SyncML Device Management Tree and Description

Abstract

This document defines the management objects on which the SyncML DM protocol acts. A standardised way to describe these objects is also specified.

SyncML Initiative

The following companies are Sponsors of the SyncML Initiative:

Ericsson
IBM
Lotus
Matsushita Communication Industrial Co, Ltd.
Motorola
Nokia
Openwave Systems, Inc.
Starfish Software
Symbian

Revision History

Revision	**Date**	**Comments**
0.8	2001-11-21	Alpha release
1.1	2002-02-15	First release

Copyright Notice

Table of Contents

1 Introduction 6
2 Formatting Conventions 6
3 Terminology 6
4 The Management Tree 7
5 Management Objects 8
5.1 Common characteristics of management objects 8
5.2 Details of the management tree 8
5.2.1 Addressing object values 9
5.2.2 Addressing object child lists 9
5.2.3 Permanent and dynamic objects 10
5.2.4 Adding dynamic objects 10
6 Properties of management objects 11
6.1 Definition 11
6.2 Supported properties 12
6.3 Property addressing 13
6.4 Property values 13
6.5 Operations on properties 13
6.5.1 Properties of permanent objects 14
6.6 Scope of properties 15
6.7 Detailed description of properties 15
6.7.1 ACL 15
6.7.2 Format 18
6.7.3 Name 19
6.7.4 Size 19
6.7.5 Title 19
6.7.6 TStamp 19
6.7.7 Type 19
6.7.8 VerNo 20
7 Device Description Framework 21
7.1 Rationale for a Device Description Framework 21
7.2 The SyncML DM Device Description Framework 22
7.3 The Description Framework DTD 22
7.4 Framework Elements 25
7.4.1 Structural elements 25
7.4.2 Run-time property elements 28
7.4.3 Framework property elements 32

7.5 Shortcomings of the SyncML DM description framework.............. 37
8 References 38

1 Introduction

Other SyncML DM specifications define the syntax and semantic 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 addressing schemes. To avoid that situation this specification defines the addressing scheme and data structure used in SyncML DM.

Since device manufacturers always will develop new functions in their devices and since these functions often are proprietary, no standardised management objects exist for them. To still make these functions manageable in the devices that has them a device description framework is needed that can provide the 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. Organisations operating device management servers should then only have to feed the new description to their servers for them to automatically recognise and manage the new functions in the devices.

2 Formatting Conventions

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

3 Terminology

This section defines terminology used throughout the specification.

ACL

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.

Management object

A manageable entity in a managed device. A management object can have other management objects linked to it as children and a collection of management objects can thus form a tree structure. Management objects can be dynamic or permanent, permanent objects cannot be deleted but their value can be changed.

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

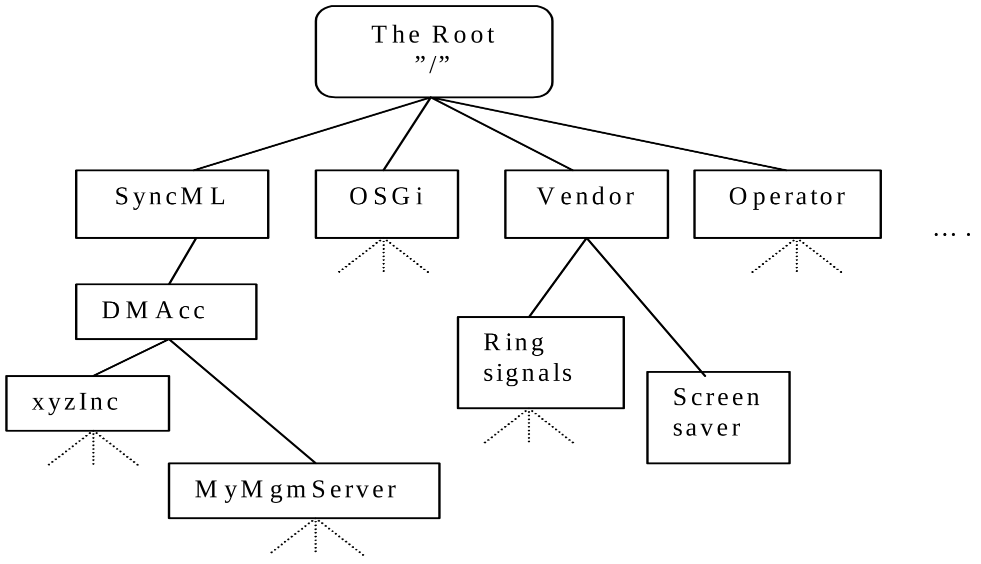
The mechanism by which the management client interacts with the device, e.g. by storing and retrieving values from it and by manipulating the properties of it, for example the access control lists.

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.

4 The Management Tree

Each device that supports SyncML DM MUST contain a management tree. The management tree organises all available management objects in the device as a hierarchical tree structure where all management objects can be uniquely addressed with a URI. The following figure shows an example management tree.



Example management tree

The URI for a management object is constructed by starting at the device root and, as the tree is traversed down to the management object in question, each object name is appended to the previous ones using "/" as the delimiting character. In the SyncML DM protocol [3], the URI used to address management objects MUST be a relative URI.

To access the xyzInc object 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 UFT-8 character set.

5 Management Objects

5.1 Common characteristics of management objects

Management objects are the entities which can be manipulated by management actions carried over the SyncML DM protocol. A management object 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 management objects and treats the object values as opaque data.

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

Each management object have properties associated with it; these properties are defined in 6.1. All properties, except the ACL, are valid only for the object to which they are associated.

5.2 Details of the management tree

As mentioned before the complete structure of all management objects 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 object has child objects linked to it the name of these child objects are returned as a result of the GET. If there are no children the object 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.

Management objects that have one or more child objects are called interior objects. Management objects without any children are called leaf objects. Leaf objects have manageable values and interior objects have child objects.

The management tree can be extended at run-time. This is done with the Add or Replace command and both new interior objects and new leaf objects can be created. However, the parent of any new object MUST be an existing interior object. 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.

5.2.1 Addressing object values

Object values are addressed by presenting a relative URI for the requested object. The base URI [4] 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, the command MUST be executed. The client MUST respond with an appropriate status code and the correct type of the value returned. The type of the value is returned as part of the meta information as specified in [6].

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

5.2.2 Addressing object child lists

Object child lists are addressed in the same way as object 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 object names according to [4].

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>
```

5.2.3 Permanent and dynamic objects

Objects in the management tree can be permanent or dynamic. Permanent objects are built in at device manufacture and cannot be deleted. An attempt to delete a permanent object results in the deletion of the object value, but not the object property values, provided that the ACL and AccessType allow deletions. For permanent interior objects, the command is recursive and will traverse down the tree until it terminates by either deleting a permanent object value or a dynamic object. All dynamic objects, both leaf and interior, in this branch will be deleted.

Dynamic objects can be created and deleted at run-time. The Add or Replace command is used to create new objects. The Delete command is used to delete an object and all its properties. If a deleted object has children, these are deleted too. Permanent objects cannot be children of dynamic objects.

The complete layout of the permanent objects in the management tree is reflected in the device description.

The value of a permanent object can also be made permanent, i.e. not possible to delete or replace. This is achieved by specifying the `AccessType` element in the device description of the permanent object in such a way that the object value cannot be modified. See chapter 7 for more information.

5.2.4 Adding dynamic objects

As stated in the previous section the Add command is used to create new objects. The definition of the Add command in [3] states that the object addressed must not exist in the tree. The name of the newly created object 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` object as specified in [2].

Example:

The following Add command:

```
<Add>
  <CmdID>5</CmdID>
  <Item>
    <Target>
      <LocURI>Vendor/Ring_signals/My_beep</LocURI>
    </Target>
    <Data>jkhdsfKJhdsf89374h</Data>
  </Item>
</Get>
```

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

When an object is created, all the properties that this object 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 default a value or a value that is automatically updated by the client are also assigned appropriate value at object creation. Other properties will have the value `null`.

Interior objects are created in the same way. The difference is that the server must explicitly say that the new object is an interior object 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>
</Get>
```

6 Properties of management objects

6.1 Definition

Properties of management objects are used to provide meta information about the object in question. All properties in this section are run-time properties, e.g. they are available during the lifetime of their associated object. Section 7.2.2 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 object values should be interpreted
Name	The name of the object in the tree
Size	Size of the object value in bytes
Title	Human readable name
TStamp	Time stamp, date and time of last change
Type	The MIME type of the object
VerNo	Version number, automatically incremented at each modification

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

6.2 Supported properties

Devices and servers 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 objects MUST NOT for interior objects
Title	MAY
TStamp	MAY
Type	MUST for leaf objects MAY for interior objects
VerNo	MAY

6.3 Property addressing

The properties of an object are addressed by appending `?prop=<property_name>` to the object 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.

6.4 Property values

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

It is un-necessary 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>
```

6.5 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 object.
Name	Get, Replace	A Replace is equivalent to a rename of the object.
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 object.
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 object is created. The values of the newly created object properties are all `null`. However, object 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.

6.5.1 Properties of permanent objects

The semantics of most properties are independent of the permanent/dynamic status of the object to which they are associated. The exceptions to this are Format and Name, which MUST NOT be changed for permanent objects. Any attempt to perform such a change will fail.

6.6 Scope of properties

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

6.7 Detailed description of properties

6.7.1 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 [5]. All management commands received in one session are assumed to originate from the same server.

6.7.1.1 ACL and inheritance

Every object MUST implement the ACL property, but there can be no guarantee that the ACL of every object has a value assigned to it. However, the root management object MUST always have an ACL value. If a server performs a management operation on an object with no value set on the ACL the device MUST look at the ACL of the parent object 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 object MUST have an assigned ACL value. This way, management objects can inherit ACL settings from one of their ancestors.

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

6.7.1.2 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 object that MUST have a value assigned to the ACL. The default value for the root ACL SHOULD be `Add=*&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 `(403) Forbidden`.

6.7.1.3 Changing the ACL

The rules for changing the ACL of an object are different for interior objects and leaf objects.

- Interior objects
  The ACL is valid for the object and all properties that the object may have, i.e. the right to access the ACL is controlled be the ACL itself. If a server identifier has Replace access rights according to the object ACL then this server identifier can change the ACL value.

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

However, for both types of objects the right to change the ACL of the object is also controlled by the ACL of the parent object. Note that any patent object is by definition an interior object. This makes it possible for a server identifier with sufficient access to a parent object to take control of a child object. This is a two-step process where the server first changes the ACL of the child object and then can access the object value, list of children or other object properties. Note that even if a server has total access to the parent object according to the parents ACL, this does not imply direct access to the child object value. To change a child object value the child ACL value MUST be changed first.

The ability for a server identifier with access to a parent object to take control of a child object implies that any server identifier with control of the root object 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 object 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 object. A successful completion of such an operation is signalled 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 `(403) Forbidden`, is returned.

If a server successfully creates a new object with the Add or Replace command the value of the objects ACL property is initially set to `null`. This means that the value is inherited from the parent object. However, there is one exception to this rule. If a server is adding an interior object and does not have Replace access rights on the parent of the new object then the device MUST automatically set the ACL of the new object so that the creating server has Add, Delete and Replace rights on the new object.

In cases where the above rule does not apply it is RECOMMENDED that the current server identifier explicitly sets the new object ACL. This is achieved by using a Replace command on the ACL URI of the new object. 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.

6.7.1.4 ACL syntax

The ACL structure is a list of server identifiers where each identifier is associated with a list of SyncML DM command names [3]. 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 object and/or its properties are granted access.

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

```
<acl> ::= <acl-entry> | <acl> & <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.

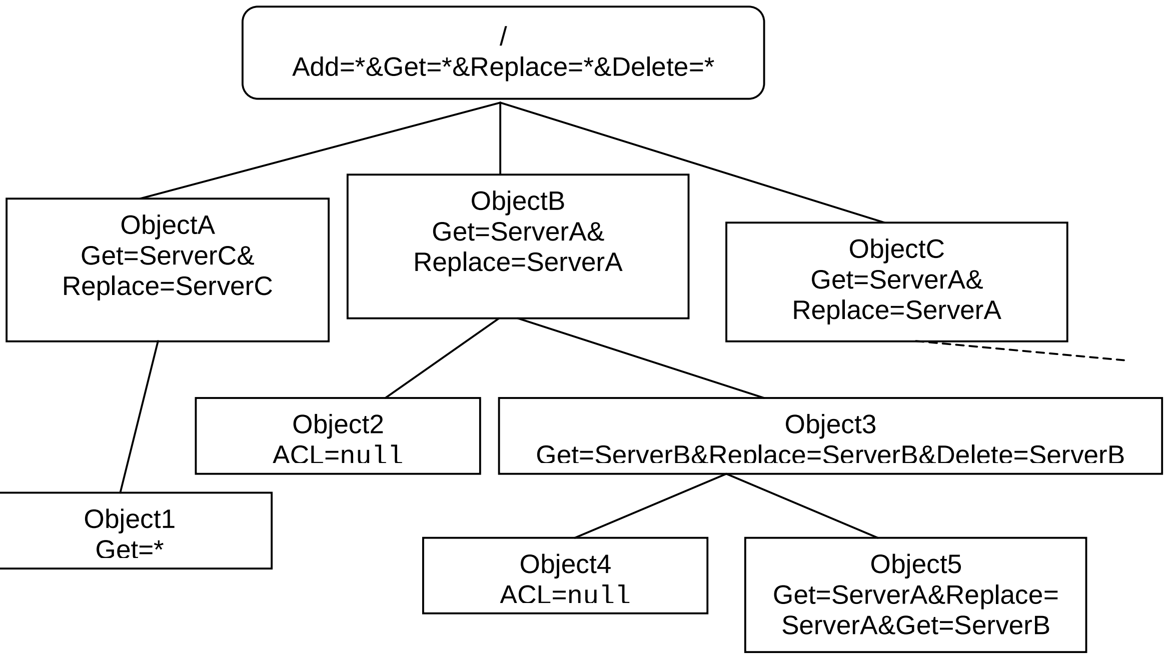
Example ACL value:

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

6.7.1.5 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 `./ObjectA/Object1`, but only ServerC can modify `./ObjectA/Object1?prop=ACL` in one operation.
- No server can directly Delete or Replace the value of `./ObjectA/Object1`.
- A Get request on `./ObjectA/Object1?prop=ACL` will return `Get=*`.
- A Get request on `./ObjectB/Object3/Object4?prop=ACL` will return `Get=ServerB&Replace=ServerB&Delete=ServerB`.
- A Replace request on `./ObjectB/Object3/Object5` by ServerB will be successful.

6.7.2 Format

The Format property always maintains information about the data format of the current object value. Allowed formats are defined in [2]. 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 6.7.7 for an example.

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

6.7.3 Name

This property reflects the name of the object to which it belongs. This is the name by which the object is addressed in the management tree. The Name property is a string with a maximum length of 80 characters.

When a new object 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 object names, before the command is executed. Since it is only the last segment of the current objects URI that is changed, the search for possible duplicate names can be limited to the siblings of the current object.

6.7.4 Size

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

When a leaf object 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 object. The value of the Size property MUST be equal to the size of the new object value in bytes.

6.7.5 Title

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

6.7.6 TStamp

This property is a record of the date and time of the last change in value of the object 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.

6.7.7 Type

The Type property always maintains information about the MIME type of the current object value. Allowed types are defined in [6]. The entity setting the value MUST supply the type information in the same command that is used to set the value. The MIME type information is carried by the Type tag in the meta information for the Item that has the data to be set. The property value is a represented by a string.

An objects description MAY specify that more than one MIME type can be stored by the object. Consequently, a server that modifies an object 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/halebop/GWName</LocURI>
    </Target>
    <Meta>
      <Format xmlns='syncml:metinf'>chr</Format>
      <Type xmlns='syncml:metinf'>text/plain</Type>
    </Meta>
    <Data>www.halebop.se</Data>
  </Item>
</Add>
```

If the Type property is implemented for interior objects the property value is undefined.

6.7.8 VerNo

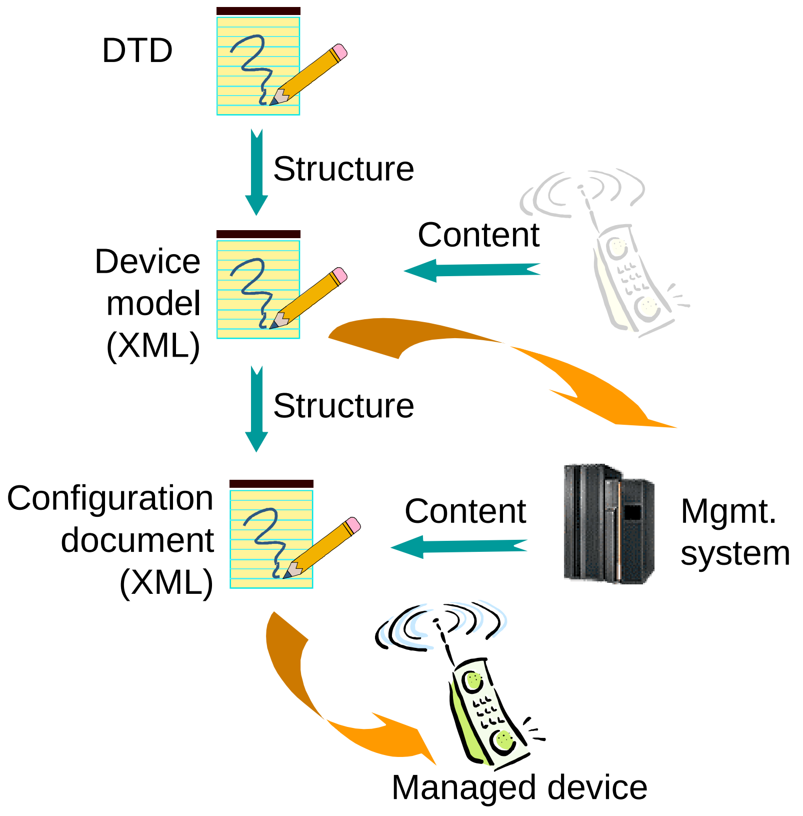
VerNo is a 16 bit unsigned integer. Each time an object with this property changes value, through a management operation or other event, this value is incremented. If the property value has reached $FFFF_{16}$, and then is incremented, it returns to 0000_{16}.

7 Device Description Framework

7.1 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 must co-exist with the existing standardized management objects, and that the borderline between the standardized objects and 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.

7.2 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.

7.3 The Description Framework DTD

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- SyncML DM Device Description Framework DTD Version 1.1   2002-02-13   -->

<!-- Copyright Notice Copyright (c) Ericsson, IBM, Lotus, Matsushita
Communication Industrial Co., LTD., Motorola, Nokia, Openwave, Palm, Inc.,
Psion, Starfish Software, Symbian and others (2002). All Rights Reserved.

Implementation of all or part of any Specification may require licenses under
third party intellectual property rights, including without limitation, patent
rights (such a third party may or may not be a Supporter). The Sponsors of the
Specification are not responsible and shall not be held responsible in any
manner for identifying or failing to identify any or all such third party
intellectual property rights.

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN ARE PROVIDED ON AN "AS IS"
BASIS WITHOUT WARRANTY OF ANY KIND AND ERICSSON, IBM, LOTUS, MATSUSHITA
COMMUNICATION INDUSTRIAL CO. LTD., MOTOROLA, NOKIA, OPENWAVE, STARFISH
SOFTWARE, SYMBIAN AND ALL OTHER SYNCML SPONSORS DISCLAIM ALL WARRANTIES,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF
THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES
OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL
ERICSSON, IBM, LOTUS, MATSUSHITA COMMUNICATION INDUSTRIAL CO., LTD., MOTOROLA,
NOKIA, OPENWAVE, STARFISH SOFTWARE, SYMBIAN OR ANY OTHER SYNCML SPONSOR BE
LIABLE TO ANY PARTY FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OF
DATA, INTERRUPTION OF BUSINESS, OR FOR DIRECT, INDIRECT, SPECIAL OR EXEMPLARY,
INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND IN CONNECTION WITH
THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH LOSS OR DAMAGE.

The above notice and this paragraph must be included on all copies of this
document that are made.

Attention is called to the possibility that implementation of this
specification may require use of subject matter covered by patent rights. By
publication of this specification, no position is taken with respect to the
existence or validity of any patent rights in connection therewith. The SyncML
Initiative is not responsible for identifying patents having necessary claims
for which a license may be required by a SyncML Initiative specification or
for conducting inquiries into the legal validity or scope of those patents
that are brought to its attention.
```

```
A patent/application owner has filed a statement of assurance that it will
grant licenses under these rights without compensation or under reasonable
rates and non-discriminatory, reasonable terms and conditions to all
applicants desiring to obtain such licenses. The SyncML Initiative makes no
representation as to the reasonableness of rates and/or terms and conditions
of the license agreements offered by patent/application owners. Further
information may be obtained from the SyncML Initiative Executive Director.-->

<!-- This DTD defines the SyncML DM Device Description Framework. This DTD is
to be identified by the URN string "syncmldm:ddf". -->

<!-- Root element -->

<!ELEMENT MgmtTree (VerDTD, Man?, Mod?, Node+)>

<!-- For this version of the DTD, the value is "1.1" -->

<!ELEMENT VerDTD (#PCDATA)>

<!ELEMENT Man (#PCDATA)>

<!ELEMENT Mod (#PCDATA)>

<!--Node is the representation of an instantiated object. Note that this
element is recursive and that 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 object 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 object may have a different set of RTProperties. An object that
only supports the mandatory properties and does not need any default values
for any property 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 | bool | chr | int | node | null | xml)>

<!ELEMENT b64 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)>

<!--The Type element contains the MIME type object. When the Type element is
used in the DFProperties element, it may contain a list of several MIME types
that the object can support at runtime. The Type property is only mandatory
for leaf objects. At run-time the Type property can only have one value at a
time.-->

<!ELEMENT Type (MIME)>

<!ELEMENT MIME (#PCDATA)>

<!ELEMENT VerNo (#PCDATA)>

<!--DFProperties=Description Framework Properties. Properties that the object
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 | bool | chr | int | node | null | xml)>

<!--Occurrence indicates how many instances of an object that can be created.
Note that each object instance must have it 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 object
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+)>
```

7.4 Framework Elements

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

7.4.1 Structural elements

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

7.4.1.1 MgmtTree

Usage: Container for one or more described objects.

Parent Elements: none

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

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

7.4.1.2 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`. The element type MUST be included in the `MgmtTree`.

Content Model: `(#PCDATA)`

7.4.1.3 Man

Usage: Specifies the manufacturer of the device.

Parent Elements: `MgmtTree`

Restrictions: This element is OPTIONAL.

Content Model: `(#PCDATA)`

7.4.1.4 Mod

Usage: Specifies the model number of the device.

Parent Elements: `MgmtTree`

Restrictions: This element is OPTIONAL.

Content Model: `(#PCDATA)`

7.4.1.5 Node

Usage: Specifies a management object.

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 objects 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>
            <DFProperties>…</DFProperties>
            <Node>
               <NodeName>GWName</NodeName>
               <DFProperties>…</DFProperties>
               <Value>gw.halebop.com</Value>
            </Node>
         </Node>
      </Node>
   </Node>
</MgmtTree>
```

7.4.1.6 NodeName

Usage: Specifies the name of the described object.

Parent Elements: `Node`

Restrictions: See [4] for general restrictions on URI. The NodeName element MAY be empty. If empty, this means that the name of the object MUST be assigned when the object is created. When the object name is assigned at object 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 object being created. See also section 6.7.3.

Content Model: `(#PCDATA)`

7.4.1.7 Path

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

Parent Elements: `Node`

Restrictions: OPTIONAL element. If omitted, the URI for the object 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 [4].

Content Model: `(#PCDATA)`

Example: The following XML is an alternative way to describe the same objects as in the example in section 7.4.1.4. 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.halebop.com</Value>
   </Node>
</MgmtTree>
```

7.4.1.8 Value

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

Parent Elements: `Node`

Restrictions: OPTIONAL element. If omitted, the object description does not specify any default value for the object. In this case the initial value new objects are undefined.

Content Model: `(#PCDATA)`

7.4.1.9 RTProperties

Usage: Aggregating element for run-time properties, i.e. properties that the objects have in a device at run-time. Used to specify which properties the described object 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 object 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?)`

7.4.1.10 DFProperties

Usage: Aggregating element for description framework properties, i.e. properties that objects 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?)`

7.4.2 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.

7.4.2.1 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 6.7.1.3.

Content Model: `(#PCDATA)`

7.4.2.2 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 object, the Format property MUST specify the correct format of the object value.

Content Model: `(b64 | bool | chr | int | node | null | xml)`

7.4.2.3 Name

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

Restrictions: See [4]. If a default property value is specified, it SHOULD be the same as the value of the `NodeName` element.

Content Model: `(#PCDATA)`

7.4.2.4 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 objects. In nodes describing leaf objects the `Size` element MUST be used within `RTProperties`.

Parent Elements: `RTProperties`

Restrictions: If a value is specified it MUST be formatted according to section 6. 4.

Content Model: `(#PCDATA)`

7.4.2.5 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 6.4.

Content Model: `(#PCDATA)`

7.4.2.6 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 6.4.

Content Model: `(#PCDATA)`

7.4.2.7 Type

Usage: Specifies support for the Type property. MAY be used to specify a default value for the property. Within its parent element, the `Type` element is defined as is optional. This is done only to facilitate the omission of the element from nodes describing interior objects. In nodes describing leaf objects the `Type` element MUST be used within `RTProperties`.

Parent Elements: `RTProperties`

Restrictions: If a default value is specified for the described object, the Type property MUST be used to specify the correct MIME type of the object value. The Type property MUST be supported by leaf objects. It MAY be supported by interior objects but for interior objects, the property value is undefined.

Content Model: `(MIME)`

7.4.2.8 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 6.4.

Content Model: `(#PCDATA)`

7.4.2.9 b64

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

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

7.4.2.10 bool

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

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

7.4.2.11 chr

Usage: SyncML format description. Specifies that the object 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`

7.4.2.12 int

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

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

7.4.2.13 node

Usage: SyncML DM format description. Specifies that the object is an interior object.

Parent Elements: `Format, DFFormat`

Restrictions: This Format MUST only be used for interior objects.

Content Model: `EMPTY`

7.4.2.14 null

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

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

7.4.2.15 xml

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

Parent Elements: `Format, DFFormat`

Restrictions: None.

Content Model: `EMPTY`

7.4.2.16 MIME

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

Parent Elements: `Type, DFType`

Restrictions: MUST only contain valid MIME type identifiers. See [6].

Content Model: `(#PCDATA)`

7.4.3 Framework property elements

The properties that described objects have in the description framework are specified with framework property elements. These are not the same as the run-time properties of an instantiated object in a device. These properties express other information about objects 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 object and the corresponding description. The following table defines the framework object properties.

Element/Property	Explanation	Usage
AccessType	Specifies which commands are allowed on the object.	MUST
DefaultValue	The object value used in a device unless specifically set to a different value.	MAY
Description	The human readable description of the object.	MAY
DFFormat	The data format of the described object.	MUST
Occurrence	Specifies the number of instances that MAY occur of the object.	MAY
Scope	Specifies whether this is a permanent or dynamic object.	MAY
DFTitle	The human readable name of the object.	MAY

DFType	The MIME type of the object value.	MUST for leaf objects. MAY for interior objects.

7.4.3.1 AccessType

Usage: Specifies which commands are supported for the described object. 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?)`

7.4.3.2 DefaultValue

Usage: Specifies the “factory default” value of the object, 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)`

7.4.3.3 Description

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

Parent Elements: `DFProperties`

Restrictions: Descriptions should be kept as short as possible.

Content Model: `(#PCDATA)`

7.4.3.4 DFFormat

Usage: Specifies the data format of the described object.

Parent Elements: `DFProperties`

Restrictions: For interior objects the Format MUST be `node`.

Content Model: `(b64 | bool | chr | int | node | null | xml)`

7.4.3.5 Occurrence

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

Parent Elements: `DFProperties`

Restrictions: If the property is omitted the object occurrence is exactly one.

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

7.4.3.6 Scope

Usage: Specifies weather the described object 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 object is `Dynamic`.

Content Model: `(Permanent | Dynamic)`

7.4.3.7 DFTitle

Usage: A human readable name for the object description.

Parent Elements: `DFProperties`

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

Content Model: `(#PCDATA)`

7.4.3.8 DFType

Usage: Specifies the MIME types of that the described object supports.

Parent Elements: `DFProperties`

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

This property is OPTIONAL for interior objects.

Content Model: `(MIME+)`

7.4.3.9 Add

Usage: Specifies support for the SyncML DM Add command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.10 Copy

Usage: Specifies support for the SyncML DM Copy command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.11 Delete

Usage: Specifies support for the SyncML DM Delete command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.12 Exec

Usage: Specifies support for the SyncML DM Exec command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.13 Get

Usage: Specifies support for the SyncML DM Get command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.14 Replace

Usage: Specifies support for the SyncML DM Replace command.

Parent Elements: `AccessType`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.15 One

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

Parent Elements: `Occurrence`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.16 ZeroOrOne

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

Parent Elements: `Occurrence`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.17 ZeroOrMore

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

Parent Elements: `Occurrence`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.18 ZeroOrN

Usage: Specifies that the described object 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)`

7.4.3.19 OneOrMore

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

Parent Elements: `Occurrence`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.20 OneOrN

Usage: Specifies that the described object 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)`

7.4.3.21 Dynamic

Usage: Specifies that the described object is dynamic.

Parent Elements: `Scope`

Restrictions: None.

Content Model: `EMPTY`

7.4.3.22 Permanent

Usage: Specifies that the described object is permanent.

Parent Elements: `Scope`

Restrictions: None.

Content Model: `EMPTY`

7.5 Shortcomings of the SyncML DM description framework

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

8 References

[1] Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, **IETF**.

[2] SyncML Device Management Standardised Objects, version 1.1, **SyncML**.

[3] SyncML Device Management Protocol, version 1.1, **SyncML**.

[4] Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, **IETF**.

[5] SyncML Device Management Security, version 1.1, **SyncML**.

[6] Assigned media types, Media Types, **IANA**.

[7] SyncML Meta-Information DTD, version 1.1, **SyncML**.