SyncML Device Management Tree and Description, Version 1.1.2

Approved version 02-December-2003

Open Mobile Alliance
OMA-SyncML-DMTND-V1_1_2-20031202-A

Continues the Technical Activities
Originated in the SyncML Initiative

Contents

1. Scope

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

The SyncML Initiative, Ltd. was a not-for-profit corporation formed by a group of companies who co-operated to produce an open specification for data synchronization and device management. Prior to SyncML, data synchronization and device management had been based on a set of different, proprietary protocols, each functioning only with a very limited number of devices, systems and data types. These non-interoperable technologies have complicated the tasks of users, manufacturers, service providers, and developers. Further, a proliferation of different, proprietary data synchronization and device management protocols has placed barriers to the extended use of mobile devices, has restricted data access and delivery and limited the mobility of the users.

SyncML Components

SyncML is a specification that contains the following main components:

- An XML-based representation protocol
- A synchronization protocol and a device management protocol
- Transport bindings for the protocol
- A device description framework for device management

2. References

2.1 Normative References

[AMT] Assigned media types. Internet Assigned Numbers Authority. URL:http://www.isi.edu/in-notes/iana/assignments/media-types/

[DMCONF] "Device Management Conformance Requirements, Version 1.1.2". Open Mobile Alliance™. OMA-SyncML-DMConReqs-V1_1_2. URL:http:www.openmobilealliance.org/tech/docs

[DMDDFDTD] "SyncML DM Device Description Framework, Version 1.1.2". Open Mobile Alliance™. OMA-SyncML-DMDDFDTD-V1_1_2. URL:http:www.openmobilealliance.org/tech/DTD

[DMPRO] "SyncML Device Management Protocol, Version 1.1.2". Open Mobile Alliance™. OMA-SyncML-DMProtocol-V1_1_2. URL:http:www.openmobilealliance.org/tech/docs

[DMSEC] "SyncML Device Management Security, Version 1.1.2". Open Mobile Alliance™. OMA-SyncML-DMSecurity-V1_1_2. URL:http:www.openmobilealliance.org/tech/docs

[DMSTDOBJ] "SyncML Device Management Standardized Objects, Version 1.1.2". Open Mobile Alliance™. OMA-SyncML-DMStdObj-V1_1_2. URL:http:www.openmobilealliance.org/tech/docs

[META] "SyncML Meta Information, version 1.1.2". Open Mobile Alliance™. OMA-SyncML-MetaInfo-V1_1_2. URL:http:www.openmobilealliance.org/tech/docs

[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

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 are specified in this `typeface`.

3.2 Definitions

Access Control List

A list of identifiers and access rights associated with each identifier.

Description Framework

A specification for how to describe the management syntax and semantics for a particular device type.

Dynamic node

A node is dynamic if the DDF property Scope is set to Dynamic, or if the Scope property is unspecified.

Interior node

A node that may have child nodes, but cannot store any value. The Format property of an interior node is `node`.

Leaf node

A node that can store a value, but cannot have child nodes. The Format property of a leaf node is not `node`.

Management object

A management object is a subtree of the management tree which is intended to be a (possibly singleton) collection of nodes which are related in some way. For example, the ./DevInfo nodes form a management object. A simple management object may consist of one single node.

Management client

A software component in a managed device that correctly interprets SyncML DM commands, executes appropriate actions in the device and sends back relevant responses to the issuing management server.

Management server

A network based entity that issues SyncML DM commands to devices and correctly interprets responses sent from the devices.

Management tree

The mechanism by which the management client interacts with the device, e.g. by storing and retrieving values from it and by manipulating the properties of it, for example the access control lists.

Node

A node is a single element in a management tree. There can be two kinds of nodes in a management tree: interior nodes and leaf nodes. The Format property of a node provides information about whether a node is a leaf or an interior node.

Permanent node

A node is permanent if the DDF property Scope is set to Permanent. If a node is not permanent, it is dynamic. A permanent node can never be deleted.

Server identifier

The SyncML DM internal name for a management server. A management server is associated with an existing server identifier in a device through SyncML DM authentication

3.3 Abbreviations

ACL	Access Control List.
DDF	Device Description Framework

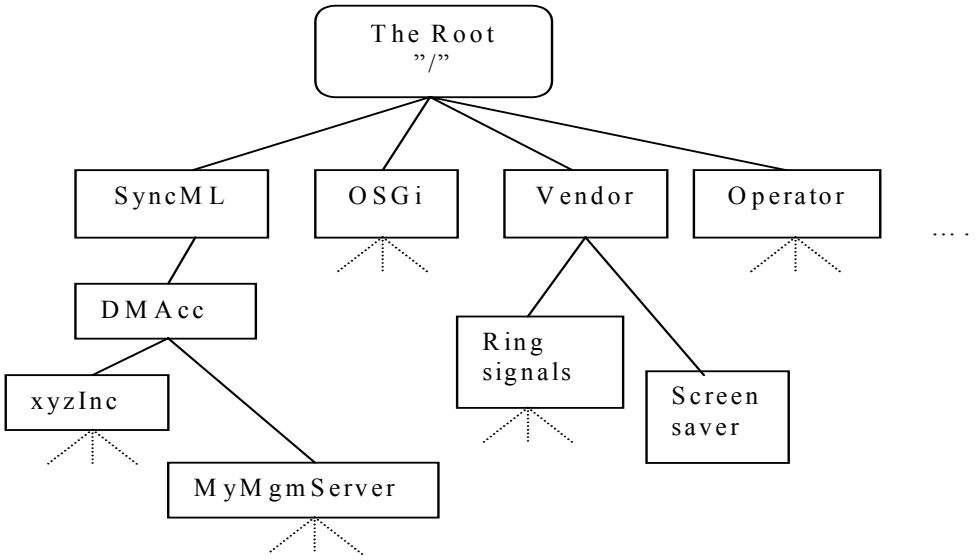
4. Introduction

Other SyncML DM specifications define the syntax and semantics of the SyncML 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. To avoid this situation this specification defines a number of mandatory management objects for various uses in devices. These objects are primarily associated with SyncML DM and SyncML configuration.

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 must have in order 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 device management 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 SyncML 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.



Example management tree

The URI for a node is constructed by starting at the device root and, as the tree is traversed down to the node in question, each node name is appended to the previous ones using "/" as the delimiting character. In the SyncML DM protocol [DMPRO], the URI used to address nodes MUST be a relative URI.

To access the xyzInc node in the SyncML branch in the management tree above, a server would present the address: `./SyncML/DMAcc/xyzInc`, or `SyncML/DMAcc/xyzInc`. Note that the URI MUST be given with the root of the management tree as the starting point.

URI used in SyncML DM MUST be treated and interpreted as case sensitive.

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

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

- 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.

6. Nodes

6.1 Common characteristics of nodes

Nodes are the entities which can be manipulated by management actions carried over the SyncML DM protocol. A node can be as small as an integer or a large and complex like a background picture or screen saver. The SyncML 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 6.1. 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. Management 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 is returned. If the interior node have 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 management 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. However, the parent of any new node MUST be an existing interior node. The device itself can also extend the management tree. This could happen as a result of user input or by attaching the 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 client MUST respond with an appropriate status code. If the command results in a value that is to be sent back to the server, then this value MUST be part of the response. The type of the value 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 cannot be part of any node names according to [AMT].

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 and cannot be deleted. Permanent nodes can also be temporarily added to a device by, for instance, connecting new accessory hardware. However, a management server cannot create or delete permanent nodes at runtime. An attempt by a management server to delete a permanent node will always 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. Note that permanent nodes MUST NOT be children of dynamic nodes.

Dynamic nodes can be created and deleted at runtime by management 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 are also deleted.

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 will 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 or a value that is automatically updated by the client are also assigned 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 may 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, and if that node is a dynamic interior node it will be deleted along with all 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

Note that this mechanism is completely independent of the ACL.

See chapter 9 for more information on DDF properties.

7. Properties of nodes

7.1 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 9.4.3 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 the node's value
VerNo	Version number, automatically incremented at each modification

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

7.2 Supported properties

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

Property	Device support
ACL	MUST
Format	MUST
Name	MUST
Size	MUST for leaf nodes MUST NOT for interior nodes
Title	MAY
TStamp	MAY
Type	MUST
VerNo	MAY

7.3 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 a SyncML DM account a server could use one of these URI.

```
./SyncML/DMAccounts/xyzInc?prop=ACL
SyncML/DMAccounts/xyzInc?prop=ACL
```

If a server addresses an unsupported property in a device, an error is returned in the form of an `(406) Optional feature not supported` status

7.4 Property values

Property values MUST be transported by SyncML 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>
```

7.5 Operations on properties

The following table defines the allowed operations for each property. An operation on a property is equivalent to a SyncML 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 device.
Title	Get, Replace	Only updated by 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.

Properties do not support the Add command. All mandatory properties, and those optional properties that a device implements, are 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 devices may allow the user to modify the ACL. If this occurs and the device supports the TStamp or VerNo properties, these MUST be updated.

7.5.1 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 property, which MUST NOT be changed for permanent nodes. Any attempt to perform such a change will fail.

7.6 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 is no inheritance of property values, implied or specified, other than for the ACL property.

7.7 Detailed description of properties

7.7.1 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 server. The server identifier is a SyncML DM specific name for a server. A management session is associated with a server identifier through SyncML DM authentication [DMSEC]. All management commands received in one session are assumed to originate from the same server.

7.7.1.1 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 server performs a management operation on a node with no value set on the ACL the device MUST look at the ACL of the parent node for a value. If the parent does not have a value for the ACL, the device MUST look at the ACL of the parents 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.

A Get command on the ACL property of a node MUST return an empty data value, e.g. `<Data/>`, if the ACL for that node has no commands present (i.e. if the ACL has no value). In other words, the client SHOULD NOT locate an ancestor with a value for the ACL and provide that value.

7.7.1.2 The root ACL value

The value of the root is special in several ways. The server identifier, or identifiers, with Replace rights according to the root ACL can take control over all ACL in the device. It is also the only node that MUST have a value assigned to the ACL. The default value for the root ACL SHOULD be `Add=*&Get=*&Replace=*`.

To ensure that any authenticated server always can extend the management tree, the root ACL value for the Add command SHOULD NOT be changed. The ACL value for the Add command in the root ACL SHOULD be "`*`". Any attempt by a server to modify this ACL value MAY fail with the status code `(405) Forbidden`.

7.7.1.3 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 can access the node value, list of children or other node properties. Note that even if a 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 server. 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 that controls the root node some devices MAY implement a UI for this purpose.

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 signaled 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 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 server is adding an interior node and does not have Replace access rights on the parent of the new node then the device MUST automatically set the ACL of the new node so that the creating server has Add, Delete and Replace 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 itself 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.

7.7.1.4 Management Server Deletion

When a management server identifier is to be deleted from the device, the management tree MUST be scanned for nodes with ACL's held by the soon to be deleted server identifier. The reference 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 7.7.1.1).

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 in order to comply with the restriction on the root ACL specified in Section 7.7.1.2 in this document.

7.7.1.5 ACL syntax

The ACL structure is a list of server identifiers where each identifier is associated with a list of SyncML 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.

ACL are carried over SyncML 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 contain the domain name of the server. For efficiency reasons it is also RECOMMENDED that it is kept as short as possible. The wildcard 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.

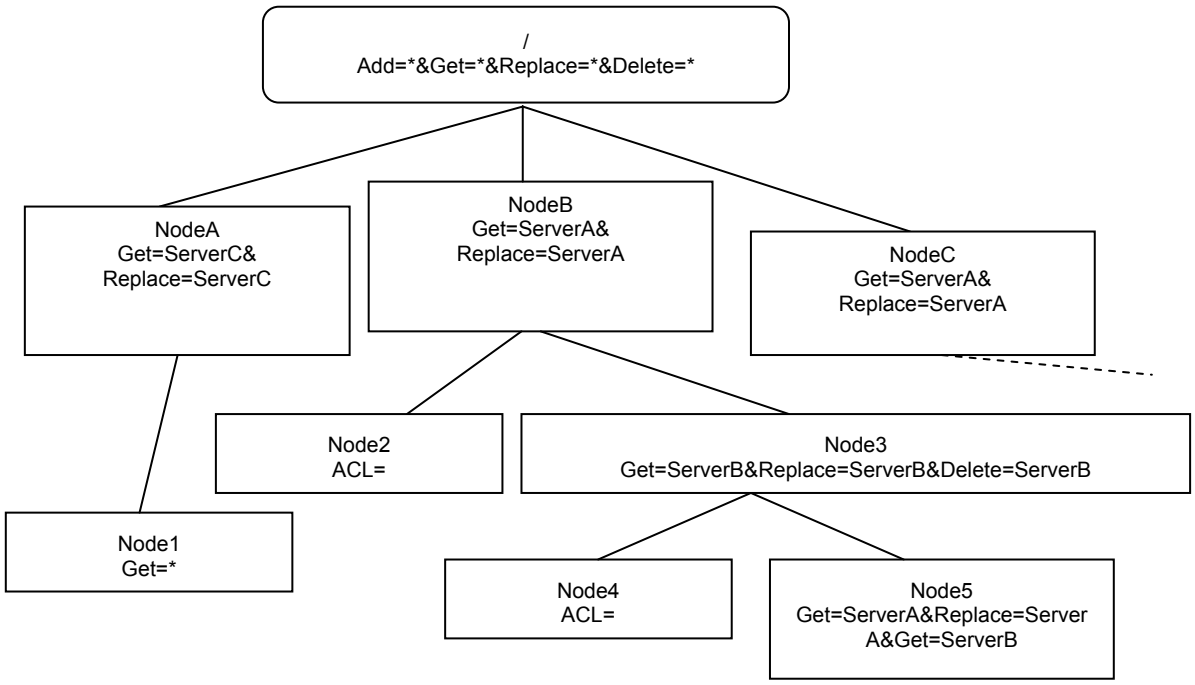
Example ACL value:

```
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, a server needs to have the correct access rights for the equivalent Add, Delete, Get, and Replace commands.

7.7.1.6 ACL Example

Consider the following management tree:



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 no value, e.g. <Data/>.
- A Replace request on `./NodeB/Node3/Node5` by ServerA will be successful.

7.7.2 Format

The Format property always maintains information about the data format of the current node value. Allowed formats are defined in [DMSTDOBJ]. The entity setting the value MUST supply the format information in the same command that is used to set the value. The format information is carried by the Format tag in the meta information for the Item that has the data to be set. The property value is represented by a string. See section 7.7.7.1 for an example.

Note that interior nodes MUST have `node` as the Format value.

Also note that the value `b64` is not used in the Format property of any node in the management tree. The value `b64` is only used in the Meta Format tag, and only in the following situation. 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 unencodes the data and associates the 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 server, and is processed as binary on the 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 and the Meta Format MUST be `b64`, if the Base64 encoding is used. In either case, the Base64 encoding is used only in transport. The management tree property Format for this data MUST be `bin`.

7.7.3 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 last segment in the Target URI.

This property supports the Replace command. When a Replace command for this property is received by the device, 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.

7.7.4 Size

The Size property stores the current size of the node value. The property value is a 32 bit unsigned integer.

When a leaf node is assigned a value with an Add, Copy or Replace command, it is the responsibility of the client to update the Size property of the node. The value of the Size property MUST be equal to the size of the new node value in bytes.

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.

7.7.5 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.

7.7.6 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, ISO 8601 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.

7.7.7 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 describes the collection rooted at that interior object.

7.7.7.1 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 the 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>
```

7.7.7.2 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 name of a Device Description Framework document that describes the collection of objects rooted at the current object. The name is formed by combining the reversed domain name of the owner with the name and version of the object, e.g. "org.syncml/1.1/DevDetail". The syntax for the management object name is as follows:

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

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

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

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

<name> ::= alphanum
```

Notes:

1. The domain is as used in some programming languages to name classes uniquely, and may include sub-domains, e.g. "org.syncml.devman" (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 (case sensitive) alphanumeric characters. (Alphanum is defined in [RFC2396].)

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

Note that a client may store DDF files internally. A 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 SyncML 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 which describes the SyncML DM Account information SHOULD correspond to the description in the DDF named in the Type property of the ./SyncML/DMAcc object. In the case of a conflict, the more local management object name MUST be used.

The server provides the management object name 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 name is intended to be useful under the following circumstances. A management server that examines the management tree in a device may be able to use the management object name to obtain a description of an object that was previously unknown to this server. This situation could arise many ways, for example:

- A different management server in the same organization may have created the collection of objects in the device.
- A management server from a different organization may have created the collection of objects in the device.
- The collection of objects may 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).

7.7.8 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 is incremented. If the property value has reached $FFFF_{16}$, and then is incremented, it returns to 0000_{16}.

8. Device Management Tree Exchange

The Management Tree is a hierarchical arrangement of managed objects in a device, which defines the management view of the device. In order to effectively manage a device, the management 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 a SyncML DM command to the recipient.

Management tree information can be represented and delivered by using an XML representation as described in section 8.2.

The SyncML 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.

8.1 Requesting a Part of a Management Tree

The DM server uses the `Get` command with an attribute in the URI to retrieve the management tree information identified by the attribute in the URI. The client SHOULD send all the requested information and the information of the child nodes to which the DM server has a read access right as specified in the table below. In case this feature is not supported the client SHOULD return status code `(406) Optional Feature Not Supported`.

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 following format:

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

Where the attribute is addressed by appending *`?list=<attribute>`* to the retrieved node's URI.

The possible attributes 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.

8.2 Representation Response of the Management Tree

Representing the tree using multiple `Item` elements inside the SyncML 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 the Appendix A):

- *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.

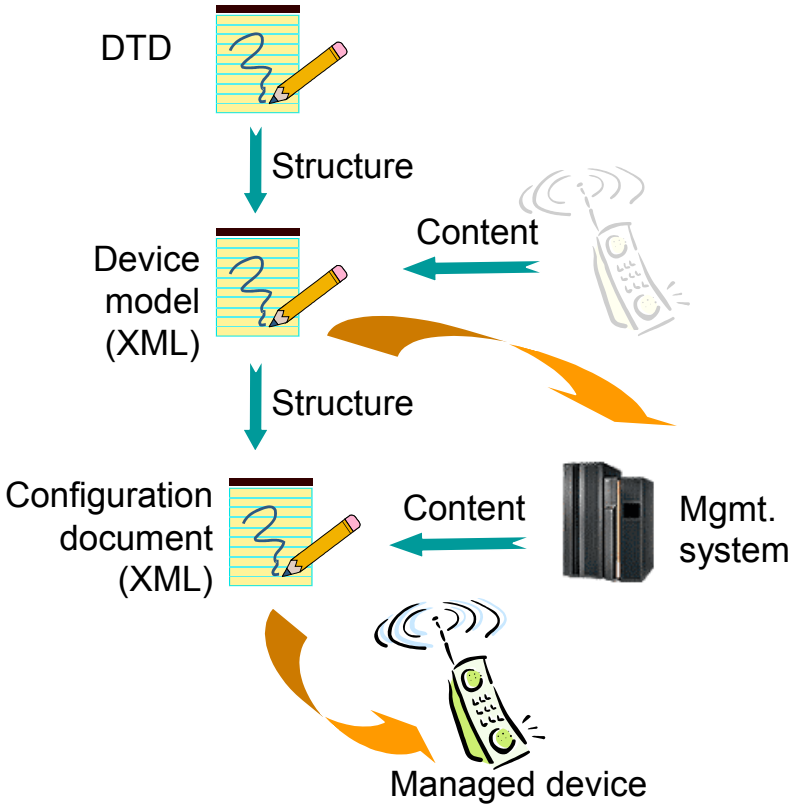
This information from the requested nodes can be included into a multiple `Item` elements inside a `Result` command or there can be multiple `Result` commands with single `Item`.

9. Device Description Framework

9.1 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 the principle.



Conceptual view of how a description framework is used

By using a description framework we also make sure that the total management system based on SyncML DM is flexible and 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 where 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 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 SyncML 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.

9.2 The SyncML DM Device Description Framework

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

However, the need for a description framework remains and therefore SyncML DM RECOMMEND using the following simple framework as an interim solution. This proposed description framework is defined by an XML DTD. Descriptions of management objects, or complete management trees, are valid XML documents. Device manufactures using the description framework MUST make the device descriptions available to management servers. The mechanism for this is currently not being standardized.

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

9.3 The Description Framework DTD (Informative)

For readability the DDF DTD is included here.

```
<!--
SyncML DM Device Description Framework (SYNCML-DMDDF) V1.1.2 Document Type
Definition, modified 02 December 2003.

Copyright Open Mobile Alliance Ltd., 2003-2004
          All rights reserved

This DTD defines the device description framework that is used within
the SyncML Device Management Protocol. Typical usage:
   <!DOCTYPE MgmtTree PUBLIC "-//OMA//DTD SYNCML-DMDDF 1.1.2//EN"
             "http://www.openmobilealliance.org/tech/DTD/OMA-SyncML-DMDDF-
1_1_2.dtd"
             [<?oma-syncml-dmddf-ver supported-versions="1.1.2"?>]>
   <MgmtTree>
      ...
   </MgmtTree>

Terms and conditions of use are available from the
Open Mobile Alliance Ltd. web site at
http://www.openmobilealliance.org/useterms.html
-->
<!-- Root element -->
<!ELEMENT MgmtTree (VerDTD, Man?, Mod?, Node+)>
<!-- For this version of the DTD, the value is "1.1.2" -->
<!ELEMENT VerDTD (#PCDATA)>
<!ELEMENT Man (#PCDATA)>
<!ELEMENT Mod (#PCDATA)>
<!-- The node element is recursive, a Node with a Value tag MUST always
terminate the recursion. It is possible for a Node to omit both the next
recursive Node and a Value, this means that the hierarchy of Nodes continues
elsewhere. This can be used to increase readability of very deep trees.-->
<!ELEMENT Node (NodeName, Path?, RTProperties?, DFProperties, (Node* | Value?))>
<!--NodeName MUST be present but may be empty. If empty this means that the name
is set when the node is created-->
<!ELEMENT NodeName (#PCDATA)>
<!--Path may be omitted. If omitted it means that the actual Path MUST be
constructed from the Path and NodeName values of all ancestral nodes.-->
<!ELEMENT Path (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

```
<!--RTProperties=Run Time Properties. Properties that exists at run-time in a
device. Each node may have a different set of RTProperties. A node that only
supports the mandatory properties and does not need any default values for any
property and may omit the RTProperties-->
<!ELEMENT RTProperties (ACL, Format, Name, Size?, Title?, TStamp?, Type,
VerNo?)>
<!--It is possible to indicate that a property has a default value by inserting
the value as PCDATA. This does not apply to the Format property, which MUST use
one of the enumerated values. The presence of a property tag, with or without
value, indicates that this property is supported. -->
<!ELEMENT ACL (#PCDATA)>
<!ELEMENT Format (b64 | bin | bool | chr | int | node | null | xml)>
<!ELEMENT b64 EMPTY>
<!ELEMENT bin EMPTY>
<!ELEMENT bool EMPTY>
<!ELEMENT chr EMPTY>
<!ELEMENT int EMPTY>
<!ELEMENT node EMPTY>
<!ELEMENT null EMPTY>
<!ELEMENT xml EMPTY>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Size (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT TStamp (#PCDATA)>
<!--For leaf nodes The Type element contains the MIME type of the node. When the
Type element is used in the DFProperties element, it may contain a list of
several MIME types that the node can support at runtime. At run-time the Type
property can only have one value at a time. For interior nodes the Type element
MAY contain the name of a DDF document describing the object rooted at this
location. If it does not contain a DDF document name the value MUST be null.-->
<!ELEMENT Type (MIME | DDFName)>
<!ELEMENT MIME (#PCDATA)>
<!ELEMENT DDFName (#PCDATA)>
<!ELEMENT VerNo (#PCDATA)>
<!--DFProperties=Description Framework Properties. Properties that the node has
only in the description framework. These are not explicit at run-time in a
device.-->
<!ELEMENT DFProperties (AccessType, DefaultValue?, Description?, DFFormat,
Occurrence?, Scope?, DFTitle?, DFType)>
<!ELEMENT AccessType (Add?, Copy?, Delete?, Exec?, Get?, Replace?)>
<!ELEMENT Add EMPTY>
<!ELEMENT Copy EMPTY>
<!ELEMENT Delete EMPTY>
<!ELEMENT Exec EMPTY>
<!ELEMENT Get EMPTY>
<!ELEMENT Replace EMPTY>
<!ELEMENT DefaultValue (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!--DFFormat uses the same child elements as Format-->
<!ELEMENT DFFormat (b64 | bin | bool | chr | int | node | null | xml)>
<!--Occurrence indicates how many instances of a node that can be created. Note
that each node instance MUST have its own unique URI.-->
<!ELEMENT Occurrence (One | ZeroOrOne | ZeroOrMore | OneOrMore | ZeroOrN |
OneOrN)>
<!ELEMENT One EMPTY>
<!ELEMENT ZeroOrOne EMPTY>
<!ELEMENT ZeroOrMore EMPTY>
<!ELEMENT OneOrMore EMPTY>
```

```
<!--The two ...OrN tags are used when an definite upper limit of node instances
needs to be specified. Note that N > 1.-->
<!ELEMENT ZeroOrN (#PCDATA)>
<!ELEMENT OneOrN (#PCDATA)>
<!ELEMENT Scope (Permanent | Dynamic)>
<!ELEMENT Permanent EMPTY>
<!ELEMENT Dynamic EMPTY>
<!ELEMENT DFTitle (#PCDATA)>
<!ELEMENT DFType (MIME+ | DDFName)>
```

9.4 Framework Elements

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

9.4.1 Structural elements

These elements provide various kinds of structural information of the described management object.

9.4.1.1 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+)`

9.4.1.2 VerDTD

Usage: Specifies the major and minor version identifier of the SyncML DM Description Framework specification used to represent the SyncML 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 SyncML representation protocol specification the value MUST be `1.1.2`. The element type MUST be included in the `MgmtTree`.

Content Model: `(#PCDATA)`

9.4.1.3 Man

Usage: Specifies the manufacturer of the device.

Parent Elements: `MgmtTree`

Restrictions: This element is OPTIONAL.

Content Model: `(#PCDATA)`

9.4.1.4 Mod

Usage: Specifies the model number of the device.

Parent Elements: `MgmtTree`

Restrictions: This element is OPTIONAL.

Content Model: `(#PCDATA)`

9.4.1.5 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, this 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 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>
        <DDFProperties>…</DFProperties>
        <Node>
          <NodeName>GWName</NodeName>
          <DFProperties>…</DFProperties>
          <Value>gw.yyy.se</Value>
        </Node>
      </Node>
    </Node>
  </Node>
</MgmtTree>
```

9.4.1.6 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 7.7.3.

Content Model: `(#PCDATA)`

9.4.1.7 Path

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

Parent Elements: `Node`

Restrictions: OPTIONAL element. If omitted, the URI for the node MUST be constructed by concatenating all ancestral NodeName and Path values. This concatenated value MUST form the correct URI. Path SHOULD only be used inside Node elements that are child elements to the MgmtTree element. For general restrictions, see [RFC2396].

Content Model: `(#PCDATA)`

Example: The following XML is an alternative way to describe the same management objects as in the example in section 9.4.1.5. 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>
```

9.4.1.8 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 are undefined.

Content Model: `(#PCDATA)`

9.4.1.9 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, Size 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?)`

9.4.1.10 DFProperties

Usage: Aggregating element for description framework properties, i.e. properties that nodes have in the 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)`

9.4.2 Run-time property elements

The usage of the run-time properties in the 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.

9.4.2.1 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 7.7.1.5.

Content Model: `(#PCDATA)`

9.4.2.2 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)`

9.4.2.3 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)`

9.4.2.4 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. This is done only to facilitate the omission of the element

from nodes describing interior nodes. In node elements describing leaf nodes the `Size` element MUST be used within `RTProperties`.

Parent Elements: `RTProperties`

Restrictions: If a value is specified it MUST be formatted according to section 7.4.

Content Model: `(#PCDATA)`

9.4.2.5 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 7.4.

Content Model: `(#PCDATA)`

9.4.2.6 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 7.4.

Content Model: `(#PCDATA)`

9.4.2.7 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 value. For interior nodes the value MUST specify a valid name of a DDF document, or be empty.

Content Model: `(MIME)`

9.4.2.8 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 7.4.

Content Model: `(#PCDATA)`

9.4.2.9 b64

Usage: SyncML format description. Specifies that the node value is Base64 encoded.

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

9.4.2.10 bin

Usage: SyncML format description. Specifies that the node value is binary data.

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

9.4.2.11 bool

Usage: SyncML DM format description. Specifies that the node value is a Boolean.

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

9.4.2.12 chr

Usage: SyncML 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`

9.4.2.13 int

Usage: SyncML format description. Specifies that the node value is an integer.

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

9.4.2.14 node

Usage: SyncML 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`

9.4.2.15 null

Usage: SyncML format description. Specifies that the node value is null.

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

9.4.2.16 xml

Usage: SyncML format description. Specifies that the node value is XML data.

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

9.4.2.17 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)`

9.4.2.18 DDFName

Usage: Specifies the DDF document name of the management object rooted at this node, or is empty.

Parent Elements: `Type, DFType`

Restrictions: MUST only contain a valid DDF document name or be empty. See 7.7.7.2.

Content Model: `(#PCDATA)`

9.4.3 Framework property elements

The properties that described nodes have in the 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 management servers may need. The framework properties MUST NOT change at run-time since such a change may 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. For interior nodes, a DDF document name or	MUST

	empty.	

9.4.3.1 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 SyncML DM commands.

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

9.4.3.2 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)`

9.4.3.3 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)`

9.4.3.4 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)`

9.4.3.5 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)`

9.4.3.6 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 description framework. This property is OPTIONAL, if omitted the described node is `Dynamic`.

Content Model: `(Permanent | Dynamic)`

9.4.3.7 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)`

9.4.3.8 DFType

Usage: For leaf nodes, specifies the MIME types of that the described node supports. For interior nodes the value MUST specify a valid name of a DDF document, or 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)`

9.4.3.9 Add

Usage: Specifies support for the SyncML DM Add command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

9.4.3.10 Copy

Usage: Specifies support for the SyncML DM Copy command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

9.4.3.11 Delete

Usage: Specifies support for the SyncML DM Delete command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

9.4.3.12 Exec

Usage: Specifies support for the SyncML DM Exec command.

Parent Elements: AccessType

Restrictions: None.

Content Model: EMPTY

9.4.3.13 Get

Usage: Specifies support for the SyncML DM Get command.

Parent Elements: AccessType

Restrictions: None.

Content Model: EMPTY

9.4.3.14 Replace

Usage: Specifies support for the SyncML DM Replace command.

Parent Elements: AccessType

Restrictions: None.

Content Model: EMPTY

9.4.3.15 One

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

Parent Elements: Occurrence

Restrictions: None.

Content Model: EMPTY

9.4.3.16 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

9.4.3.17 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

9.4.3.18 ZeroOrN

Usage: Specifies that the described node can occur any number 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)`

9.4.3.19 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`

9.4.3.20 OneOrN

Usage: Specifies that the described node can occur any number 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)`

9.4.3.21 Dynamic

Usage: Specifies that the described node is dynamic.

Parent Elements: `Scope`

Restrictions: None.

Content Model: `EMPTY`

9.4.3.22 Permanent

Usage: Specifies that the described node is permanent.

Parent Elements: `Scope`

Restrictions: None.

Content Model: `EMPTY`

9.5 Shortcomings of the SyncML DM description framework

It is not possible to specify that only one node of an allowed set can be instantiated at a time. Compare 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 description framework. When new management objects are designed for the framework, this situation can be avoided.

Appendix A. Static Conformance Requirements (Normative)

The static conformance requirements can be found in [DMCONF].

Appendix B. Change History (Informative)

B.1 Approved Version History

Reference	Date	Description
n/a	n/a	No previous version within OMA

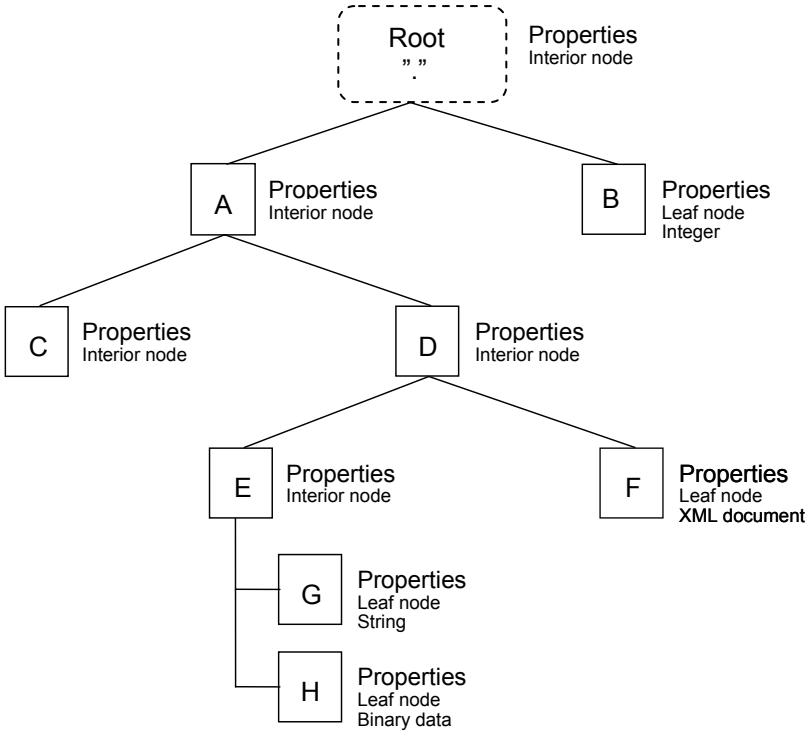
B.2 Draft/Candidate Version 1.1.2 History

<<This section is available in pre-approved versions – it should be removed in the actual approved versions>>

Document Indetifier	Date	Section	Description
Class 0	10-Apr-2003	All	The initial version of this document, based on SyncML DM 1.1.1.
Class 2	10-Apr-2003	All	Change Requests listed in OMA-DM-2003-0047R3
Class 3	08-May-2003	All	Editorial corrections
Draft Version OMA-SyncML-DMTND-V1_1_2-20030508-D	8 May 2003		Draft for TP approval
Candidate Version OMA-SyncML-DMTND-V1_1_2-20030612-C	12 June 2003		Status Changed to Candidate by TP TP ref# OMA-TP-2003-0266R1
Class 3	02 December 2003	2.1 and 9.3	Change request OMA-DM-2003-0168R01-CR_DM-DDF-Update

Appendix C. Tree Exchange Examples (Informative)

In the examples below the DM client has a following management tree.



A management tree in a device

C.1 Example of a Get with Attribute Struct

In the example below 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 DM server requests a management tree structure as well as 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>
```