OMA Device Management Tree and Description

Candidate Version 1.3 – 06 Mar 2012

Open Mobile Alliance

OMA-TS-DM_TND-V1_3-20120306-C

Contents

Figures

Tables

1. Scope

This document defines the management tree and the Nodes on which the OMA DM protocol acts. A standardized way to describe these Nodes is also specified.

2. References

2.1 Normative References

[AMT] Assigned media types. Internet Assigned Numbers Authority.
URL:http:// www.iana.org

[DMBOOT] "OMA Device Management Bootstrap, Version 1.3". Open Mobile Alliance™.
OMA-TS-DM_Bootstrap-V1_3.
URL:http://www.openmobilealliance.org

[DMDDFDTD] "OMA DM Device Description Framework DTD, Version 1.3". Open Mobile Alliance™.
URL:http://www.openmobilealliance.org/tech/DTD/dm_ddf-v1_3.dtd

[DMDICT] "OMA Device Management Dictionary, Version 1.0". Open Mobile Alliance™.
OMA-SUP-DM_Dictionary-v1_0.
URL:http://www.openmobilealliance.org/

[DMPRO] "OMA Device Management Protocol, Version 1.3". Open Mobile Alliance™.
OMA-TS-DM_Protocol-V1_3.
URL:http://www.openmobilealliance.org

[DMSEC] "OMA Device Management Security, Version 1.3". Open Mobile Alliance™.
OMA-TS-DM_Security-V1_3.
URL:http://www.openmobilealliance.org

[DMSTDOBJ] "OMA Device Management Standardized Objects, Version 1.3". Open Mobile Alliance™. OMA-TS-DM_StdObj-V1_3.
URL:http://www.openmobilealliance.org

[DMTNDS] "OMA Device Management Tree and Description Serialization Specification, Version 1.3". Open Mobile Alliance™. OMA-TS-DM_TNDS-V1_3.
URL:http://www.openmobilealliance.org

[ISO8601] ISO 8601:2000, Data elements and interchange formats -- Information interchange -- Representation of dates and times.
URL:http://www.iso.ch/

[META] "OMA Device Management Meta Information, version 1.3". Open Mobile Alliance™.
OMA-TS-DM_MetaInfo-V1_3.
URL:http://www.openmobilealliance.org

[OMNA] "Open Mobile Naming Authority". Open Mobile Alliance™.
URL:http://www.openmobilealliance.org/tech/omna/index.html

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels". S. Bradner. March 1997.
URL:http://www.ietf.org/rfc/rfc2119.txt

[RFC2396] "Uniform Resource Identifiers (URI): Generic Syntax". T. Berners-Lee, et al. August 1998.
URL:http://www.ietf.org/rfc/rfc2396.txt

[SCRRULES] "SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures,
URL:http://www.openmobilealliance.org/

[WBXML1.1] "WAP Binary XML Content Format Specification", WAP Forum™, SPEC-WBXML-19990616.pdf,
URL:http://www.openmobilealliance.org

[WBXML1.2] "WAP Binary XML Content Format Specification", WAP Forum™, WAP-154-WBXML,
URL:http://www.openmobilealliance.org

[WBXML1.3] "WAP Binary XML Content Format Specification", WAP Forum™, WAP-192-WBXML,
URL:http://www.openmobilealliance.org

[XMLSCHEMADT] "XML Schema Part 2: Datatypes", W3C Recommendation 02 May 2001,
URL:http://www.w3.org/XML/Schema/

2.2 Informative References

None.

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

Any reference to components of the DTD's or XML snippets is specified in this typeface.

3.2 Definitions

Kindly consult [DMDICT] for all definitions used in this document.

3.3 Abbreviations

Kindly consult [DMDICT] for all abbreviations used in this document.

4. Introduction

Other OMA DM specifications define the syntax and semantics of the OMA DM protocol. However, the usefulness of such a protocol would be limited if the managed entities in devices required different data formats and displayed different behaviors.

Since device manufacturers will always develop new functions in their devices and since these functions often are proprietary, no standardized Management Objects exist for them. To make these functions manageable in the devices that have them, a device description framework is needed that can provide servers with the necessary information they need to manage the new functions. The intention with this framework is that device manufacturers will publish descriptions of their devices as they enter the market. Organizations operating DM Servers should then only have to feed the new description to their servers for them to automatically recognize and manage the new functions in the devices.

5. The Management Tree

Each device that supports OMA DM MUST contain a Management Tree. The Management Tree organizes all available Management Objects in the device as a hierarchical tree structure where all Nodes can be uniquely addressed with a URI. The following figure shows an example Management Tree.

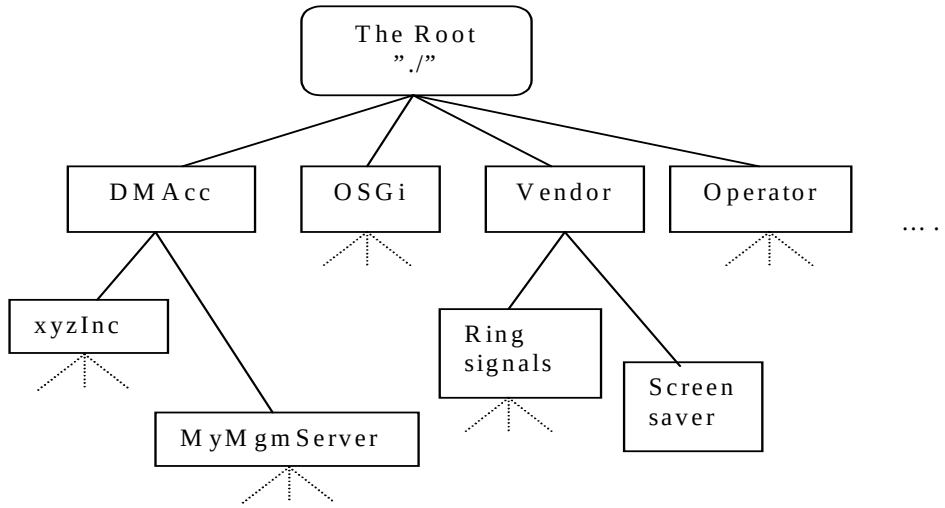

Figure 1: Example Management Tree

To access the xyzInc Node in the Management Tree above, a server MAY for example present the address: ./DMAcc/xyzInc, or DMAcc/xyzInc.

URI used in OMA DM MAY be treated and interpreted as case sensitive. Node names SHOULD be chosen such that resulting URI strings differ in more than just the case of individual letters. Implementations, even if treating and interpreting URIs as case insensitive, SHALL preserve the case of symbols in the names of dynamically created Nodes.

Client device SHOULD indicate the Node name case sensitivity in the DDF using the CaseSense element as specified in Section 10.5.3.

URI used in OMA DM MUST use the UTF-8 character set.

The following restrictions on URI syntax are intended to simplify the parsing of URIs.

- A URI MUST NOT end with the delimiter "/". Note that this implies that the root Node MUST be denoted as "." and not "./".
- A URI MUST NOT be constructed using the character sequence "../". The character sequence "./" MUST NOT be used anywhere else but in the beginning of a URI.

Section 7 explains the different ways which can be used in order to address a node inside the Management Tree.

6. Nodes

6.1 Common Characteristics of Nodes

Nodes are the entities that can be manipulated by management actions carried over the OMA DM protocol. A Node can be as small as an integer or large and complex like a background picture or screen saver. The OMA DM protocol is agnostic about the contents, or values, of the Nodes and treats the Leaf Node values as opaque data.

An Interior Node can have an unlimited number of child Nodes linked to it in such a way that the complete collection of all Nodes in a management database forms a tree structure. Each Node in a tree MUST have a unique URI.

Each Node has properties associated with it; these properties are defined in Section 8.1. All properties, except the ACL, are valid only for the Node to which they are associated.

6.2 Details of the Management Tree

As mentioned before, the complete structure of all Nodes and the root (the device itself) forms a tree. DM Servers can explore the structure of the tree by using the GET command. If the accessed Node is an Interior Node, a list of all child Node names for which the requesting DM Server has the Get access is returned. If the Interior Node has no children, an empty list of child Node names is returned, e.g., <Data/>. If the Node is a Leaf Node it MUST have a value, which could be null, and this value is returned. A DM Server MAY maintain information about its current position in the Management Tree.

Nodes with the Format property set to node are defined as Interior Nodes. Nodes that are not Interior Nodes are defined as Leaf Nodes. Interior Nodes can have 0 or more children; Leaf Nodes cannot have children.

The Management Tree can be extended at run-time. This is done with the Add or Replace command and both new Interior Nodes and new Leaf Nodes can be created. The parent of any new Node MUST be an Interior Node. The device itself can also extend the Management Tree. This could happen as a result of user input or by attaching some kind of accessory to the device.

6.2.1 Addressing Object Values

Node values are addressed by presenting a relative URI for the requested Node. The base URI [RFC2396] MUST be the root of the Management Tree in the device. If a valid URI is presented along with a command for which the current Server Identifier has sufficient access rights, the command MUST be executed. The DM Client MUST respond with an appropriate status code. If the command results in a value that is to be sent back to the DM Server, then this value MUST be part of the response. The value type is returned as part of the meta information as specified in [AMT].

Example:

The following Get command:

```
<Get>
  <CmdID>4</CmdID>
  <Item>
    <Target>
      <LocURI>Vendor/Ring_signals/Default_ring</LocURI>
    </Target>
  </Item>
</Get>
```

could receive a response like this:

```
<Results>
  <CmdRef>4</CmdRef>
  <CmdID>7</CmdID>
  <Item>
    <Data>MyOwnRing</Data>
```

```
    </Item>
  </Results>
```

6.2.2 Addressing Interior Node Child Lists

Interior Node child lists are addressed in the same way as Node values. The list of children is returned as an unordered list of names. The individual names in the list are separated by the character "/". A returned child list has the format node in the meta information of the result. Note that the "/" character MUST NOT be part of any Node names according to [RFC2396].

Example:

The following Get command:

```
<Get>
  <CmdID>4</CmdID>
  <Item>
    <Target>
      <LocURI>Vendor/Ring_signals</LocURI>
    </Target>
  </Item>
</Get>
```

could receive a response like this:

```
<Results>
  <CmdRef>4</CmdRef>
  <CmdID>7</CmdID>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
      <Type xmlns='syncml:metinf'>text/plain</Type>
    </Meta>
    <Data>Default_ring/Ring1/Ring2/Ring3/Ring4</Data>
  </Item>
</Results>
```

If a Get command is issued on an Interior Node that does not have any children, the client MUST return an empty child list, e.g., <Data/>, and the status code (200) OK.

6.2.3 Permanent and Dynamic Nodes

Nodes in the Management Tree can be either permanent or dynamic.

Permanent Nodes are typically built in at device manufacture. Permanent Nodes can also be temporarily added to a device by, for instance, connecting new accessory hardware. However, a DM Server cannot create or delete permanent Nodes at run-time. An attempt by a DM Server to delete a permanent Node will return status (405) Command not allowed. The same status code will also be returned for all attempts to modify the Name property of a permanent Node.

Dynamic Nodes can be created and deleted at run-time by DM Servers. The Add command is used to create new Nodes. The Delete command is used to delete Dynamic Nodes and all their properties. If a deleted Node has children, i.e., is an Interior Node, the children MUST also be deleted.

Note that a permanent node MAY be the child of either a dynamic or a permanent node. In such cases, the permanent child node is created at the same time its parent node is created.

The complete layout of the permanent Nodes in the Management Tree is reflected in the device description.

6.2.4 Adding Dynamic Nodes

As stated in the previous section, the Add command is used to create new Nodes. The definition of the Add command in [DMPRO] states that the Node addressed MUST NOT exist in the tree. The name of the newly created Node SHALL be the last segment of the URI presented in the Target element of the Add command. Note that devices might have various constraints as to the lengths of the URI used to address the Management Tree. These constraints MUST be recorded in the `./DevDetail` Management Object as specified in [DMSTDOBJ].

Example:

The following Add command:

```
<Add>
  <CmdID>5</CmdID>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>b64</Format>
      <Type xmlns='syncml:metinf'>audio/myMelodyType</Type>
    </Meta>
    <Target>
      <LocURI>Vendor/Ring_signals/My_beep</LocURI>
    </Target>
    <Data>jkhdsfKJhdsf89374h</Data>
  </Item>
</Add>
```

would create a previously non-existing Node called `My_beep` as a child Node to the Interior Node `Vendor/Ring_signals`.

When a Node is created, all the properties that this Node supports are automatically created by the client. Those property values that depend on information present in the Add or Replace command, e.g., Name, are assigned these values. Properties with a default value (e.g., Name) or a value that is automatically updated by the client (e.g., Size) are also assigned an appropriate value at Node creation. Other properties will have no value.

Interior Nodes are created in the same way. The difference is that the server MUST explicitly say that the new Node is an Interior Node by including a Meta element with a Format value of node.

Example:

```
<Add>
  <CmdID>5</CmdID>
  <Item>
    <Target>
      <LocURI>Vendor/Ring_signals/MyOwnSongs</LocURI>
    </Target>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
  </Item>
</Add>
```

6.2.5 Protecting Dynamic Nodes

Some Dynamic Nodes can be part of a larger context and only have meaning in this context. Alternatively, the context (a Management Object) might become useless if a detail is lost, e.g., by the deletion of a dynamic Leaf Node. This could, for instance, be the address of an SMTP server for an e-mail Management Object. To protect this kind of Node from deletion and thus maintain the integrity of the context, the DDF property `AccessType` can be set so that the Delete command is excluded. This means that a Delete command will fail with status `(405) Command not allowed`.

Note that this is not the same thing as a permanent Node. A Node protected by the `AccessType` property can still be deleted if it is part of a sub-tree that is being deleted. The Delete command acts strictly on the Node it is addressed at; if that Node is a dynamic Interior Node, it will be deleted along with its child Nodes. If any of these child Nodes have the `AccessType` property set to exclude Delete, they will be deleted anyway. The following table summarizes the protection mechanisms for Dynamic Nodes.

		How the Node is addressed	
		Directly by URI in a Delete command	Indirectly, as a child of a Node being deleted
What `AccessType` specifies for the Node	Delete allowed	Deleted	Deleted
	Delete not allowed	Not Deleted	Deleted

Table 1: Protection Mechanisms for Dynamic Nodes

Note that this mechanism is completely independent of the ACL.

See chapter 10 for more information on DDF properties.

7. Node addressing

Each node MUST be addressed by a unique Full Device URI. The Full Device URI supports three formats:

- An absolute URI start from the device root node. For example ‘./Vendor/A1/E/G’.
- A URI relative to the device root node. For example ‘Vendor/A1/E/G’.
- A URI relative to the parent node of the management object which starts with [x]. The [x] represents the virtual root of the management object. For example ‘[x]/E/G’.

URIs MUST follow requirements specified in Uniform Resource Identifiers (URI) [RFC2396] with the restrictions as specified in OMA Device Management Tree and Descriptions **Error! Reference source not found.**. Node addressing is defined in **Error! Reference source not found.**.

DM supports three types of node addressing

- Absolute URI addressing
- Relative URI addressing
- Virtual URI Addressing.

Both DM Client and DM Server MUST support these three URI addressing mechanisms.

7.1 Absolute URI Addressing and Relative URI Addressing

In absolute URI addressing, the URI specified in the Source and Target element MUST be an absolute URI which start from the device root. In relative URI addressing, the URI specified in the Source and Target element MUST be a URI relative to the device root.

7.2 Virtual URI Addressing

In case when the DM Server does not specify the absolute URI or relative URI, the possibility is to address the node using virtual URI addressing mechanism defined below. There are two URIs used in this mechanism.

- URI A: This URI is specified in the TargetParent element and is used to identify the location of the Management Object that is to be managed.
- URI B: This URI is specified in the Target element and is used to indicate the node to be manipulated in the DM Session. This URI MUST be a URI relative to the parent node of the management object which starts with [x]. The [x] represents the virtual root of the management object.

The DM Client MUST resolve the URI A into the absolute URI of the targeted MO node. This absolute URI is constructed by starting at the device root and, as tree is traversed down to the root node of the targeted MO. The DM Client MUST resolve the URI B into the relative URI of the targeted node. This relative URI is constructed by starting at the root node of the MO, as tree is traversed down to the targeted node.

The DM Server SHOULD specify enough parameters in URI A in order for the DM Client to find unique Management Object instance to be managed. In case multiple Management Object instances were found, the DM Client MUST perform the DM commands on all targeted nodes.

When the DM Client successfully resolved the URI A and URI B, the DM Client MUST concatenate the resolved URI A and resolved URI B to get the absolute URI of the node to be managed.

The syntax definition for URI A is <***URI***>**?*MOID*=<*m-value*>&<*attribute*>=<*a-value*>** as described below:

- <URI>: The start point for the DM Client is to find the MO occurrences in the whole sub-tree which begins from 'URI'. This element SHOULD be included in the virtual URI addressing.
- ?: This is the separator between the 'URI' and MOID. This element MUST be included in the virtual URI addressing only when URI is present.
- MOID=<m-value>: This element is used to specify the MO identifier which identifies the MO that is to be managed. This element MUST be included in the virtual URI addressing. The character ":" MUST be percent encoded as "%3A" within the <m-value>.
- &: This is the separator between the MOID and the attribute condition and between consecutive attribute conditions. This element MUST be included in the virtual URI addressing if the '<attribute>=<a-value>' part is present and when multiple <attribute>=<a-value> pairs are specified
- <attribute>=<a-value>: This is only used when the DM Server anticipates that multiple MO occurrences will be found. The <attribute> identifies the specific leaf node's URI relative to the root of the MO. The <a-value> identifies the value of this leaf node. This part is used by the DM Client to find the unique MO occurrence that is to be managed. This element MAY be included in the virtual URI addressing. If this element is specified, the preceding '&' MUST be specified as well. Reserved characters (as defined in the section 3.4 "Query Component" of [RFC2396]) MUST be percent encoded appropriately. If the DM Server needs more attribute conditions to specify a unique MO, "<attribute>=<a-value>" can occur multiple times with different attributes.

The URI A MUST be included in the <TargetParent>/<LocURI> element. The URI B of virtual URI MUST be included in the <Target>/<LocURI> element if it is present. For Add, Replace and Exec commands, the <TargetParent> and the <Target> elements MUST be included in the DM Message. For Delete command, the <TargetParent> element MUST be included in the DM Message and the <Target> element MAY be included in the DM Message.

The DM Client MUST resolve the virtual URI to actual absolute URI if it supports virtual URI addressing. The DM Client MUST find all MO occurrences in the whole sub-tree according to the specified MO identifier. In case there are multiple MO occurrences found and "<attribute>=<a-value>" pairs are specified in the request, the DM Client MUST use the "<attribute>=<a-value>" pairs to resolve the root URI of the unique MO occurrence that is to be managed.

In case the DM Server wishes to delete the whole MO occurrence(s), the URI B is not needed. Then the URI A will be enough to resolve the root URI(s) of the MO occurrence(s) to be removed from the Device. In this case, the <Target> element will not be included in the <Delete> command of the DM Message.

There are two scenarios about how to address failure:

- Within normal DM Session:
  - If the DM Client failed to resolve the virtual URI to actual absolute URI, the DM Client MUST return status code 400 ("Bad Request").
  - If the DM Client failed to find the unique MO occurrence when <attribute>=<-a-value> pairs were specified, the DM Client MUST return status code 404 ("Not Found").
- Within Sessionless or Bootstrap Session:
  - Since the response is not expected by the DM server, no status code will be returned. If the DM Client failed to resolve the URI to absolute URI, or failed to find the unique MO occurrence, the DM Client MUST NOT perform the command that targeted the virtual URI.

The virtual URI concept is illustrated in the following diagram:

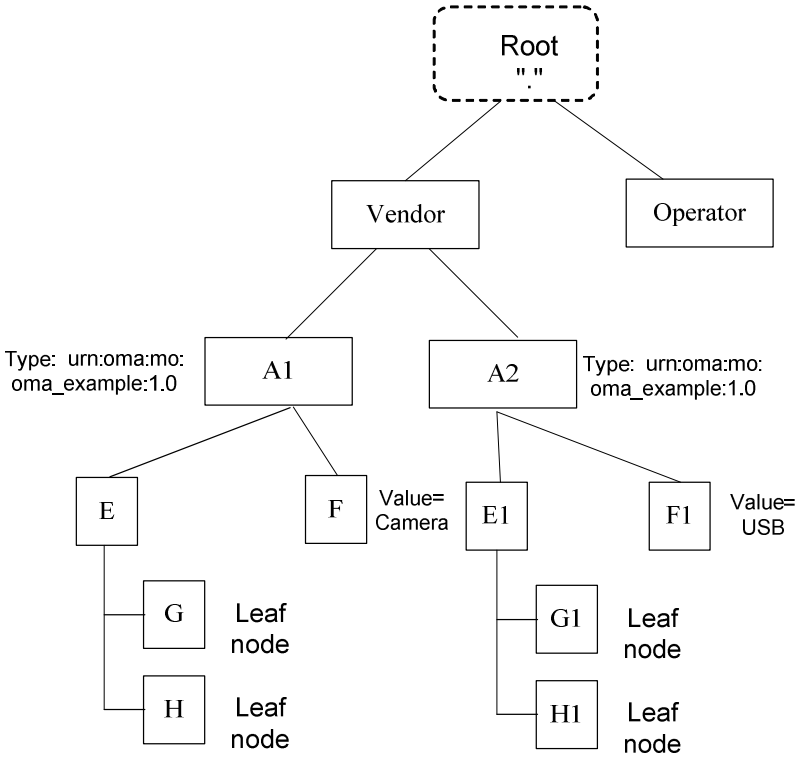

Figure 2: Virtual URI Concept

In this example the DM Server wants to manipulate node './Vendor/A1/E/G'. The URI A sent by the DM Server is "`./Vendor?MOID=urn:oma:mo:oma_example:1.0&F=Camera`". The URI B sent by the DM Server is "[x]/E/G". The example message sent by the DM Server is shown below:

```
<Exec>
  <CmdID>4</CmdID>
  <Item>
    <TargetParent>
      <LocURI>./Vendor?MOID=urn%3Aoma%3Amo%3Aoma_example%3A1.0&F=Camera</LocURI>
    </TargetParent>
    <Target>
      <LocURI>[x]/E/G</LocURI>
    </Target>
  </Item>
</Exec>
```

The resolved URI from the URI A is "./Vendor/A1" and the resolved URI from the URI B is "A1/E/G". Therefore the actual absolute URI of the node G is "./Vendor/A1/E/G". Then the DM Client is able to execute the node './Vendor/A1/E/G as expected by the DM Server.

The more complete virtual URI concept is illustrated in the following diagram:

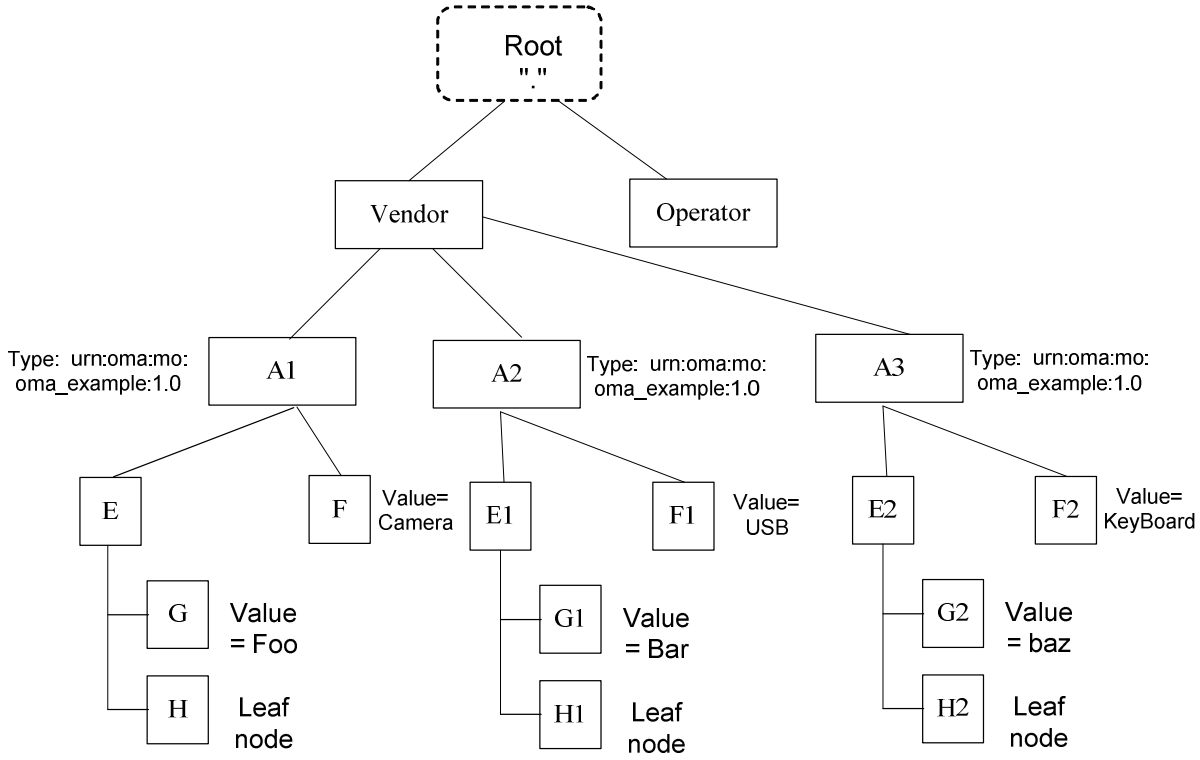

Figure 3: Virtual URI Concept

In this example, the DM Server wants to manipulate node ‘./Vendor/A1/E/H’. The URI A sent by the DM Server is “`./Vendor?MOID=urn:oma:mo:oma_example:1.0&F=Camera&E/G=Foo`”. The URI B sent by the DM Server is “[x]/E/H”. The example message sent by the DM Server is shown below:

```
<Exec>
  <CmdID>4</CmdID>
  <Item>
    <TargetParent>

<LocURI>./Vendor?MOID=urn%3Aoma%3Amo%3Aoma_example%3A1.0&F=Camera&E/G=Foo</LocURI>
    </TargetParent>
    <Target>
      <LocURI>[x]/E/H</LocURI>
    </Target>
  </Item>
</Exec>
```

The resolved URI from the URI A is “./Vendor/A1” and the resolved URI from the URI B is “A1/E/H”. Therefore the actual absolute URI of the node G is “./Vendor/A1/E/H”. Then the DM Client is able to execute the node ‘./Vendor/A1/E/H as expected by the DM Server.

8. Properties of Nodes

8.1 Definition

Properties of Nodes are used to provide meta information about the Node in question. All properties in this section are run-time properties, e.g., they are available during the lifetime of their associated Node. Section 10.5.3 deals with the properties used in the context of device descriptions, which are completely separate from the run-time properties dealt with here.

Property	Explanation
ACL	Access Control List.
Format	Specifies how Node values should be interpreted.
Name	The name of the Node in the tree.
Size	Size of the Node value in bytes.
Title	Human readable name.
TStamp	Time stamp, date, and time of last change.
Type	The MIME type of a Leaf Node's value or a URN representing the Management Object identifier for Interior Nodes that root a Management Object sub-tree.
VerNo	Version number, automatically incremented at each modification.

Table 2: Run-time properties

It MUST NOT be possible to create new properties in an existing device.

8.2 Supported Properties

DM Client MAY support different sets of properties. Some properties are OPTIONAL for a DM Client to implement, but all are REQUIRED for DM Servers. The following table defines property support for devices.

Property	Device Support
ACL	MUST
Format	MUST
Name	MUST
Size	MAY for Leaf Nodes; MUST NOT for Interior Nodes
Title	MAY
TStamp	MAY
Type	MUST
VerNo	MAY

Table 3: Property Support for Devices

8.3 Property Addressing

The properties of a Node are addressed by appending `?prop=<property_name>` to the Node's URI. For instance, to access the ACL of an OMA DM account, a DM Server could use one of these URIs:

```
./DMAcc/xyzInc?prop=ACL
DMAcc/xyzInc?prop=ACL
```

If a DM Server addresses an unsupported property in a DM Client, an error is returned in the form of an `(406) Optional feature not supported` status.

8.4 Property Values

Property values MUST be transported by OMA DM as UTF-8 encoded strings. Numerical property values MUST be converted to numerical strings, expressed in decimal. It is NOT RECOMMENDED to use a <Meta> element for property values.

It is unnecessary to use a <Meta> element for property values because they are all strings. This means that they would all have the same <Meta>, like this:

```
<Meta>
  <Format xmlns='syncml:metinf'>chr</Format>
  <Type xmlns='syncml:metinf'>text/plain</Type>
</Meta>
```

8.5 Operations on Properties

The following table defines the allowed operations for each property. An operation on a property is equivalent to an OMA DM command that is performed by a server on the URI of the property.

Property	Applicable Commands	Comment
ACL	Get, Replace	Get and Replace are the only valid commands for ACL manipulation. Note that Replace always replaces the complete ACL.
Format	Get	Automatically updated by Add and Replace commands on the associated Node.
Name	Get, Replace	A Replace is equivalent to a rename of the Node.
Size	Get	Automatically updated by the DM Client.
Title	Get, Replace	Only updated by DM Server actions or software version changes.
TStamp	Get	Automatically updated by the device.
Type	Get	Automatically updated by Add and Replace commands on the associated Leaf Node.
VerNo	Get	Automatically updated by the device.

Table 4: Allowed Operations on Properties

Properties do not support the Add command. All mandatory properties—and those optional properties that a device implements—MUST be automatically created when a new Node is created. The values of the newly created Node properties are all empty, e.g., `<Data/>`. However, Node properties that have a default value or are automatically updated by the device MUST be assigned appropriate values by the device.

Use of an unsupported command on a property will result in an error and the status `(405) Command not allowed` is returned.

Property values MAY also change for reasons other than direct server operations. For instance, some DM Client MAY allow the user to modify the ACL. If this occurs and the device supports the TStamp or VerNo properties, these MUST be updated.

8.5.1 Properties of Permanent Nodes

The semantics of most properties are independent of the permanent/dynamic status of the Node to which they are associated. The exception is the Name, Format, and Type properties, which MUST NOT be changed for permanent Nodes. Any attempt to perform such a change SHALL fail.

8.6 Scope of Properties

With the exception of the ACL property, all properties are only applicable to the Node with which they are associated. Properties are individual characteristics of each Node. There SHALL be no inheritance of property values—implied or specified—other than for the ACL property.

8.7 Detailed Description of Properties

8.7.1 ACL

The ACL property has some unique characteristics when compared to the other properties.

The access rights granted by an ACL are granted to Server Identifiers and not to the URI, IP address, or certificate of a DM Server. The Server Identifier is an OMA DM specific name for a server. A Management Session is associated with a DM Server Identifier through OMA DM authentication [DMSEC]. All management commands received in one session are assumed to originate from the same DM Server.

8.7.1.1 ACL and Inheritance

Every Node MUST implement the ACL property, but there can be no guarantee that the ACL of every Node has a value assigned to it. However, the root Node MUST always have an ACL value. If a DM Server performs a management operation on a Node with no value set on the ACL, the DM Client MUST look at the ACL of the parent Node for a value. If the parent does not have a value for the ACL, the DM Client MUST look at the ACL of the parent's parent and so on until an ACL value is found. This search will always result in a found value since the root Node MUST have an assigned ACL value. This way, Nodes can inherit ACL settings from one of their ancestors.

Inheritance only takes place if there is no value assigned to the complete ACL property, i.e., there are no commands present. As soon as any value for an ACL property is present, this value is the only valid one for the current Node. ACL values MUST NOT be constructed by concatenation of values from the current Node and its ancestors.

If an ACL does not contain any Server Identifier for a particular command, then this command MUST NOT be present in the ACL. Inheritance does not take place on a per-command basis. Whenever an ACL is changed by a server or by the client itself, care needs to be taken so that command names without Server Identifiers are not stored in the new ACL, resulting in an improperly formatted ACL.

If the ACL for a Node has no value, when a Get command is performed to get the ACL property of this node, the client MUST check its parent node and return the inherited ACL to the server. If the parent does not have a value for the ACL, the device MUST look at the ACL of the parent's parent and so on until an ACL value is found, then the client MUST return the status code `(217) OK with Inherited ACL` to the server.

8.7.1.2 Initial Access Right to Management Tree

The root Node is special – it is owned by the device. It is also the only Node that MUST have a value assigned to the ACL. The default value for the root ACL, for a Device that is not bootstrapped, SHOULD be `Add=*&Get=*`. Any attempt by a DM Server to modify the ACL value of the root SHOULD fail with the status code `(405) Command not allowed`.

The initial access rights to the Management Tree assigned to the DM Server SHOULD be based on Device policy or it MAY be controlled by the Bootstrap Config MO [DMBOOT].

8.7.1.3 Changing the ACL

The rules for changing the ACL of a Node are different for Interior Nodes and Leaf Nodes.

- Interior Nodes
  The ACL is valid for the Node and all properties that the Node may have, i.e., the right to access the ACL is controlled by the ACL itself. If a Server Identifier has Replace access rights according to the Node ACL, then this Server Identifier can change the ACL value.

- Leaf Nodes
  The ACL is valid for the Node value and all properties that the Node may have, except the ACL property itself. If a Server Identifier has Replace access rights according to the Node ACL, then this Server Identifier can change the Node value and all property values, but not the ACL value.

However, for both types of Nodes the right to change the ACL of the Node is also controlled by the ACL of the parent Node. Note that any parent Node is by definition an Interior Node. This makes it possible for a Server Identifier with sufficient access to a parent Node to take control of a child Node. This is a two-step process where the server first changes the ACL of the child Node and then accesses the Node value, list of children, or other Node properties. Note that even if a DM Server has total access to the parent Node according to the parent's ACL, this does not imply direct access to the child Node value. To change a child Node value the child ACL value MUST be changed first.

The ability for a Server Identifier with access to a parent Node to take control of a child Node implies that any Server Identifier with control of the root Node can take control of the complete Management Tree. Doing so is a laborious process that involves many separate management commands being issued by the DM Server. To avoid such a laborious process, once the DM Client received the command to change the ACL value of a Node, the DM Client MUST allow the DM Server to change the ACL of this node in case the server has the Replace right of its parent or ancestor Node. It also implies that unless two Server Identifiers agree about passing authority between them, transition of authority cannot take place. This also makes "hostile takeovers" of devices impossible. To provide the end user with the ability to change which Server Identifier controls the root Node, some devices MAY implement a UI for this purpose.

DM Servers can explicitly set ACL values by performing a Replace operation on the ACL property of any given Node. A successful completion of such an operation is indicated by an `(200) OK` status code. If the operation fails due to lack of device memory, status code `(420) Device full` is returned. In addition, if the reason for failure is access violation, the status code `(425) Permission denied` is returned.

If a DM Server successfully creates a new Node with the Add command, the value of the Node's ACL property is initially set to no value, e.g., `<Data/>`. This means that the value is inherited from the parent Node. However, there is one exception to this rule. If a DM Server is adding an Interior Node and does not have Replace access rights on the parent of the new Node, the Node MUST not inherit the ACL value from its parent Node. Instead, the ACL of the new Node MUST be automatically set so that the creating server has full access rights on the new Node.

In cases where the above rule does not apply, it is RECOMMENDED that the current Server Identifier explicitly set the new Node ACL. This is achieved by using a Replace command on the ACL URI of the new Node. The current server SHOULD set the ACL value so that it has Delete, Get, and Replace access.

Note that since the only command available to change an ACL is Replace, all existing Server Identifiers and access rights are overwritten. If a server wishes to keep the existing entries in an ACL, it MUST read the ACL, perform the needed changes, and then Replace the existing ACL with the new one.

8.7.1.4 Deletion of DM Account

When a DM Account is to be deleted from the device, the Management Tree MUST be scanned for Nodes with ACLs held by the soon-to-be-deleted Server Identifier corresponding to the DM account. All the references to this Server Identifier MUST be deleted. In the event that this process removes the only Server Identifier for a particular command on a particular Node, then this command is removed from the ACL for this Node. Note that if all commands are removed from an ACL in this process—resulting in an ACL with no value—the ACL becomes inherited (see Section 8.7.1.1).

In the event that this process removes the Server Identifier for Replace and Delete commands on a particular Node, the DM Client MUST scan all of its ancestor Nodes to check whether any of them has Replace or Delete rights assigned to any DM Server. If no DM Server has Replace or Delete access rights to all of the ancestor Nodes of this original Node, then the 'Replace=*' access rights MUST be added into the ACL for this Node.

In the event that this process removes the only Server Identifier for a command in the root ACL, the ACL for that command MUST become "*" or a suitable factory default. This is to comply with the restriction on the root ACL specified in Section 8.7.1.2 in this document.

8.7.1.5 ACL Syntax

The ACL structure is a list of Server Identifiers in which each identifier is associated with a list of OMA DM command names [DMPRO]. The right to perform a command is granted if an identifier is associated with the name of the command that is to be performed.

The Server Identifier can also have a wildcard value assigned to it. This means that any Server Identifier used to access the Node and/or its properties is granted access.

ACLs are carried over OMA DM as a string. The string MUST be formatted according to the following simple grammar.

```
<acl> ::= <acl-value> | ”No value”

<acl-value> ::= <acl-entry> | <acl-value> & <acl-entry>

<acl-entry> ::= <command> = <server-identifiers>

<server-identifiers> ::= <server-identifier> | <server-identifier> +
<server-identifiers>

<server-identifier> ::= * | “All printable characters except ‘=’, ‘&’,
‘*’, ‘+’ or white-space characters.”

<command> ::= Add | Delete | Exec | Get | Replace
```

For uniqueness, it is RECOMMENDED that the Server Identifier contains the domain name of the server. For efficiency reasons, it is also RECOMMENDED that it is kept as short as possible. The wild card value for a Server Identifier is character ‘*’. If a <server-identifier> has the value ‘*’, then there SHOULD NOT be any other <server-identifier> values associated with this command in the current ACL. If an ACL entry contains both a wild card, ‘*’, and a <server-identifier>, the access right granted by the <server-identifier> is overridden by the wild card.

Example ACL value:

```
Add=www.sonera.fi-8765&Delete=www.sonera.fi-
8765&Replace=www.sonera.fi-8765+321_ibm.com&Get=*
```

There is no ACL representation for the Copy command. Copy exists as a command on its own mainly for efficiency reasons. Any result of a Copy command can always be created by a sequence of other commands. To successfully execute a Copy command, a server needs to have the correct access rights for the equivalent Add, Delete, Get, and Replace commands.

8.7.1.6 ACL Example

Consider the following Management Tree:

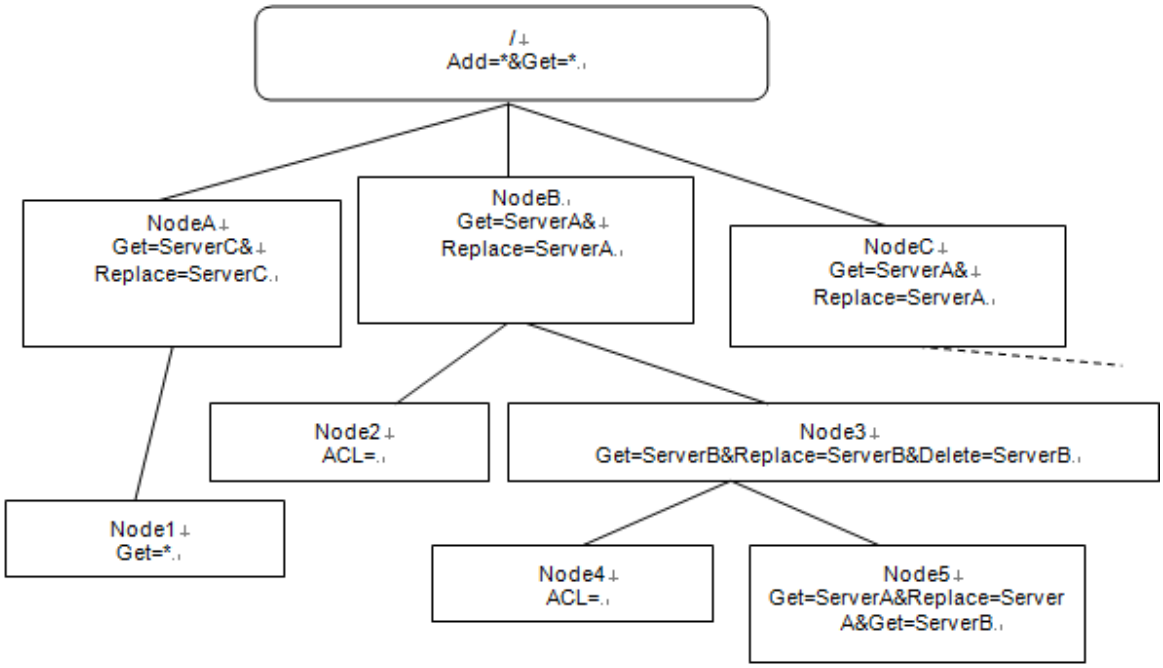

Figure 4: Example Management Tree with ACLs

The following statements about this Management Tree are true:

- Any server can Get the value of `./NodeA/Node1`, but only ServerC can modify `./NodeA/Node1?prop=ACL` in one operation.
- No server can directly Delete or Replace the value of `./NodeA/Node1`.
- A Get request on `./NodeA/Node1?prop=ACL` will return `Get=*`.
- A Get request on `./NodeB/Node3/Node4?prop=ACL` will return `Get=ServerB&Replace=ServerB&Delete=ServerB`, and with the status code (217) OK with Inherited ACL.
- A Replace request on `./NodeB/Node3/Node5` by ServerA will be successful.

8.7.2 Format

The Format property always maintains information about the data format of the current Node value. Allowed formats are defined in [META].

The entity setting value MUST supply the format information in the same command that is used to set the value. The format information is carried (in the DM message) by the Format tag within the Meta element [META] of the Item that has the data to be set. The property value is represented by a string. See section 8.7.7.1 for an example.

Note that Interior Nodes MUST have `node` as the Format value.

When a Node's native value is B64-encoded binary, the value `b64` is used as the node Format property and a Meta Format [META] value of "b64" MAY be used when sent over DM protocol. When data has a binary form (as indicated by the MIME Type in the Meta Type) and the data is sent over XML, then the Meta Format MUST be `b64`. The recipient of this data then decodes the data and associates the node Format property `bin` with the data so that any time a Get command on the Format property of this Node is executed, the response MUST be `bin`.

In effect, the binary data exists as binary on the DM Server and is processed as binary on the DM Client. It MAY be only temporarily encoded in Base64 if it needs to be sent over XML.

When binary data is sent over WBXML:

- The Meta Format MUST be `bin` if the Base64 encoding is not used.
- The Meta Format MUST be `b64` if the Base64 encoding is used.

In either case, the Base64 encoding is used only during transport. The Management Tree property Format MUST be `bin` for this data.

8.7.3 Name

This property reflects the name of the Node to which it belongs. This is the name by which the Node is addressed in the Management Tree. The Name property is a string with a maximum length that is defined in ./DevDetail/URI/MaxSegLen as described in [DMSTDOBJ].

When a new Node is created, the value of the Name property MUST be assigned with the value of the last segment in the Target URI.

This property supports the Replace command. When a Replace command for this property is received by the DM Client, it MUST first check that the result of the command does not lead to an inconsistent tree, e.g., duplicate Node names, before the command is executed. Since it is only the last segment of the current Node's URI that is changed, the search for possible duplicate names can be limited to the siblings of the current Node.

8.7.4 Size

The Size property is used for the current size of the Node value. The property value is a 32-bit unsigned integer.

The value of the Size property MUST be equal to the size of the Node value in bytes and the DM Client is responsible for doing that.

Note that the Size property of a binary data value MUST indicate the size in bytes of the actual (unencoded) value and NOT the length of a Base64 encoded string that may or may not have been used to convey the data over XML.

Also note that the Meta Size tag used in conveying data always indicates the size of the data in the message. If the message is in XML and the data is binary, then the data will be encoded. Consequently, the Meta Size of data that is encoded as `b64` is the length of the Base64 encoded string.

8.7.5 Title

The Title property is used to store a human readable, alphanumeric string that provides some information about the Node to which this property belongs. The Title property is a string with a maximum length of 255 bytes.

8.7.6 TStamp

This property is a record of the date and time of the last change in value of the Node which has this property. The value is represented by a string containing a UTC-based, [ISO8601] basic format, complete representation of a date and time value, e.g., 20010711T163817Z means July 11, 2001 at 16 hours, 38 minutes and 17 seconds.

8.7.7 Type

The Type property is inspired by the concept of typed data in programming languages. For leaf objects, the Type property describes the kind of data stored as the object's value. For interior objects, the Type property identifies the collection rooted at that Interior Node.

8.7.7.1 Leaf Objects

The Type property of a leaf object is always the MIME type of the current object value. Allowed types are defined in [AMT]. The entity setting value MUST supply the type information in the same command that is used to set the value. The Type tag in the Meta information carries the MIME type information for the Item. The property value is represented by a string.

An object's description MAY specify that the object can store more than one MIME type. Consequently, a server that modifies an object's value MUST supply the MIME type of the data when the object value is set.

Example:

The following Add command illustrates how the Type and Format properties are set:

```
<Add>
  <CmdID>4</CmdID>
  <Item>
    <Target>
      <LocURI>Vendor/ISP/yyy/GWName</LocURI>
    </Target>
    <Meta>
      <Format xmlns='syncml:metinf'>chr</Format>
      <Type xmlns='syncml:metinf'>text/plain</Type>
    </Meta>
    <Data>www.yyy.se</Data>
  </Item>
</Add>
```

8.7.7.2 Interior Objects

The Type property of an interior object is represented by a string. The property MAY have no value. When the property does have a value, it MUST represent the Management Object Identifier of the collection of objects rooted at the current object.

The version information in the Management Object Identifier MUST be updated whenever the management object definition changes due to:

- Modification of the management object structure (e.g., add new nodes, rename nodes, remove nodes, change node locations etc.).
- Modification of node data definition (e.g., modify definition for status, occurrence, format, or minimum access type).

Changes to the Ext nodes or in the Ext sub-trees do not require updating of the Management Object Identifier.

The Management Object Identifier SHOULD be a URI. When the Management Object Identifier is registered by the Open Mobile Naming Authority [OMNA], the identifier will most likely be a URN. Some examples are: urn:oma:mo:oma-fumo:1.0, or urn:oma:mo:oma-imps:1.0.

Management Object Identifiers not registered by the Open Mobile Naming Authority MAY use the following reversed domain name scheme:

Reversed Domain Management Object Identifiers

A reversed domain Management Object Identifier is formed by combining the reversed domain name of the owner with the name and version of the object, e.g., "com.companyx/1.2/ProductY". The syntax is as follows:

```
<management_object_name> ::= <reversed-
domain>/<major>.<minor>/<name>[/<name>]*

<reversed-domain> ::= "reversed internet domain name"
```

```
<major> ::= [1-9][0-9]*

<minor> ::= 0 | [1-9][0-9]*

<name> ::= alphanum
```

NOTES:

1. The reversed domain is used in some programming languages to name classes uniquely and MAY include sub-domains, e.g., “com.companyx.producty” (a fictional example).
2. The major and minor elements of the version are considered unsigned integer counters, without leading zeros.
3. For simplicity and portability, each name element is currently restricted to alphanumeric characters. (Alphanum is defined in [RFC2396].)

The domain name of the object owner specifies a namespace. In that namespace, the Management Object Identifier MUST be unique. A DM Server that wishes to obtain the DDF file corresponding to the object does so in an implementation-specific way. The DM Server MAY maintain a local repository of DDF files describing objects that are managed by the server. t is RECOMMENDED that manufacturers and standards organizations maintain Web-accessible repositories of these documents. Then DM Servers could be implemented to obtain DDF files automatically (as needed) by maintaining a small mapping from the organization (domain) to the Web location (URL) of the DDF file, given the Management Object Identifier. Alternatively, another technique could be specified in a future version of OMA DM specifications.

Note that a client may store DDF files internally. Management Object could be designed to provide a consistent location within the Management Tree for these descriptions.

When an object has a DDF description and an ancestor of that object also has a DDF description, the two descriptions SHOULD be consistent. For example, the OMA DM Account information is described by a specific DDF. A manufacturer MAY supply a DDF file for an entire device, which corresponds to the DDF of the root. The information in the root DDF that describes the OMA DM Account information SHOULD correspond to the description in the Management Object Identifier of the DMAcc object. In the case of a conflict, the more local Management Object Identifier MUST be used.

The DM Server provides the Management Object Identifier in the Type property when creating interior objects (see the example below). Device manufacturers would typically provide it for permanent objects.

Example:

The following Add command illustrates how the Type property might be set when an interior object is created:

```
<Add>
  <CmdID>3</CmdID>
  <Item>
    <Target>
      <LocURI>Vendor/ISP/yyy/Profile</LocURI>
    </Target>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
      <Type xmlns='syncml:metinf'>se.yyy.dm/2.10/Profile/Class1</Type>
    </Meta>
  </Item>
</Add>
```

Note that in the example above, a sub-domain is used (dm.yyy.se), the major version is 2 (two), the minor version is 10 (ten), and the name is “Profile/Class1”.

The Management Object Identifier is intended to be useful under the following circumstances. A DM Server MAY use DDF definition files from different devices to identify which positions the different devices store Management Objects. A DM Server that examines the Management Tree in a device can use the Management Object Identifier to obtain a description of an object that was previously unknown to this server. This situation could arise in many ways, for example:

- A different DM Server in the same organization may have created the collection of objects in the device.
- A DM Server from a different organization may have created the collection of objects in the device.
- The collection of objects could have been added to the device's Management Tree by software in the device itself, perhaps in response to the addition of a piece of hardware to the device (e.g., a card inserted into an expansion slot).

8.7.8 VerNo

VerNo is a 16-bit unsigned integer. Each time a Node with this property changes value—through a management operation or other event—this value SHALL be incremented. If the property value has reached $FFFF_{16}$, and then is incremented, it SHALL return to 0000_{16}.

9. Device Management Tree Exchange

The Management Tree is a hierarchical arrangement of managed objects in a device that defines the management view of the device. To effectively manage a device, the DM Server SHOULD be able to manage relevant parts of the Management Tree in the device. Without knowing the Management Tree, the DM Server would not be able to access a specific managed object in the device and hence, effectively manage the device.

The method for exchanging Management Tree information between a DM Client and a DM Server involves:

- Representing the wanted information from the tree, without loss of information about its hierarchical structure.
- Sending it in an OMA DM command to the recipient.

Management Tree information can be represented and delivered by using an XML representation as described in section 10.

The OMA DM specifications allow the DM Server to add, replace, and delete parts of a Management Tree in a single package, but to get the information from the client DM Server requires multiple round trips. The following section specifies a way to `Get` a part of a Management Tree in a single package.

9.1 Requesting a Part of a Management Tree

The DM Server uses the `Get` command with an attribute “Struct”, “StructData”, “TNDS”, “MORoot”, or “MORootData” in the <URI> to retrieve the Management Tree information identified by the attribute. The syntax for these structural queries is as follows:

GET <*URI*>**?list=<attribute>**

The attribute is addressed by appending *`?list=<attribute>`* to the retrieved Node’s URI. The attributes are added to the URI specified in the `Item` element inside the `Get` command.

When the 'list=<attribute>' element is specified, it MUST contain exactly one attribute.

The DM Client MUST send all the requested information of the sub-tree that begins at the <URI> node as specified in the table below. Sub-trees rooted at nodes for which the requesting DM Server does not have the Get access SHALL NOT be included in the response.

The valid values for <attribute> are:

Attribute	Description
Struct	The structure of a Management Tree is returned without any data.
StructData	The structure of the Management Tree is returned with the Leaf Nodes data.
TNDS	The returned data is a serialized sub-tree as defined in [DMTNDS].
MORoot	The roots of all MO occurrences corresponding to the specified MO identifier for which the requesting DM Server has the Get access are returned. If this attribute is used, the MO identifier MUST be specified in the Item/Data element within the GET command.
MORootData	The roots of all MO occurrences corresponding to the specified MO identifier for which the requesting DM Server has the Get access, along with the requested Leaf Node’s data and URI, are returned. If this attribute is used, the MO identifier appended with “?” and the requested Leaf Node’s URI relative to the MO Root MUST be specified in the Item/Data element within the GET command. Only one Leaf Node can be requested.

Table 5: Possible Attributes for Management Tree Information Retrieval

The DM Client SHOULD support all the values listed in table 5 above.

In case an attribute value is not supported, the client MUST return status code `406` (`Optional Feature Not Supported`).

The DM Server uses the `Get` command with the attribute "MORoot" in the <URI> to retrieve the roots of MO occurrences identified by the MO identifier. The DM Client MUST return the requested information for all MOs with the specified MO identifier that were found in the whole sub-tree, which begins at the <URI> node.

The DM Server uses the `Get` command with the attribute "MORootData" in the <URI> to retrieve the roots of MO occurrences identified by the MO identifier, along with the requested Leaf Nodes' data and URI. The DM Client MUST return the requested information for all MOs with the specified MO identifier that were found in the whole sub-tree, which begins at the <URI> node.

The attributes are added to the URI specified in the `Item` element inside the `Get` command. The `Get` command and the URI in it have the format as shown in various examples in Appendix C.

9.2 Representation Response of the Management Tree

Representing the tree using multiple `Item` elements inside the OMA DM `Result` command is the simplest approach. Requested information from the Node (URI in the Get command with attribute) is embedded into an `Item` inside a `Result` command with the following requirements (see also the examples in Appendix C):

- *Struct* attribute: `Meta` MUST be used to indicate the `Type` and `Format` of the Node, unless the `Type` and `Format` have the default values [META]. `LocURI` inside the `Source` MUST indicate the URI of the Node.
- *StructData* attribute: `Meta` MUST be used to indicate the `Type` and `Format` of the Node, unless the `Type` and `Format` have the default values [META]. `LocURI` inside the `Source` MUST indicate the URI of the Node. Leaf Node data MUST be included to the `Data` element.
- *TNDS* attribute: `META` MUST be used to indicate the `Type` and `Format` as defined in [DMTNDS]. `LocURI` inside the `Source` MUST indicate the URI of the starting Node. If no extra attributes are included after "TNDS", the DM Client SHOULD include all valid properties that that DM Server has access rights to receive. Optionally, the DM Server MAY request the properties that only SHOULD be included in the Serialized management sub-tree by adding "+" and the property name. More than one property name is allowed in the list. The DM Server MAY also exclude some properties from the default, all property-list, by adding a "-"character followed by the property name. It is also possible to add multiple "-PropertyName" after each other. This include property "+" and remove property "-" cannot be combined in the same Get command. Valid Property names are all properties defined in chapter 6.1 in [DMTNDS], which are inside "RTProperties" and "Value" property. The DM Client MUST return Status Code "406 Optional feature not supported" if the DM Client does not support TNDS or any one of the present properties in the presented list.
  Example on valid property list: "?list=TNDS+Value+ACL" for only data and ACL or "?list=TNDS-Format" for all supported properties except the Format Property.
- *MORoot* attribute: `Meta` MUST be used to indicate the `Type` and `Format` of the Node, unless the `Type` and `Format` have the default values [META]. `LocURI` inside the `Source` MUST indicate the MO occurrence's root.
- *MORootData* attribute: Meta MUST be used to indicate the Type and Format of the Node, unless the Type and Format have the default values [META]. The DM Client MUST return two Items for each instance of MO found in the DM Tree—one Item for the URI of the instance and one Item for the Leaf Node data of the instance. For the Item containing the URI, the Source/LocURI MUST indicate the MO occurrence's root. For the Item containing the Leaf Node data, the requested Leaf Node's URI MUST be stored in the Source/LocURI; the Leaf Node's data MUST be stored in the Data element.

For *Struct* and *StructData*, this information from the requested Nodes MAY be included in multiple `Item` elements inside a `Result` command or there MAY be multiple `Result` commands within a single `Item`.

10.Device Description Framework

10.1 Rationale for a Device Description Framework

In an ideal world, all devices would display the same structure and behavior to a management system. But since different vendors are competing with each other on the market for various kinds of devices, it seems very unlikely that this would ever happen. But management systems still need to understand each individual device even though they do appear to have different internal structures and behaviors.

To address this issue, the concept of a device description framework is introduced. In short, this framework prescribes a way for device vendors to describe their devices so that a management system can understand how to manage the device. The following figure illustrates this principle.

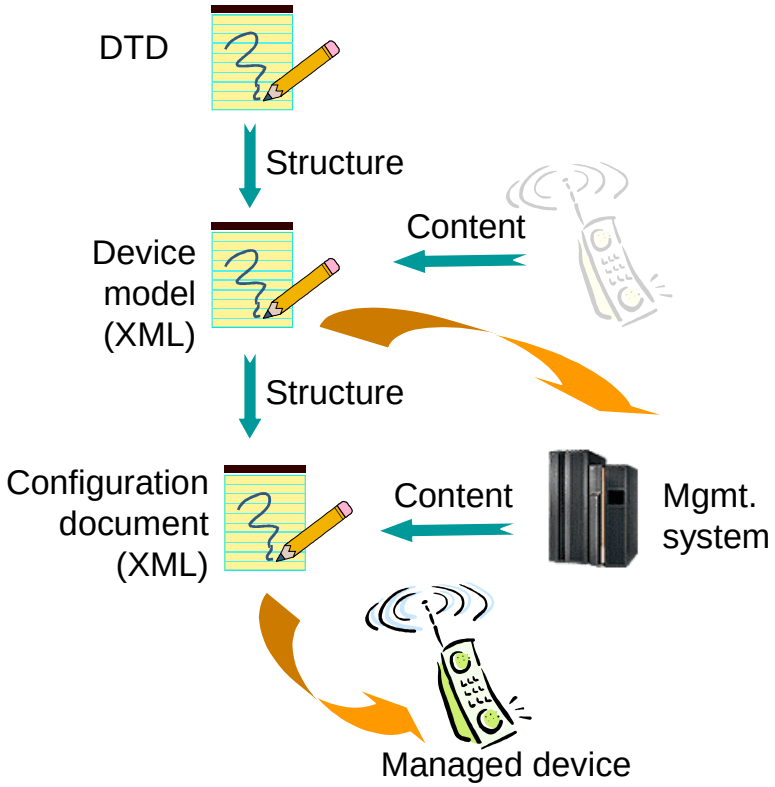

Figure 5: Conceptual View of How a Device Description Framework Is Used

By using a device description framework, we also make sure that the total management system based on OMA DM is flexible, easily extendible, and that it can accommodate not only the demands we put on it today, but also the ones we might have tomorrow. We also avoid a situation in which all future management needs would have to be standardized before they could be used in devices to simplify the use of these devices.

It is important to note that the device description framework needs to co-exist with the existing standardized Management Objects, and that the borderline between the standardized Management Objects and Management Objects described by the framework will change over time. This will mainly happen when a standards body decides to create specifications on how their technology should be managed. It is in the interest of the OMA Device Management committee to encourage and support such initiatives from standard bodies active in wireless standardization so that the set of standardized Management Objects increases.

10.2 The OMA DM Device Description Framework

Today there exist a number of different device description frameworks, but none of these seem to suit the purposes of OMA DM. There are also activities in other standards bodies that aim to develop new frameworks specifically for the wireless industry. With this in mind, OMA DM does not currently specify the use of any particular framework.

However, the need for a device description framework remains and therefore OMA DM RECOMMENDS using the following simple framework as an interim solution. This proposed device description framework is defined by an XML DTD. Descriptions of Management Objects, or complete Management Trees, are valid XML documents. Device manufacturers using the device description framework MUST make the device descriptions available to DM Servers. The mechanism for this is currently not being standardized.

The OMA DM Device Description Framework DTD is defined in [DMDDFDTD].

10.3 XML Usage

The OMA DM DDF information XML documents are specified using well-formed XML; however, they MAY not be valid XML as they do not need to specify the XML prolog.they only need to specify properly identified name space element types from the OMA DM DDF information DTD. This restriction allows for the OMA DM DDF information to be specified with greater terseness than would be possible if a well-formed, valid XML document was REQUIRED.

This DTD makes heavy use of XML name spaces. Name spaces MUST be declared on the first element type that uses an element type from the name space.

Names in XML are case sensitive. The element types in the OMA DM DDF information DTD are defined in [DMDDFDTD] or the URN syncml:dmddf.

OMA DM also makes use of XML standard attributes, such as xml:lang. Any XML standard attribute can be used in an XML document conforming to this DTD.

A document complying with [DMDDFDTD] and which is not a [DMTNDS] document MUST use a specific MIME-type to indicate it. The two MIME-types, one for XML and one for WBXML, are specified in Appendix D.

10.4 Management Objects

Management Objects are logical collections of related nodes that enable the targeting of management operations, using OMA DM Commands. Each node in a MO can be as small as an integer or large and complex like a background picture or screen saver. The OMA DM protocol is agnostic about the contents, or values, of the MO and treats the node values as opaque data.

10.4.1 DDF description and graphical representation of MO

OMA DM MO are described using the OMA DM DDF Files. The use of this description framework produces detailed information about the device in question. The DDF File is a machine readable file describing a MO or how a DM Client has implemented the DM Tree.

In order to make it easier to quickly get an overview of how a MO is organized and its intended use, a simplified graphical notation in the shape of a block diagram is used in the technical specification documents. Even though the notation is graphical, it still uses some printable characters, e.g… to denote the number of occurrences of a node. These are mainly borrowed from the syntax of DTDs for XML. The characters and their meaning are defined in the following table.

Character	Meaning
+	one or many occurrences
*	zero or more occurrences

?	zero or one occurrences

If none of these characters is used the default occurrence is exactly once.

Another feature of the DDF that needs to have a corresponding graphical notation is the un-named block. Un-named are nodes which act as placeholders in the description and are instantiated with information when the nodes are used at run-time. Un-named blocks in the description are represented by less than ("<") and a greater than (">") character containing a lower case character, e.g. ("<x>").

Each block in the graphical notation corresponds to a described node, and the text is the name of the node. If a block contains an <x>, it means that the name is not known in the description and that it will be assigned at run-time. The names of all ancestral nodes are used to construct the URI for each node in the MO. It is not possible to see the actual parameters, or data, stored in the nodes by looking at the graphical notation of a MO.

Some MOs specify explicit names of nodes but the name is still assigned at run-time. These nodes MAY not be described as <x> in the DDF Fileand it is possible to use the syntax [NodeName] where "NodeName" is a logical name for the node. In this case the graphical representation and the DDF File will contain the logical name of the node to improve the readability.

The nodes which the DM Client is required to support are drawn in the graphical notation with solid line, while nodes whose support is not mandatory for the DM Client are drawn with a dotted line.

Leaf nodes are drawn as rectangle while interior nodes are drawn as rectangle with rounded corners.

The following is an example of what a MO can look like when it is expressed using the graphical notation:

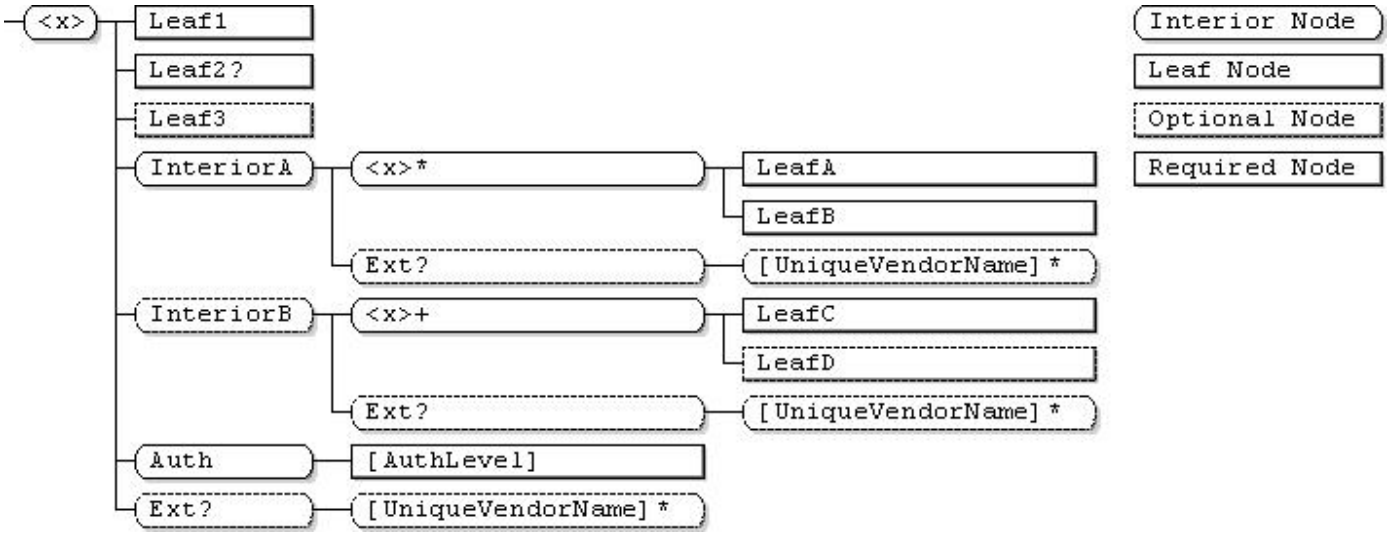

Figure 6: Example of a MO pictured using the graphical notation

Naturally, this graphical overview does not show all details of the full description, but it provides a good map of the description so that it is easier to find the individual node. Although the figure only provides an overall view of the description, there are still some things worth noticing.

All blocks with names in place occur exactly once, except Leaf2, InteriorA/<x>, InteriorB/<x>, all Ext nodes and their children.

Leaf3, InteriorB, all Ext and their children nodes are optional to be supported by the DM Client.

All nodes whose name starts with "Leaf" and the node "[AAuthLevel]" are *leaf* nodes. They MAY contain data but cannot contain child nodes; all other nodes are *interior* nodes, they cannot contain data but can contain child nodes.

The un-named leaf nodes are marked with * or +. This means that although the description only contains one node description at this position in the tree, there can be any number of instantiated nodes at run-time, including none in the first case, at least one in the second. The only limit is that the node names MUST be unique and the DM Client MUST have sufficient memory to store the nodes.

The next figure shows an example of what the device information MO could look like at run-time.

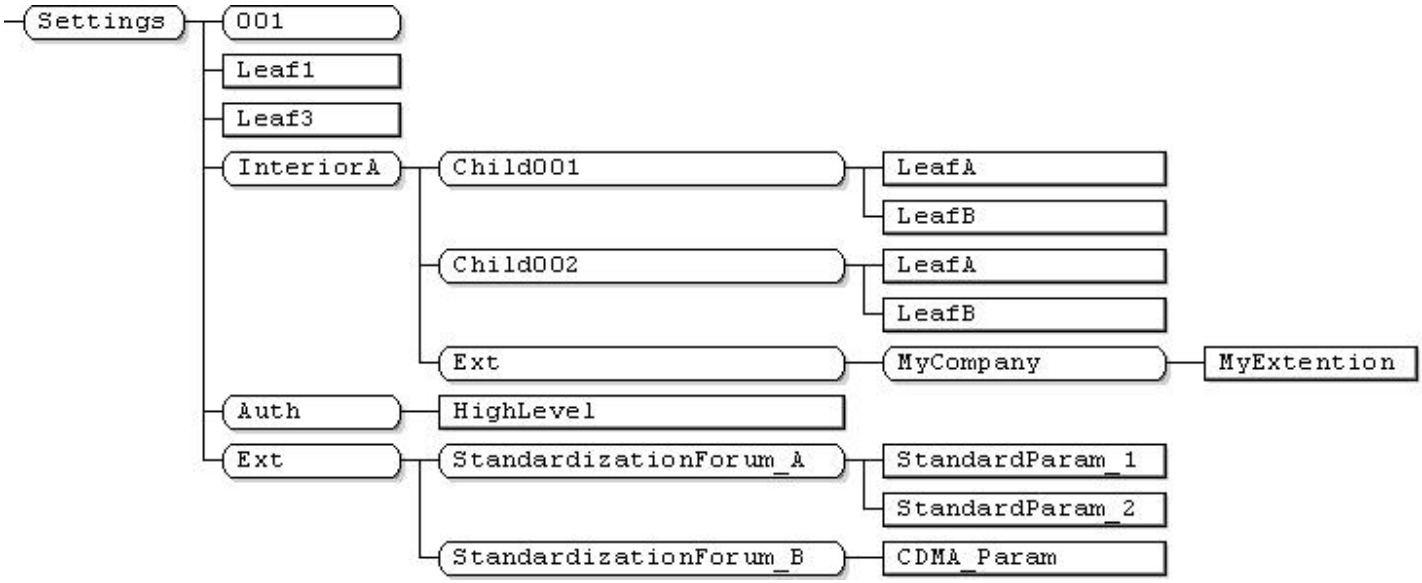

Figure 7: Example of an instance of this MO

The difference between this figure and the previous one is that now the un-named blocks have been instantiated and some optional nodes are not shown.

Note that none of the stored data in the leaf nodes is shown in the figure: only the node names are visible.

10.4.2 DDF compliance

The MO descriptions are normative. However, they also contain a number of informative aspects that could be included to enhance readability or to serve as examples. Other informative aspects are, for instance, the `ZeroOrMore` and `OneOrMore` elements, where implementations MAY introduce restrictions. All these exceptions are listed here:

- All XML comments, e.g. "`<!-- some text -->`", are informative.
- The descriptions do not contain an `RTProperties` element, or any of its child elements, but a description of an actual implementation of this object MAY include these.
- If a default value for a leaf node is specified in a description, by the `DefaultValue` element, an implementation MUST supply its own appropriate value for this element. If the `DefaultValue` element is present in the description of a node, it MUST be present in the implementation, but MAY have a different value.
- The value of all `Man, Mod, Description` and `DFTitle` elements are informative and included only as examples.
- Below the interior `Ext` node, an implementation MAY add further nodes at will.
- The contents of the `AccessType` element MAY be extended by an implementation.

- If any of the following `AccessType` values are specified, they MUST NOT be removed in an implementation: `Copy, Delete, Exec, Get,` and `Replace`.
- If the `AccessType` value of `Add` is specified, the nodeMAY be removed in an implementation if the implementation only supports a fixed number of child nodes.
- An implementation MAY replace the `ZeroOrMore` or `OneOreMore` elements with `ZeroOrN` or `OneOrN` respectively. An appropriate value for `N` MUST also be given with the …`OrN` elements.
- Path element is informative.
- All nodes are included but the specification MAY allow an implementation to not require that all nodes are to be supported.

10.4.2.1 Standardized DDF Files

Since the standardized DDF Files contain machine readable information, the DM Client vendor MAY use standardized DDF Files to verify the implementation compliance to the standard specifications. The standardized DDF File MAY contain additional information compared to what is valid according to the DTD of the DDF Files: for instance, a standardized DDF File contains both mandatory and optional nodes and nodes which occurrence can be `ZeroOrMore`, while an implementation DDF File reports strictly the node which are implemented.

The rules defined by the [DMDDFDTD] define how DDF MUST be handled by the DM Client and the DM Server; in addition to these rules, other information can be helpful for the DM Client vendors in order to create implementation DDF Files that define the DM Tree supported by a DM Client. These DDF Files MAY also be used as input for creating the diagram of the MO Tree structure. For this reason, the following information MAY be included in standardized DDF Files:

- The node status is defined by a XML Comment as next sibling to the node "NodeName"; if the value is "*Required"* then the DM Client MUST support the node (if the parent node is supported); if the value is "*Optional"*, then the node is not unconditional mandatory to support in the implementation.

  The syntax for this is: "<!-- Status: Required -->" or "<!-- Status: Optional -->".

  If an interior node <x> is *Optional* and its child ChildA is *Required*, then DM Client MUST support ChildA only if <x> is supported.

  If the support of a node depends from conditions external to the MO (for instance, if the device support a physical feature, then MO MUST include the specific node), then the node occurrence SHOULD be *Optional*.
- The Path Element is used to define the location of the MO in the DM Tree. If the standard specification does not define a fixed location into the DM Tree, then value of the Path Element is "…"; if the specifications defines an unconditional fixed location into the DM Tree, then the Path Element contains this exact location.
- In addition to specifying the minimum set of DM Commands which the node MUST support, standard specification can specify explicitly the DM Commands which are NOT allowed for that node. This is achieved by adding a XML Comment for each allowed DM Command as child of the "AccessType" Element.

  The valid syntax is to use the word "No" plus the space character plus the unallowed DM Command, for example "<!-- No Get -->".

10.5 Framework Elements

This section explains the elements used in the description framework DTD.

10.5.1 Structural Elements

These elements provide various kinds of structural information of the described Management Object.

10.5.1.1 MgmtTree

Usage: Container for one or more described Management Objects.

Parent Elements: None.

Restrictions: This MUST be the root element of all descriptions.

Content Model: `(VerDTD, Man?, Mod?, Node+)`

10.5.1.2 VerDTD

Usage: Specifies the major and minor version identifier of the OMA DM Device Description Framework specification used to represent the OMA DM description.

Parent Elements: `MgmtTree`

Restrictions: Major revisions of the specification create incompatible changes that will generally require a new parser. Minor revisions involve changes that do not impact basic compatibility of the parser. When the XML document conforms to this revision of the OMA DM Device Description Framework, the value MUST be `1.2` and the element type MUST be included in the `MgmtTree`.

Content Model: `(#PCDATA)`

10.5.1.3 Man

Usage: Specifies the manufacturer of the device.

Parent Elements: `MgmtTree`

Restrictions: This element is OPTIONAL.

Content Model: `(#PCDATA)`

10.5.1.4 Mod

Usage: Specifies the model number of the device.

Parent Elements: `MgmtTree`

Restrictions: This element is OPTIONAL.

Content Model: `(#PCDATA)`

10.5.1.5 Node

Usage: Specifies a Node.

Parent Elements: `MgmtTree`

Restrictions: This element is recursive. A Node with a Value element MUST always terminate the recursion. It is possible for a Node to omit both the next recursive Node and a Value, which means that the hierarchy of Nodes continues elsewhere. This can be used to increase readability of very deep trees. In the continuation, the Path element MUST contain a full URI that specifies the insertion point in the tree.

Content Model: `(NodeName, Path?, RTProperties?, DFProperties?, (Node*|Value?))`

Example: The following XML is a description of a number of Nodes that form the URI `Vendor/ISP/GWInfo/GWName`. Note that all of the details of DFProperties are deliberately left out.

```
<MgmtTree>
  <Node>
    <NodeName>Vendor</NodeName>
    <DFProperties>…</DFProperties>
    <Node>
      <NodeName>ISP</NodeName>
      <DFProperties>…</DFProperties>
      <Node>
        <NodeName>GWInfo</NodeName>
        <DFProperties>…</DFProperties>
        <Node>
          <NodeName>GWName</NodeName>
          <DFProperties>…</DFProperties>
          <Value>gw.yyy.se</Value>
        </Node>
      </Node>
    </Node>
  </Node>
</MgmtTree>
```

10.5.1.6 NodeName

Usage: Specifies the name of the described Node.

Parent Elements: `Node`

Restrictions: See [RFC2396] for general restrictions on URI. The NodeName element MAY be empty. If empty, this means that the name of the Node MUST be assigned when the Node is created. When the Node name is assigned at Node creation time, the value for the name is set to the last segment of the URI specified as Target for the command that results in the Node being created. See also section 8.7.3.

Content Model: `(#PCDATA)`

10.5.1.7 Path

Usage: Specifies the URI up to, but not including, the described Node.

Parent Elements: `Node`

Restrictions: The Path element is OPTIONAL and it is used in standardized DDFs to indicate whether or not the location of the MO in the Management Tree is fixed. It is NOT RECOMMENDED for use by device vendors. If omitted, the Node element MUST be described as the child element of its parent 's Node element. This concatenated value MUST be valid path from the parent node.

Content Model: `(#PCDATA)`

Example: The following XML is an alternative way to describe the same Management Objects as in the example in section 10.5.1.5. This description specifies the same URI as the other example: `Vendor/ISP/GWInfo/GWName.` Note that all the details of DFProperties are deliberately left out.

```
<MgmtTree>
   <Node>
      <NodeName>Vendor</NodeName>
      <DFProperties>…</DFProperties>
   </Node>
   <Node>
      <NodeName>ISP</NodeName>
      <Path>Vendor</Path>
      <DFProperties>…</DFProperties>
```

```
   </Node>
   <Node>
      <NodeName>GWInfo</NodeName>
      <Path>Vendor/ISP</Path>
      <DFProperties>…</DFProperties>
   </Node>
   <Node>
      <NodeName>GWName</NodeName>
      <Path>Vendor/ISP/GWInfo</Path>
      <DFProperties>…</DFProperties>
      <Value>gw.yyy.se</Value>
   </Node>
</MgmtTree>
```

10.5.1.8 Value

Usage: Specifies a default value for Nodes that are instantiated using the current description.

Parent Elements: `Node`

Restrictions: OPTIONAL element. If omitted, the Node description does not specify any default value for the Node. In this case, the initial value of new Nodes is undefined.

Content Model: `(#PCDATA)`

10.5.1.9 RTProperties

Usage: Aggregating element for run-time properties, i.e., properties that the Nodes have in a device at run-time. Used to specify which properties the described Node supports at run-time. Can also be used to supply default values for supported run-time properties.

Parent Elements: `Node`

Restrictions: OPTIONAL element. If omitted, the Node MUST only support the mandatory run-time properties ACL, Format, Name, and Type. If any optional properties are supported, they MUST be specified by using this element.

Content Model: `(ACL?, Format?, Name?, Size?, Title?, TStamp?, Type, VerNo?)`

10.5.1.10 DFProperties

Usage: Aggregating element for device description framework properties, i.e., properties that Nodes have in the device description framework and that are not explicitly present at run-time.

Parent Elements: `Node`

Restrictions: This is a REQUIRED element.

Content Model: `(AccessType, DefaultValue?, Description?, DFFormat, Occurrence?, Scope?, DFTitle?, DFType, CaseSense?)`

10.5.2 Run-time Property Elements

The usage of the run-time properties in the device description framework reflects the mandatory/optional status of the corresponding run-time property. These elements can also be used to describe default values for the supported properties. Since these properties can change at run-time, the values of these properties in the description and in a real device will differ.

The `RTProperties` element, which encapsulates the run-time properties, is OPTIONAL within its parent element `Node`. The purpose of this is to make it possible to omit the `RTProperties` element if only the mandatory properties are supported and there is no need to specify any default values. If the `RTProperties` element is used in a description, all the mandatory run-time properties MUST be specified.

10.5.2.1 ACL

Usage: Specifies support for the ACL property. MAY be used to specify a default value for the property.

Parent Elements: `RTProperties`.

Restrictions: If a value is specified it MUST be formatted according to section 8.7.1.5.

Content Model: `(#PCDATA)`

10.5.2.2 Format

Usage: Specifies support for the Format property. MAY be used to specify a default value for the property.

Parent Elements: `RTProperties`.

Restrictions: If a default value is specified for the described Node, the Format property MUST specify the correct format of the Node value.

Content Model: `(b64 | bin | bool | chr | int | node | null | xml | date |time |float)`

10.5.2.3 Name

Usage: Specifies support for the Name property. MAY be used to specify a default value for the property. Parent Elements: `RTProperties`.

Restrictions: See [RFC2396]. If a default property value is specified, it SHOULD be the same as the value of the `NodeName` element.

Content Model: `(#PCDATA)`

10.5.2.4 Size

Usage: Specifies support for the Size property. MAY be used to specify a default value for the property. Within its parent element, the `Size` element is defined as optional. Size MUST NOT be used for Interior Nodes. In Node elements describing Leaf Nodes, the `Size` element MAY be used within `RTProperties`.

Parent Elements: `RTProperties`.

Restrictions: If a value is specified, it MUST be formatted according to section 8.4.

Content Model: `(#PCDATA)`

10.5.2.5 Title

Usage: Specifies support for the Title property. MAY be used to specify a default value for the property.

Parent Elements: `RTProperties`.

Restrictions: If a value is specified, it MUST be formatted according to section 8.4.

Content Model: `(#PCDATA)`

10.5.2.6 TStamp

Usage: Specifies support for the TStamp property. MAY be used to specify a default value for the property.

Parent Elements: `RTProperties`.

Restrictions: If a value is specified, it MUST be formatted according to section 8.4.

Content Model: (#PCDATA)

10.5.2.7 Type

Usage: Specifies support for the Type property. MAY be used to specify a default value for the property.

Parent Elements: RTProperties.

Restrictions: For Leaf Nodes, if a default value is specified for the described Node, the Type property MUST be used to specify the correct MIME type of the Node's present value using a single MIME element. For Interior Nodes, a DDFName element MUST be present and specify a valid Management Object identifier, which MAY be empty.

Content Model: (MIME | DDFName)

10.5.2.8 VerNo

Usage: Specifies support for the VerNo property. MAY be used to specify a default value for the property.

Parent Elements: RTProperties.

Restrictions: If a value is specified, it MUST be formatted according to section 8.4.

Content Model: (#PCDATA)

10.5.2.9 b64

Usage: OMA DM format description. Specifies that the Node value is Base64 encoded string.

Parent Elements: Format, DFFormat.

Restrictions: None.

Content Model: EMPTY.

10.5.2.10 bin

Usage: OMA DM format description. Specifies that the Node value is binary data. The value MAY be encoded with base64 transport encoding.

Parent Elements: Format, DFFormat.

Restrictions: None.

Content Model: EMPTY.

10.5.2.11 bool

Usage: OMA DM format description. Specifies that the Node value is a Boolean.

Parent Elements: Format, DFFormat.

Restrictions: None.

Content Model: EMPTY.

10.5.2.12 chr

Usage: OMA DM format description. Specifies that the Node value is text.

Parent Elements: Format, DFFormat.

Restrictions: The character set used is specified either by the transport protocol, MIME content type header, or XML prologue.

Content Model: `EMPTY`.

10.5.2.13 int

Usage: OMA DM format description. Specifies that the Node value is a 32-bit signed integer.

Parent Elements: `Format, DFFormat`.

Restrictions: None.

Content Model: `EMPTY`.

10.5.2.14 node

Usage: OMA DM format description. Specifies that the Node is an Interior Node.

Parent Elements: `Format, DFFormat`.

Restrictions: This Format MUST only be used for Interior Nodes.

Content Model: `EMPTY`.

10.5.2.15 null

Usage: OMA DM format description. Specifies that the Node value is null.

Parent Elements: `Format, DFFormat`.

Restrictions: None.

Content Model: `EMPTY`.

10.5.2.16 xml

Usage: OMA DM format description. Specifies that the Node value is XML data.

Parent Elements: `Format, DFFormat`.

Restrictions: None.

Content Model: `EMPTY`.

10.5.2.17 date

Usage: OMA DM format description. Specifies that the Node value is a date in ISO 8601 format with the century being included in the year [ISO8601].

Parent Elements: `Format, DFFormat`.

Restrictions: None.

Content Model: `EMPTY`.

10.5.2.18 time

Usage: OMA DM format description. Specifies that the Node value is a time in ISO 8601 format [ISO8601].

Parent Elements: `Format, DFFormat.`

Restrictions: None.

Content Model: `EMPTY.`

10.5.2.19 float

Usage: OMA DM format description. Specifies that the Node value is a real number corresponding to a single precision 32-bit floating point type as defined in XML Schema 1.0 as the `float` primitive type [XMLSCHEMADT].

Parent Elements: `Format, DFFormat.`

Restrictions: None.

Content Model: `EMPTY.`

10.5.2.20 MIME

Usage: Specifies the MIME type of the current Node value.

Parent Elements: `Type, DFType.`

Restrictions: MUST only contain valid MIME type identifiers. See [META].

Content Model: `(#PCDATA)`

10.5.2.21 DDFName

Usage: Specifies the Management Object Identifier of the Management Object rooted at this Node, or is empty.

Parent Elements: `Type, DFType.`

Restrictions: MUST only contain a valid Management Object Identifier or be empty. See 8.7.7.2.

Content Model: `(#PCDATA)`

10.5.3 Framework Property Elements

The properties that described Nodes in the device description framework are specified with framework property elements. These are not the same as the run-time properties of an instantiated Node in a device. These properties express other information about Nodes that DM Servers might need. The framework properties MUST NOT change at run-time as such a change might introduce discrepancies between the run-time Node and the corresponding description. The following table defines the framework Node properties.

Element/Property	Explanation	Usage
AccessType	Specifies which commands are allowed on the Node.	MUST
DefaultValue	The Node value used in a device unless specifically set to a different value.	MAY
Description	The human readable description of the Node.	MAY
DFFormat	The data format of the described Node.	MUST
Occurrence	Specifies the number of instances that MAY occur of the Node.	MAY
Scope	Specifies whether this is a Permanent or Dynamic Node.	MAY
DFTitle	The human readable name of the Node.	MAY
DFType	For Leaf Nodes, the MIME type of the Node value.	MUST

	For Interior Nodes, a Management Object Identifier or empty.	
CaseSense	Specifies whether the Node name and names of descendant Nodes in the tree below should be treated as case sensitive or case insensitive.	MAY

Table 6: Framework Property Elements

10.5.3.1 AccessType

Usage: Specifies which commands are supported for the described Node. This property is independent of the ACL run-time property.

Parent Elements: `DFProperties.`

Restrictions: The value of this property MUST be an unordered list of the valid OMA DM commands.

Content Model: `(Add?, Copy?, Delete?, Exec?, Get?, Replace?)`

10.5.3.2 DefaultValue

Usage: Specifies the "factory default" value of the Node, if such a value exists.

Parent Elements: `DFProperties.`

Restrictions: The MIME type of the value MUST correspond with the MIME type specified by the DFType property.

Content Model: `(#PCDATA)`

10.5.3.3 Description

Usage: A human readable description of the described Node. This is used to convey any relevant information regarding the Node that is not explicitly given by the other properties.

Parent Elements: `DFProperties.`

Restrictions: Descriptions SHOULD be kept as short as possible.

Content Model: `(#PCDATA)`

10.5.3.4 DFFormat

Usage: Specifies the data format of the described Node.

Parent Elements: `DFProperties.`

Restrictions: For Interior Nodes the Format MUST be `node`.

Content Model: `(b64 | bin | bool | chr | int | node | null | xml | date | time | float)`

10.5.3.5 Occurrence

Usage: Specifies the potential number of instances that MAY occur of the described Node.

Parent Elements: `DFProperties.`

Restrictions: If the property is omitted, the Node occurrence is exactly one.

Content Model: `(One | ZeroOrOne | ZeroOrMore | OneOrMore | ZeroOrN | OneOrN)`

10.5.3.6 Scope

Usage: Specifies whether the described Node is Permanent or Dynamic.

Parent Elements: `DFProperties.`

Restrictions: The value is indicated by the tags `Permanent` and `Dynamic` in the device description framework. This property is OPTIONAL; if omitted, the described Node is `Dynamic`.

Content Model: `(Permanent | Dynamic)`

10.5.3.7 DFTitle

Usage: A human readable name for the Node description.

Parent Elements: `DFProperties.`

Restrictions: Titles SHOULD be kept as short and informative as possible.

Content Model: `(#PCDATA)`

10.5.3.8 DFType

Usage: For Leaf Nodes, one or more elements specify the MIME types the described Node supports. For Interior Nodes, a single MIME element MUST be present and specify a valid Management Object Identifier, which MAY be empty.

Parent Elements: `DFProperties.`

Restrictions: Note that a Leaf Node can support multiple MIME types, e.g., a ring signal Node might support both audio/mpeg and audio/MP4-LATM. The values of the `DFType` property MUST be registered MIME types.

Content Model: `(MIME+ | DDFName)`

10.5.3.9 Add

Usage: Specifies support for the OMA DM Add command.

Parent Elements: `AccessType.`

Restrictions: None.

Content Model: `EMPTY.`

10.5.3.10 Copy

Usage: Specifies support for the OMA DM Copy command.

Parent Elements: `AccessType.`

Restrictions: Copy command is deprecated in DM 1.3.

Content Model: `EMPTY.`

10.5.3.11 Delete

Usage: Specifies support for the OMA DM Delete command.

Parent Elements: `AccessType.`

Restrictions: None.

Content Model: EMPTY.

10.5.3.12 Exec

Usage: Specifies support for the OMA DM Exec command.

Parent Elements: AccessType.

Restrictions: None.

Content Model: EMPTY.

10.5.3.13 Get

Usage: Specifies support for the OMA DM Get command.

Parent Elements: AccessType.

Restrictions: None.

Content Model: EMPTY.

10.5.3.14 Replace

Usage: Specifies support for the OMA DM Replace command.

Parent Elements: AccessType.

Restrictions: None.

Content Model: EMPTY.

10.5.3.15 One

Usage: Specifies that the described Node can occur exactly one (1) time.

Parent Elements: Occurrence.

Restrictions: None.

Content Model: EMPTY.

10.5.3.16 ZeroOrOne

Usage: Specifies that the described Node can occur either one (1) time or not at all.

Parent Elements: Occurrence.

Restrictions: None.

Content Model: EMPTY.

10.5.3.17 ZeroOrMore

Usage: Specifies that the described Node can occur an unspecified number of times, or not occur at all.

Parent Elements: Occurrence.

Restrictions: None.

Content Model: `EMPTY`.

10.5.3.18 ZeroOrN

Usage: Specifies that the described Node can occur any number of times up to N times, or not occur at all.

Parent Elements: `Occurrence`.

Restrictions: N MUST be specified as a character string representing a positive integer value between 2 and 65536.

Content Model: `(#PCDATA)`

10.5.3.19 OneOrMore

Usage: Specifies that the described Node can occur an unspecified number of times, but MUST occur at least once.

Parent Elements: `Occurrence`.

Restrictions: None.

Content Model: `EMPTY`.

10.5.3.20 OneOrN

Usage: Specifies that the described Node can occur any number of times up to N times, but MUST occur at least once.

Parent Elements: `Occurrence`.

Restrictions: N MUST be specified as a character string representing a positive integer value between 2 and 65536.

Content Model: `(#PCDATA)`

10.5.3.21 Dynamic

Usage: Specifies that the described Node is Dynamic.

Parent Elements: `Scope`.

Restrictions: None.

Content Model: `EMPTY`.

10.5.3.22 Permanent

Usage: Specifies that the described Node is Permanent.

Parent Elements: `Scope`.

Restrictions: None.

Content Model: `EMPTY`.

10.5.3.23 CaseSense

Usage: Specifies whether the Node name and names of descendant Nodes in the tree below SHOULD be treated as case sensitive or case insensitive.

Parent Elements: `DFProperties`.

Restrictions: MUST only contain value CS or CIS.

Content Model: `(CS | CIS)`

10.5.3.24 CS

Usage: Case sensitivity declaration. Specifies that child Node names MUST be treated as case sensitive.

Parent Elements: `CaseSense`.

Restrictions: None.

Content Model: `EMPTY`.

10.5.3.25 CIS

Usage: Case insensitivity declaration. Specifies that child Node names MUST be treated as case insensitive.

Parent Elements: `CaseSense`.

Restrictions: None.

Content Model: `EMPTY`.

10.6 Shortcomings of the OMA DM Device Description Framework

It is not possible to specify that only one Node of an allowed set can be instantiated at a time. Compared with the DTD construct (A | B), this is currently described by making both A and B optional, but there is no way to represent that they are mutually exclusive. However, this is mostly a problem that occurs when mapping existing Management Objects onto the device description framework. When new Management Objects are designed for the framework, this situation can be avoided.

11.WBXML Definition

The following tables define the token assignments for the mapping of the DMDDF-related DTDs and element types into WBXML as defined by [WBXML1.1], [WBXML1.2], [WBXML1.3].

11.1 Code Page Definitions

The following code page tokens represent DM DDF public identifiers. This version of the DM protocol specification utilizes the WBXML code page tokens for identifying DTDs.

DTD Name	WBXML Code Page Token (Hex Value)	Formal Public Identifier
DMDDF	02	-//OMA//DTD-DM-DDF 1.2//EN

Table 7: Code Page Tokens

11.2 Token Definitions

The following WBXML token codes represent element types (i.e., tags) from code page x02 (two).

Element Type Name	WBXML Tag Token (Hex Value)
AccessType	05
ACL	06
Add	07
b64	08
bin	09
bool	0A
chr	0B
CaseSense	0C
CIS	0D
Copy	0E
CS	0F
date	10
DDFName	11
DefaultValue	12
Delete	13
Description	14
DFFormat	15

DFProperties	16
DFTitle	17
DFType	18
Dynamic	19
Exec	1A
float	1B
Format	1C
Get	1D
int	1E
Man	1F
MgmtTree	20
MIME	21
Mod	22
Name	23
Node	24
node	25
NodeName	26
null	27
Occurrence	28
One	29
OneOrMore	2A
OneOrN	2B
Path	2C
Permanent	2D
Replace	2E
RTProperties	2F
Scope	30
Size	31
time	32

Title	33
TStamp	34
Type	35
Value	36
VerDTD	37
VerNo	38
xml	39
ZeroOrMore	3A
ZeroOrN	3B
ZeroOrOne	3C

Table 8: Token Definitions

Appendix A. History (Informative)

A.1 Approved Version History

Reference	Date	Description
N/A	N/A	No prior version 1.3

A.2 Draft/Candidate Version 1.3 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-TS-DM_TND-V1_3	15 Oct 2008	All	Baseline to v1.3. using OMA-TS-DM_TND-V1_2_1-20080617-A
	12 Jan 2009	7.7.1.4	Applied OMA-DM-DM13-2008-0001R01-CR_Server_Deletion_ACL_Fix.
	29 Jan 2009	6.2.3	Applied OMA-DM-2008-0088R02-CR_Clarify_Permanent_Nodes.
	08 Sep 2009	All	Applied OMA-DM-DM13-2009-0067-CR_TND_ACL_Bug_Fix, OMA_DM-DM13-2009-0053R01-CR_TND_MORoot_Query OMA-DM-DM13-2009-0042R01-CR_TND_MORoot_Query
	09 Sep 2009	7.7.7.1	Applied OM-DM-DM13-2009-005601-CR_ACL_Enhance_Empty_Right.
	28 Oct 2009	All	Applied OMA-DM-DM13-2009-0055-CR_ACL_Enhance_Replace_RightOMA-DM-DM13-2009-CR_TND_SCR_Entries
	18 Nov 2009	All	Applied OMA-DM-DM13-2009-0079-CR_Revising_the_ACL_Example OMA-DM-DM13-2009-0110R01-CR_TND_Cleanup OMA-DM-DM13-2009-0113R01-CR_TND_MORoot_Clarifications
	14 Jan 2010	All	Applied OMA-DM-DM13-2009-0136-CR_TND_ACL_Bug_Fix Editorial clean-up by DSO
	19 Jan 2010	App C	Inserted title and number to figure 4 Renumbering of the sequence of figures in the document
	03 May 2010	All	Editorial clean-up
	04 May 2010	7.7.1	Applied OMA-DM-DM13-2010-0077R01-CR_Error_Status_Code_for_Get_Empty_ACL_in_TND Language set to English UK in the whole document
	05 May 2010	All	Changed snippets font from Courier to Courier New where relevant to harmonize them in the document
Candidate Version OMA-TS-DM_TND-V1_3	25 May 2010	N/A	Status changed to Candidate by TP Ref # OMA-TP-2010-0221-INP_DM_V1.3_ERP_and_ETR_for_Candidate_approval
Draft Versions OMA-TS-DM_TND-V1_3	25 Aug 2010	7.7.1.3, 7.7.7.2, 10.2	Applied OMA-DM-DM13-2010-0068R01-CR_ACL_Clarification OMA-DM-DM13-2010-0078R01-CR_TND_Minor_Fixes OMA-DM-DM13-2010-0100R03-CR_MOID_Uniqueness
	01 Oct 2010	7.5.1	Applied OMA-DM-DM13-2010-0111-CR_RepPro_vs_TND_mismatch
	04 Nov 2010	8.1	Applied OMA-DM-DM13-2010-0117-CR_TND_Bug_Fixes
Candidate Version OMA-TS-DM_TND-V1_3	07 Dec 2010	N/A	Status changed to Candidate by TP Ref # OMA-TP-2010-0502-INP_DM_V1_3_ERP_and_ETR_for_Candidate_re_approval
Draft Versions OMA-TS-DM_TND-V1_3	17 Feb 2011	6.2, 8.1	Applied OMA-DM-DM13-2011-0008-CR_Get_Response_Bugfix

Document Identifier	Date	Sections	Description
	21 Nov 2011	All	Applied OMA-DM-DM13-2011-0089R03-CR_ACL_value_for_the_root OMA-DM-DM13-2011-0104-CR_TechWriter_Review_Update
Draft Versions OMA-TS-DM_TND-V1_3	16 Jan 2012	All	Applied OMA-DM-DM13-2011-0139R02-CR_CONR_TND
	02 Feb 2012	2, 4, App B	Applied 2012 TS template to SCR tables according to Action Item DM-2012-A008 + added reference to SCRRULES Restored cross references in the whole document, update of references to DM 1.3 versions and change of DMBOOT reference into a normative ref according to Action Item DM-2012-A016 Applied 2012 template to introduction section.
	07 Feb 2012	10.4 (new) 7 (new) App B	Applied OMA-DM-DM13-2012-0033R01-CR_DMTND_Move_DDFNotation OMA-DM-DM13-2012-0054R02-CR_Node_addressing_introduced_in_TND Updated all the cross-reference fields according to sections re-numbering
	16 Feb 2012	10.5.2.9 10.5.2.10 10.5.3.10	Applied OMA-DM-DM13-2012-0059R01-CR_TND_CONRR_DDF_Path
	22 Feb 2012	8.7.1.6 10.3 10.4 7.2	Applied OMA-DM-DM13-2012-0068R02-CR_TND_Mime_Type OMA-DM-DM13-2012-0069R01-CR_TND_Standardized_Object_A027 OMA-DM-DM13-2012-0080R01-CR_TND_Management_Tree OMA-DM-DM13-2012-0084-CR_TND_Virtal_URI OMA-DM-DM13-2012-0091-CR_Reference_for_CONRR_K012
	23 Feb 2012	All	Revision made online and agreed during the Barcelona F2F meeting
	24 Feb 2012	4.1	Header removed by DSO according to Action Item DM-2012-A030
Candidate Version OMA-TS-DM_TND-V1_3	06 Mar 2012	N/A	Status changed to Candidate by TP Ref # OMA-TP-2012-0084-INP_DM_V1_3_ERP_and_ETR_for_Candidate_re_approval

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

B.1 SCR for DM Client

Item	Function	Reference	Requirement
DMTND-Prop-C-001-M	Support for the ACL property	Section 8.2	
DMTND-Prop-C-002-M	Support for the Format property	Section 8.2	
DMTND-Prop-C-003-M	Support for the Name property	Section 8.2	
DMTND-Prop-C-004-O	Support for the Size property in Leaf Nodes	Section 8.2	
DMTND-Prop-C-005-M	No support for the Size property in Interior Nodes	Section 8.2	
DMTND-Prop-C-006-O	Support for the Title property	Section 8.2	
DMTND-Prop-C-007-O	Support for the TStamp property	Section 8.2	
DMTND-Prop-C-008-M	Support for the Type property	Section 8.2	
DMTND-Prop-C-009-O	Support for the VerNo property	Section 8.2	
DMTND-Prop-C-010-O	Support Get?list=Struct	Section 9.1	
DMTND-Prop-C-011-O	Support Get?list=StructData	Section 9.1	
DMTND-Prop-C-012-O	Support Get?list=TNDS	Section 9.1	
DMTND-Prop-C-013-O	Support Get?list=MORoot	Section 9.1	
DMTND-Prop-C-014-O	Support Get?list=MORootData	Section 9.1	
DMTND-Prop-C-015-M	Support Relative URI Addressing Mechanism	Section 7.2	

B.2 SCR for DM Server

Item	Function	Reference	Requirement
DMTND-Prop-S-001-M	Support for the ACL property	Section 8.2	
DMTND-Prop-S-002-M	Support for the Format property	Section 8.2	
DMTND-Prop-S-003-M	Support for the Name property	Section 8.2	
DMTND-Prop-S-004-M	Support for the Size property in Leaf Nodes	Section 8.2	
DMTND-Prop-S-005-M	No support for the Size property in Interior Nodes	Section 8.2	

Item	Function	Reference	Requirement
DMTND-Prop-S-006-M	Support for the Title property	Section 8.2	
DMTND-Prop-S-007-M	Support for the TStamp property	Section 8.2	
DMTND-Prop-S-008-M	Support for the Type property	Section 8.2	
DMTND-Prop-S-009-M	Support for the VerNo property	Section 8.2	
DMTND-Prop-S-010-O	Support Get?list=Struct	Section 9.1	
DMTND-Prop-S-011-O	Support Get?list=StructData	Section 9.1	
DMTND-Prop-S-012-O	Support Get?list=TNDS	Section 9.1	
DMTND-Prop-S-013-O	Support Get?list=MORoot	Section 9.1	
DMTND-Prop-S-014-O	Support Get?list=MORootData	Section 9.1	
DMTND-Prop-S-015-M	Support Relative URI Addressing Mechanism	Section 7.2	

Appendix C. Tree Exchange Examples (Informative)

In the examples below, the DM Client has the following Management Tree.

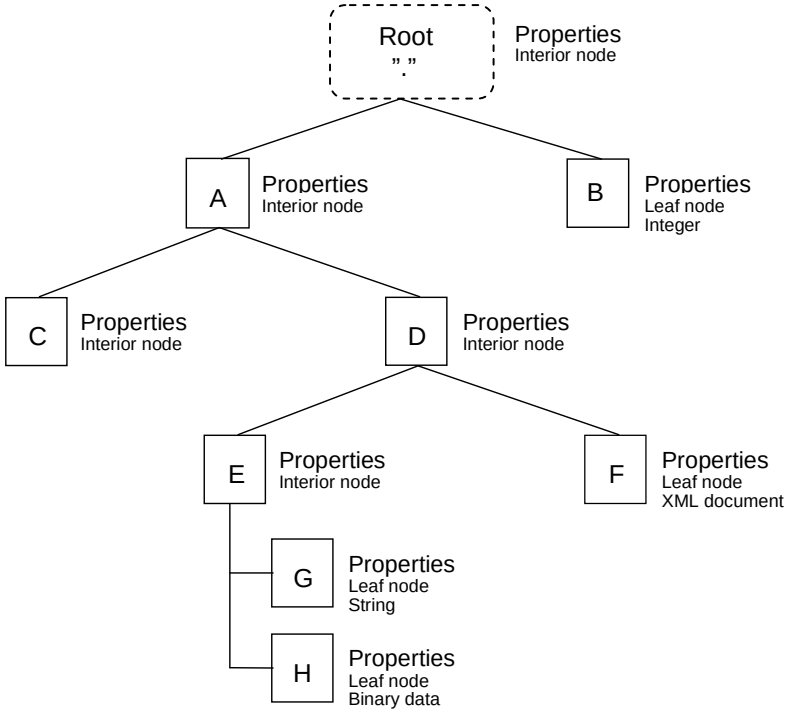

Figure 8: Example Management Tree

C.1 Example of a Get with Attribute Struct

In the example below, the DM Server requests a Management Tree structure from a client.

```
<Get>
  <CmdID>4</CmdID>
  <Item>
    <Target>
      <LocURI>./A?list=Struct</LocURI>
    </Target>
  </Item>
</Get>
```

Client response using `Result` and multiple `Item` elements.

```
<Results>
  <CmdRef>4</CmdRef>
  <CmdID>7</CmdID>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
```

```
      <LocURI>./A</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
      <LocURI>./A/C</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
      <LocURI>./A/D</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
      <LocURI>./A/D/E</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>xml</Format>
    </Meta>
    <Source>
      <LocURI>./A/D/F</LocURI>
    </Source>
  </Item>
  <Item>
    <Source>
      <LocURI>./A/D/E/G</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>b64</Format>
    </Meta>
    <Source>
      <LocURI>./A/D/E/H</LocURI>
    </Source>
  </Item>
</Results>
```

C.2 Example of a Get with Attribute StructData

In the example below, the DM Server requests a Management Tree structure and the data from a client.

```
<Get>
  <CmdID>4</CmdID>
  <Item>
    <Target>
```

```
      <LocURI>./A/D?list=StructData</LocURI>
    </Target>
  </Item>
</Get>
```

Client response using `Result` and multiple `Item` elements.

```
<Results>
  <CmdRef>4</CmdRef>
  <CmdID>7</CmdID>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
      <LocURI>./A/D</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
      <LocURI>./A/D/E</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>xml</Format>
    </Meta>
    <Source>
      <LocURI>./A/D/F</LocURI>
    </Source>
    <Data>'XML document'</Data>
  </Item>
  <Item>
    <Source>
      <LocURI>./A/D/E/G</LocURI>
    </Source>
    <Data>leaf node data</Data>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>b64</Format>
      <Type xmlns='syncml:metinf'>image/jpeg</Type>
    </Meta>
    <Source>
      <LocURI>./A/D/E/H</LocURI>
    </Source>
    <Data>JSCNMDTUVWXYZcuokcdghfidjssatu</Data>
  </Item>
</Results>
```

C.3 Example of a Get with Attribute MORoot

In the examples below, the DM Client has the following Management Tree.

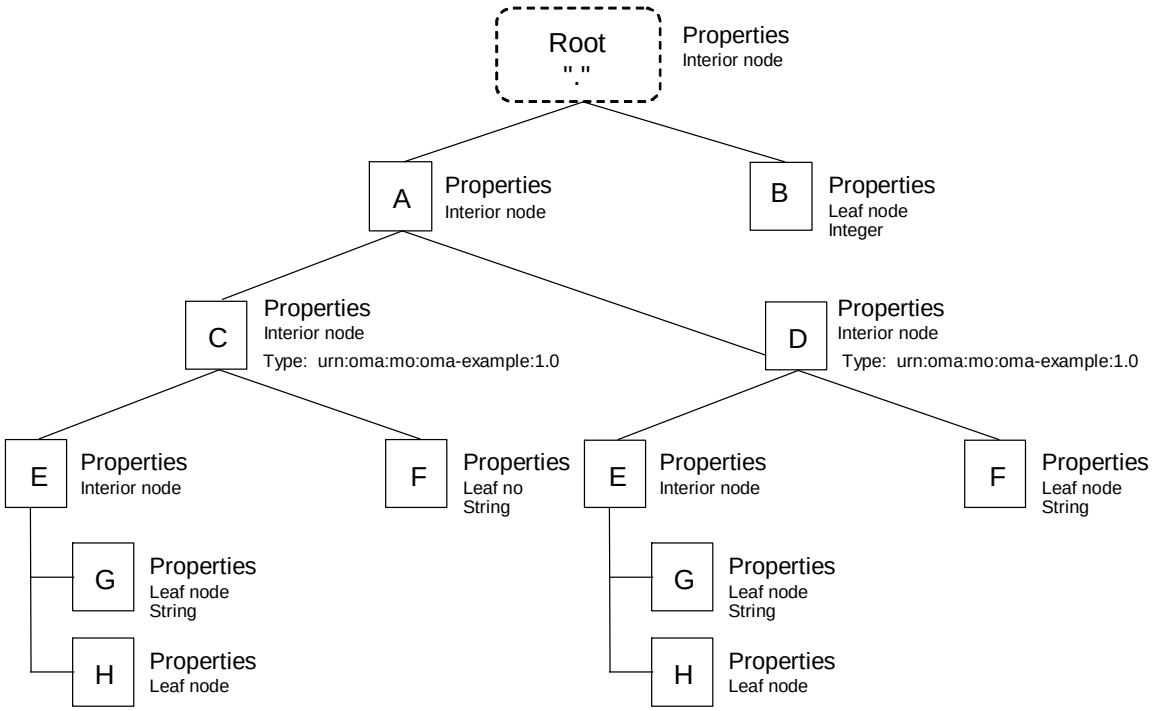

Figure 9: Example of Get with Attribute MORoot

In the example below DM server requests roots of MO occurrences from a client.

```
<Get>
  <CmdID>4</CmdID>
  <Item>
    <Target>
      <LocURI>.?list=MORoot</LocURI>
    </Target>

    <Data>urn:oma:mo:oma-example:1.0</Data>
  </Item>
</Get>
```

Client response using `Result` and multiple `Item` elements.

```
<Results>
  <CmdRef>4</CmdRef>
  <CmdID>7</CmdID>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
      <LocURI>./A/C</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
```

```
    <Source>
      <LocURI>./A/D</LocURI>
    </Source>
  </Item>
</Results>
```

C.4 Example of a Get with Attribute MORootData

In the example below, the DM Server requests roots of MO occurrences of DCMO with specified Leaf Nodes 'Property' data from a client. The assumption is that nodes ./A/C and ./A/D in the diagram in section C.3 are of type 'urn:oma:mo:oma-dcmo:1.0' and nodes ./A/C/F and ./A/D/F are actually the 'Property' Leaf Nodes of DCMO.

```
<Get>
  <CmdID>4</CmdID>
  <Item>
    <Target>
      <LocURI>.?list=MORootData</LocURI>
    </Target>

    <Data>urn:oma:mo:oma-dcmo:1.0?/Property</Data>
  </Item>
</Get>
```

Client response using `Result` and multiple `Item` elements.

```
<Results>
  <CmdRef>4</CmdRef>
  <CmdID>7</CmdID>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
      <LocURI>./A/C</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>chr</Format>
    </Meta>
    <Source>
      <LocURI>./A/C/Property</LocURI>
    </Source>

    <Data>Camera</Data>   <!-- this is camera -->
 </Item>
 <Item>
    <Meta>
      <Format xmlns='syncml:metinf'>node</Format>
    </Meta>
    <Source>
      <LocURI>./A/D</LocURI>
    </Source>
  </Item>
  <Item>
    <Meta>
```

```
      <Format xmlns='syncml:metinf'>chr</Format>
    </Meta>
    <Source>
      <LocURI>./A/D/Property</LocURI>
    </Source>

    <Data>Bluetooth</Data>    <!-- this is Bluetooth -->
 </Item>
</Results>
```

Appendix D. Type Definitions (Normative)

D.1 MIME Media Type Definition

MIME Type	Description
application/vnd.syncml.dmddf+xml	XML encoded DDF document complying to this specification.
application/vnd.syncml.dmddf+wbxml	WBXML encoded DDF document complying to this specification.