXECUTE
`public static final int` **`CMD_EXECUTE`**
See Also:
Constant Field Values

6.9.4 CMD_REPLACE
`public static final int` **`CMD_REPLACE`**
See Also:
Constant Field Values

6.9.5 CMD_GET
`public static final int` **`CMD_GET`**
See Also:
Constant Field Values

6.9.6 PERMANENT
`public static final int` **`PERMANENT`**
See Also:
Constant Field Values

All Page Within This Box

6.9.7 DYNAMIC

`public static final int DYNAMIC`

See Also:

Constant Field Values

Method Detail

6.9.8 can

`public boolean can(int operation)`

Check whether the given operation is valid for this node.

Parameters:

`operation` - One of the `DmtMetaNode.CMD_...` constants.

Returns:

`false` if the operation is not valid for this node or the operation code is not one of the allowed constants.

6.9.9 isLeaf

`public boolean isLeaf()`

Check whether the node is a leaf node or an internal one

Returns:

`true` if the node is a leaf node

6.9.10 getScope

`public int getScope()`

Return the scope of the node. Valid values are `DmtMetaNode.PERMANENT` and `DmtMetaNode.DYNAMIC`. Note that a permanent node is not the same as a node where the DELETE operation is not allowed. Permanent nodes never can be deleted, whereas a non-deletable node can disappear in a recursive DELETE operation issued on one of its parents.

Returns:

`DmtMetaNode.PERMANENT` or `DmtMetaNode.DYNAMIC`

6.9.11 getDescription

`public java.lang.String getDescription()`

Get the explanation string associated with this node

Returns:

Node description string

6.9.12 getMaxOccurrence

`public int getMaxOccurrence()`

Get the number of maximum occurrences of this type of nodes on the same level in the DMT. Returns `Integer.MAX_VALUE` if there is no upper limit. Note that if the occurrence is greater than 1 then this node can not have siblings with different metadata. That is, if different type of nodes coexist on the same level, their occurrence can not be greater than 1.

Returns:

The maximum allowed occurrence of this node type

6.9.13 isZeroOccurrenceAllowed

`public boolean isZeroOccurrenceAllowed()`

Check whether zero occurrence of this node is valid

Returns:

`true` if zero occurrence of this node is valid

6.9.14 getDefault

`public DmtData getDefault()`

Get the default value of this node if any.

Returns:

The default value or `null` if not defined.

6.9.15 getMax

`public int getMax()`

Get the maximum allowed value associated with an integer node.

Returns:

The allowed maximum. If there is no upper limit defined or the node's format is not integer then `Integer.MAX_VALUE` is returned.

6.9.16 getMin

`public int getMin()`

Get the minimum allowed value associated with an integer node.

Returns:

The allowed minimum. If there is no lower limit defined or the node's format is not integer then `Integer.MIN_VALUE` is returned.

6.9.17 getValidValues

`public DmtData[] getValidValues()`

Return an array of DmtData objects if valid values are defined for the node, or `null` otherwise

Returns:

the valid values for this node, or `null` if not defined

6.9.18 getValidNames

`public java.lang.String[] getValidNames()`

Return an array of Strings if valid names are defined for the node, or `null` if no valid name list is defined or if this piece of meta info is not supported

Returns:

the valid values for this node name, or `null` if not defined

6.9.19 getFormat

`public int getFormat()`

Get the node's format, expressed in terms of type constants defined in `DmtData`. If there are multiple formats allowed for the node then the format constants ar OR-ed. Note that the 'format' term is a legacy from OMA DM, it is more customary to think of this as 'type'.

Returns:

The format of the node.

6.9.20 getPattern

`public java.lang.String getPattern()`

Get the regular expression associated with the value of this node if any. This method makes sense only in the case of `chr` nodes.

All Page Within This Box

Returns:
The regular expression associated with this node or `null` if not defined, or if the node is not of type `chr`.

6.9.21 getNamePattern

`public java.lang.String` **`getNamePattern`**`()`

Get the regular expression associated with the name of this node if any.

Returns:
The regular expression associated with the name of this node or `null` if not defined.

6.9.22 getMimeTypes

`public java.lang.String[]` **`getMimeTypes`**`()`

Get the list of MIME types this node can hold.

Returns:
The list of allowed MIME types for this node or `null` if not defined. If there is a default value defined for this node then the associated MIME type (if any) must be the first element of the list.

6.10 DmtPermisssion class

public class **DmtPermission**
extends java.security.Permission

DmtPermission controls access to management objects in the Device Management Tree (DMT). It is intended to control local access to the DMT. DMTPermission target string identifies the management object URI and the action field lists the OMA DM commands that are permitted on the management object. Example:

```
DMTPermission("./OSGi/bundles", "Add,Replace,Get");
```

This means that owner of this permission can execute Add, Replace and Get commands on the ./OSGi/bundles management object. It is possible to use wildcards in both the target and the actions field. Wildcard in the target field means that the owner of the permission can access children nodes of the target node. Example

```
DMTPermission("./OSGi/bundles/*", "Get");
```

This means that owner of this permission has Get access on every child node of ./OSGi/bundles. The asterix does not necessarily have to follow a '/' character. For example the `"./OSGi/a*"/ target matches the ./OSGi/applications subtree.`

If wildcard is present in the actions field, all legal OMA DM commands are allowed on the designated nodes(s) by the owner of the permission.

See Also:
Serialized Form

Constructor Summary

DmtPermission`(java.lang.String dmturi, java.lang.String actions)`	Creates a new DmtPermission object for the specified DMT URI with the specified actions.

Method Summary

`boolean`	**equals**`(java.lang.Object obj)` Checks two DMTPermission objects for equality.
`java.lang.String`	**getActions**`()` Returns the String representation of the action list.

All Page Within This Box

int	**hashCode**() Returns hash code for this permission object.
boolean	**implies**(java.security.Permission p) Checks if this DMTPermission object "implies" the specified permission.
java.security.PermissionCollection	**newPermissionCollection**() Returns a new PermissionCollection object for storing DMTPermission objects.

Methods inherited from class java.security.Permission
checkGuard, getName, toString

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

Constructor Detail

6.10.1 DmtPermission

```
public DmtPermission(java.lang.String dmturi,
                     java.lang.String actions)
```

Creates a new DmtPermission object for the specified DMT URI with the specified actions.

Parameters:

`dmturi` - URI of the management object (or subtree).

`actions` - OMA DM actions allowed.

Method Detail

6.10.2 equals

```
public boolean equals(java.lang.Object obj)
```

Checks two DMTPermission objects for equality. Two DMTPermissions are equal if they have the same target and action strings.

Returns:

true if the two objects are equal.

6.10.3 getActions

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

Returns the String representation of the action list.

Returns:

Action list for this permission object.

6.10.4 hashCode

```
public int hashCode()
```

Returns hash code for this permission object. If two DMTPermission objects are equal according to the equals method, then calling the hashCode method on each of the two DMTPermission objects must produce the same integer result.

Returns:

hash code for this permission object.

All Page Within This Box

6.10.5 implies

`public boolean implies(java.security.Permission p)`

Checks if this DMTPermission object "implies" the specified permission.

Parameters:

p - Permission to check.

Returns:

true if this DMTPermission object implies the specified permission.

6.10.6 newPermissionCollection

`public java.security.PermissionCollection newPermissionCollection()`

Returns a new PermissionCollection object for storing DMTPermission objects.

Returns:

the new PermissionCollection.

6.11 DmtPrincipalPermission class

public class **DmtPrincipalPermission**
extends java.security.BasicPermission

Indicates the callers authority to create DMT sessions in the name of a remote management server. Only protocol adapters communicating with management servers should be granted this permission. `DmtPrincipalPermission` has a target string which controls the name of the principal on whose behalf the protocol adapter can act. A wildcard is allowed in the target as defined by `BasicPermission`, for example a "*" means the adapter can create a session in the name of any principal.

See Also:

Serialized Form

Constructor Summary

DmtPrincipalPermission(`java.lang.String target`) Creates a new `DmtPrincipalPermission` object with its name set to the target string
DmtPrincipalPermission(`java.lang.String target, java.lang.String actions`) Creates a new `DmtPrincipalPermission` object using the 'canonic' two argument constructor.

Methods inherited from class java.security.BasicPermission
`equals, getActions, hashCode, implies, newPermissionCollection`

Methods inherited from class java.security.Permission
`checkGuard, getName, toString`

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

Constructor Detail

All Page Within This Box

6.11.1 DmtPrincipalPermission

`public DmtPrincipalPermission(java.lang.String target)`

Creates a new `DmtPrincipalPermission` object with its name set to the target string

Parameters:

`target` - Name of the principal

6.11.2 DmtPrincipalPermission

```
public DmtPrincipalPermission(java.lang.String target,
                              java.lang.String actions)
```

Creates a new `DmtPrincipalPermission` object using the 'canonic' two argument constructor.

Parameters:

`target` - Name of the principal
`actions` - ignored

6.12 DmtReadOnly interface

public interface **DmtReadOnly**

This interface collects basic DMT operations. The application programmers use these methods when they are interacting with a `DmtSession` which inherits from this interface. The `DmtDataPlugin` and the DmtReadOnlyDataPlugin interfaces also extend this interface.

If the admin or a data plugin does not support an optional DMT property (like `timestamp`) then the corresponding getter method throws `DmtException` with the error code FEATURE_NOT_SUPPORTED. In case the Dmt Admin receives a `null` from a plugin (or other value it can not interpret as a proper result for a property getter method) it must also throw the appropriate `DmtException`.

Method Summary

`void`	**close**`()` Closes a session and makes the changes made to the DMT persistent.
`java.lang.String[]`	**getChildNodeNames**`(java.lang.String nodeUri)` Get the list of children names of a node.
DmtMetaNode	**getMetaNode**`(java.lang.String nodeUri)` Get the meta data which describes a given node.
`int`	**getNodeSize**`(java.lang.String nodeUri)` Get the size of the data in the node in bytes.
`java.util.Date`	**getNodeTimestamp**`(java.lang.String nodeUri)` Get the timestamp when the node was last modified
`java.lang.String`	**getNodeTitle**`(java.lang.String nodeUri)` Get the title of a node
`java.lang.String`	**getNodeType**`(java.lang.String nodeUri)` Get the type of a node.
DmtData	**getNodeValue**`(java.lang.String nodeUri)` Get the data contained in a leaf node.
`int`	**getNodeVersion**`(java.lang.String nodeUri)` Get the version of a node.
`boolean`	**isNodeUri**`(java.lang.String nodeUri)`

All Page Within This Box

	Check whether the specified URI corresponds to a valid node in the DMT.

Method Detail

6.12.1 close

```
public void close()
            throws DmtException
```

Closes a session and makes the changes made to the DMT persistent. Persisting the changes works differently for exclusive and atomic lock. For the former all changes that were accepted are persisted. For the latter once an error is encountered, all successful changes are rolled back.

This method can fail even if all operations were successful. This can happen due to some multi-node semantic constraints defined by a specific implementation. For example, node A can be required to always have children A.B, A.C and A.D. If this condition is broken when `close()` is executed, the method will fail, and throw an exception.

After this method is called the session's state is `DmtSession.STATE_CLOSED`.

Throws:

DmtException - with the following possible error codes

- `COMMAND_FAILED` if an underlying plugin failed to close
- `DATA_STORE_FAILURE`
- `METADATA_MISMATCH`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

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

---

6.12.2 isNodeUri

```
public boolean isNodeUri(java.lang.String nodeUri)
```

Check whether the specified URI corresponds to a valid node in the DMT.

Parameters:

`nodeUri` - the URI to check

Returns:

true if the given node exists in the DMT

Throws:

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

6.12.3 getNodeValue

```
public DmtData getNodeValue(java.lang.String nodeUri)
                     throws DmtException
```

Get the data contained in a leaf node.

Parameters:

`nodeUri` - The URI of the node to retrieve

Returns:

The data of the leaf node

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`

All Page Within This Box

- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED
- COMMAND_NOT_ALLOWED if the specified node is not a leaf node
- DATA_STORE_FAILURE

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.12.4 getNodeTitle

```
public java.lang.String getNodeTitle(java.lang.String nodeUri)
                               throws DmtException
```

Get the title of a node
Parameters:
nodeUri - The URI of the node
Returns:
The title of the node
Throws:
DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED
- COMMAND_NOT_ALLOWED
- FEATURE_NOT_SUPPORTED
- DATA_STORE_FAILURE

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.12.5 getNodeType

```
public java.lang.String getNodeType(java.lang.String nodeUri)
                              throws DmtException
```

Get the type of a node. The type of leaf node is the MIME type of the data it contains. The type of interior node is an URL pointing to a DDF document.
Parameters:
nodeUri - The URI of the node
Returns:
The type of the node
Throws:
DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED

All Page Within This Box

- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED
- COMMAND_NOT_ALLOWED
- FEATURE_NOT_SUPPORTED
- DATA_STORE_FAILURE

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.12.6 getNodeVersion

```
public int getNodeVersion(java.lang.String nodeUri)
                  throws DmtException
```

Get the version of a node. The version can not be set, it is calculated automatically by the device. It is incremented each time the value of a leaf node is changed. When a node is created the initial value is 0. The version property is undefined for interior nodes.

Parameters:
nodeUri - The URI of the node
Returns:
The version of the node
Throws:
DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED
- COMMAND_NOT_ALLOWED
- FEATURE_NOT_SUPPORTED
- DATA_STORE_FAILURE

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.12.7 getNodeTimestamp

```
public java.util.Date getNodeTimestamp(java.lang.String nodeUri)
                              throws DmtException
```

Get the timestamp when the node was last modified

Parameters:
nodeUri - The URI of the node
Returns:
The timestamp of the last modification
Throws:
DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED

All Page Within This Box

- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED
- COMMAND_NOT_ALLOWED
- FEATURE_NOT_SUPPORTED
- DATA_STORE_FAILURE

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.12.8 getNodeSize

```
public int getNodeSize(java.lang.String nodeUri)
                throws DmtException
```

Get the size of the data in the node in bytes. Throws DmtException with the error code COMMAND_NOT_ALLOWED if issued on an interior node.

Parameters:
nodeUri - The URI of the node
Returns:
The size of the node
Throws:
DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED
- COMMAND_NOT_ALLOWED
- FEATURE_NOT_SUPPORTED
- DATA_STORE_FAILURE

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.12.9 getChildNodeNames

```
public java.lang.String[] getChildNodeNames(java.lang.String nodeUri)
                                     throws DmtException
```

Get the list of children names of a node. The returned array contains the names - not the URIs - of the immediate children nodes of the given node. The returned array must not contain null entries. If a plugin returns null as an array element, then the admin must remove it from the array.

Parameters:
nodeUri - The URI of the node
Returns:
The list of children node names as a string array or null if the node has no children.
Throws:
DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG

- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED
- COMMAND_NOT_ALLOWED
- DATA_STORE_FAILURE

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.12.10 getMetaNode

```
public DmtMetaNode getMetaNode(java.lang.String nodeUri)
                        throws DmtException
```

Get the meta data which describes a given node. Meta data can be only inspected, it can not be changed. This method is inherited in the DmtSession and also in the Data Plugin interfaces, but the semantics of the method is slightly different in these two cases. Meta data can be supported at the engine level (i.e. in Dmt Admin), and its support by plugins is an optional (and advanced) feature. It can be used, for example, when a data plugin is implemented on top of a data store or another API that has their own metadata, such as a relational database, in order to avoid metadata duplication and inconsistency. The meta data specific to the plugin returned by the plugin's getMetaNode() method is complemented by the engine level meta data. The DmtMetaNode the client receives on the DmtSession.getMetaNode() call is the combination of the meta data returned by the data plugin plus the meta data returned by the Dmt Admin. If there are differences in the meta data elements known by the plugin and the Dmt Admin then the plugin specific elements take precedence.

Parameters:
nodeUri - the URI of the node
Returns:
a DmtMetaNode which describes meta data information
Throws:
DmtException - with the following possible error codes

- NODE_NOT_FOUND
- URI_TOO_LONG
- INVALID_URI
- PERMISSION_DENIED
- OTHER_ERROR if the URI is not within the current session's subtree
- COMMAND_FAILED
- COMMAND_NOT_ALLOWED
- FEATURE_NOT_SUPPORTED
- DATA_STORE_FAILURE

java.lang.IllegalStateException - if the session is invalidated because of timeout, or if the session is already closed.
java.lang.SecurityException - if the caller does not have the necessary permissions to execute the underlying management operation

---

6.13 DmtReadOnlyDataPlugin interface

public interface **DmtReadOnlyDataPlugin**

All Page Within This Box

extends DmtReadOnly

An implementation of this interface takes the responsibility over a non-modifiable subtree of the DMT. In an OSGi environment such implementations should be registered at the OSGi service registry specifying the list of root node URIs in a String array in the `dataRootURIs` registration parameter.

Method Summary

void	**open**(java.lang.String subtreeUri, DmtSession session) This method is called to signal the start of a transaction when the first reference is made within a DmtSession to a node which is handled by this plugin.

Methods inherited from interface org.osgi.service.dmt.DmtReadOnly

close, getChildNodeNames, getMetaNode, getNodeSize, getNodeTimestamp, getNodeTitle, getNodeType, getNodeValue, getNodeVersion, isNodeUri

Method Detail

6.13.1 open

```
public void open(java.lang.String subtreeUri,
                 DmtSession session)
          throws DmtException
```

This method is called to signal the start of a transaction when the first reference is made within a DmtSession to a node which is handled by this plugin. Although a read only plugin need not be concerned about transactionality, knowing the session from which it is accessed can be useful for example in the case of sending alerts.

Parameters:

`subtreeUri` - The subtree which is locked in the current session

`session` - The session from which this plugin instance is accessed

Throws:

DmtException

6.14 DmtSession interface

public interface **DmtSession**
extends Dmt

DmtSession provides concurrent access to the DMT. All DMT manipulation commands for management applications are available on the DmtSession interface. The session is associated with a root node which limits the subtree in which the operations can be executed within this session. Most of the operations take a node URI as parameter, it can be either an absolute URI (starting with "./") or a URI relative to the root node of the session. If the URI specified does not correspond to a legitimate node in the tree an exception is thrown. The only exception is the isNodeUri() method which returns false in case of an invalid URI.

Each method of the DmtSession can throw InvalidStateException in case the session is invalidated because of timeout.

Field Summary

static int	**LOCK_TYPE_ATOMIC**

All Page Within This Box

	LOCK_TYPE_ATOMIC is an exclusive lock with transactional functionality.
static int	**LOCK_TYPE_EXCLUSIVE** LOCK_TYPE_EXCLUSIVE lock guarantees full access to the tree, but can not be shared with any other locks.
static int	**LOCK_TYPE_SHARED** Sessions created with LOCK_TYPE_SHARED lock allows read-only access to the tree, but can be shared between multiple readers.
static int	**STATE_CLOSED** The session is closed, DMT manipulation operations are not available, they throw InvalidStateException if tried.
static int	**STATE_INVALID** The session is invalid because of it was timed out or a fatal exception happened in an atomic session.
static int	**STATE_OPEN** The session is open, all session operations are available

Method Summary

void	**execute**(java.lang.String nodeUri, java.lang.String data) Executes a node.
DmtAcl	**getEffectiveNodeAcl**(java.lang.String nodeUri) Gives the Access Control List in effect for a given node.
int	**getLockType**() Gives the type of lock the session currently has.
DmtAcl	**getNodeAcl**(java.lang.String nodeUri) Gives the Access Control List associated with a given node.
java.lang.String	**getPrincipal**() Gives the name of the principal on whose behalf the session was created.
java.lang.String	**getRootUri**() Get the root URI associated with this session.
int	**getSessionId**() The unique identifier of the session.
int	**getState**() Get the current state of this session
boolean	**isLeafNode**(java.lang.String nodeUri) Tells whether a node is a leaf or an interior node of the DMT.
void	**setNodeAcl**(java.lang.String nodeUri, DmtAcl acl) Set the Access Control List associated with a given node.

Methods inherited from interface org.osgi.service.dmt.Dmt

commit, copy, createInteriorNode, createInteriorNode, createLeafNode, createLeafNode, createLeafNode, deleteNode, renameNode, rollback, setDefaultNodeValue, setNodeTitle, setNodeType, setNodeValue

All Page Within This Box

Methods inherited from interface org.osgi.service.dmt.DmtReadOnly
close, getChildNodeNames, getMetaNode, getNodeSize, getNodeTimestamp, getNodeTitle, getNodeType, getNodeValue, getNodeVersion, isNodeUri

Field Detail

6.14.1 LOCK_TYPE_SHARED

`public static final int` **`LOCK_TYPE_SHARED`**

Sessions created with `LOCK_TYPE_SHARED` lock allows read-only access to the tree, but can be shared between multiple readers.

See Also:
Constant Field Values

6.14.2 LOCK_TYPE_EXCLUSIVE

`public static final int` **`LOCK_TYPE_EXCLUSIVE`**

`LOCK_TYPE_EXCLUSIVE` lock guarantees full access to the tree, but can not be shared with any other locks.

See Also:
Constant Field Values

6.14.3 LOCK_TYPE_ATOMIC

`public static final int` **`LOCK_TYPE_ATOMIC`**

`LOCK_TYPE_ATOMIC` is an exclusive lock with transactional functionality. Commands of an atomic session will either fail or succeed together, if a single command fails then the whole session will be rolled back.

See Also:
Constant Field Values

6.14.4 STATE_OPEN

`public static final int` **`STATE_OPEN`**

The session is open, all session operations are available

See Also:
Constant Field Values

6.14.5 STATE_CLOSED

`public static final int` **`STATE_CLOSED`**

The session is closed, DMT manipulation operations are not available, they throw `InvalidStateException` if tried.

See Also:
Constant Field Values

6.14.6 STATE_INVALID

`public static final int` **`STATE_INVALID`**

The session is invalid because of it was timed out or a fatal exception happened in an atomic session. DMT manipulation operations are not available, they throw `InvalidStateException` if tried.

See Also:
Constant Field Values

All Page Within This Box

Method Detail

6.14.7 getState

`public int` **`getState`**`()`

Get the current state of this session

Returns:

the state of the session, one of the `STATE_...` constants

6.14.8 getLockType

`public int` **`getLockType`**`()`

Gives the type of lock the session currently has.

Returns:

One of the `LOCK_TYPE_...` constants.

6.14.9 getPrincipal

`public java.lang.String` **`getPrincipal`**`()`

Gives the name of the principal on whose behalf the session was created. Local sessions do not have an associated principal, in this case `null` is returned.

Returns:

the identifier of the remote server that initiated the session, or `null` for local sessions

---

6.14.10 getSessionId

`public int` **`getSessionId`**`()`

The unique identifier of the session. The ID is generated automatically, and it is guaranteed to be unique on a machine.

Returns:

the session identification number

6.14.11 execute

```
public void execute(java.lang.String nodeUri,
                    java.lang.String data)
             throws DmtException
```

Executes a node. This corresponds to the EXEC operation in OMA DM. The semantics of an EXEC operation depend on the definition of the managed object on which it is issued.

Parameters:

`nodeUri` - the node on which the execute operation is issued

`data` - the parameters to the execute operation. The format of the data string is described by the managed object definition. Can be `null`.

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED` if the node is non-executable
- `COMMAND_FAILED` if no DmtExecPlugin is associated with the node
- `DATA_STORE_FAILURE`

All Page Within This Box

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

6.14.12 isLeafNode

```
public boolean isLeafNode(java.lang.String nodeUri)
                  throws DmtException
```

Tells whether a node is a leaf or an interior node of the DMT.

Parameters:

`nodeUri` - the URI of the node

Returns:

true if the given node is a leaf node

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

6.14.13 getNodeAcl

```
public DmtAcl getNodeAcl(java.lang.String nodeUri)
                 throws DmtException
```

Gives the Access Control List associated with a given node. The returned DmtAcl does not take inheritance into account, it gives the ACL specifically given to the node.

Parameters:

`nodeUri` - the URI of the node

Returns:

the Access Control List belonging to the node or `null` if none defined

Throws:

DmtException - with the following possible error codes

- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED`
- `DATA_STORE_FAILURE`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

6.14.14 getEffectiveNodeAcl

```
public DmtAcl getEffectiveNodeAcl(java.lang.String nodeUri)
                          throws DmtException
```

Gives the Access Control List in effect for a given node. The returned DmtAcl takes inheritance into accout, that is if there is no ACL defined for the node, it will be derived from the closest ancestor having an ACL defined.

All Page Within This Box

Parameters:
`nodeUri` - the URI of the node
Returns:
the Access Control List belonging to the node
Throws:
DmtException - with the following possible error codes
- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED`
- `DATA_STORE_FAILURE`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

6.14.15 setNodeAcl

```
public void setNodeAcl(java.lang.String nodeUri,
                       DmtAcl acl)
                throws DmtException
```

Set the Access Control List associated with a given node.
Parameters:
`nodeUri` - the URI of the node
`acl` - the Access Control List to be set on the node
Throws:
DmtException - with the following possible error codes
- `NODE_NOT_FOUND`
- `URI_TOO_LONG`
- `INVALID_URI`
- `PERMISSION_DENIED`
- `OTHER_ERROR` if the URI is not within the current session's subtree
- `COMMAND_NOT_ALLOWED`
- `DATA_STORE_FAILURE`

`java.lang.IllegalStateException` - if the session is invalidated because of timeout, or if the session is already closed.

6.14.16 getRootUri

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

Get the root URI associated with this session. Gives "." if the session was created without specifying a root, which means that the target of this session is the whole DMT.
Returns:
the root URI

---

7 Considered Alternatives

8 Security Considerations

9 Document Support

9.1 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]. IETF RFC2578. Structure of Management Information Version 2 (SMIv2). April 1999.

[4]. Java™ Management Extensions Instrumentation and Agent Specification, v1.2, October 2002

[5]. Federated Management Architecture (FMA) Specification, Version 1.0, January 2000

[6]. WEBM Profile Template, DSP1000 Status: Draft, Version 1.0 Preliminary, March 11, 2004

[7]. SyncML Device Management Protocol, Version 1.1.2, Open Mobile Alliance, OMA-SyncML-DMProtocol-V1_1_2-20031203-A

[8]. SyncML Device Management Tree and Description, Version 1.1.2, Open Mobile Alliance, OMA-SyncML-DMTND-V1_1_2-20031202-A

[9]. SyncML Device Management Standardized Objects, Version 1.1.2, Open Mobile Alliance, OMA-SyncML-DMStdObj-V1_1_2-20031203-A

[10]. RFC 0078 MEG High-level Architecture

[11]. RFP 0058 Device Management

[12]. RFP 0055 MEG Policy Framework

[13]. RFP 0062 MEG Glossary

[14]. RFC 0087 DMT Structure

All Page Within This Box

[15]. OSGi R3 specification

[16]. RFC 0097 Generic Event Mechnanism

9.2 Author's Address

Name	Vadim Draluk
Company	Motorola
Address	
Voice	
e-mail	vdraluk@motorola.com

Name	Balazs Godeny
Company	Nokia
Address	1092 Budapest Koztelek u. 6. Hungary
Voice	
e-mail	balazs.godeny@nokia.com

9.3 Acronyms and Abbreviations

9.4 End of Document

All Page Within This Box