XDOP

XDOP Device Architecture 2.0

(IndeXed Device Object Protocol)

Version 2.0.6, Friday, February 17, 2012

Authors:

Norbert Menzl, mailto:n.menzl@altotec.de

Andreas Feil, mailto:a.feil@altotec.de

Martin Gruber, mailto:m.gruber@altotec.de

Table of Contents

Introduction

What is XDOP Technology?

XDOP is an attempt to define a universal, simple and lightweight device communication protocol which can be used for controlling devices. As for now there is no standard protocol on the market which can be used to control various devices from very limited microcontrollers to high end devices. Every device manufacturer defines his own protocol which leads to more effort because with every new device a vendor developer must learn and implement a new protocol. XDOP tries to close this gap by defining an transport independent protocol which main application areas are the Universal Serial Bus (USB), 802.15.4 (also ZigBee), Ethernet but also older transport layers such as RS232.

XDOP itself is a very simple and lightweight protocol, which can be implemented for almost every microcontroller (embedded systems). The minimal code size for XDOP will be in the most cases between 2k and 8k, depending on the number of variables, actions and events.

The design of XDOP is based on the following requirements

- XDOP is a universal control protocol for device objects
- The device objects are accessed by numeric indexes (mapping between indexes and names)
- The device delivers a description of its objects
- XDOP is independent of any transport layer
- The communication messages are mostly text-based and therefore human-readable
- XML can easily be embedded into the messages
- The protocol overhead is minimal
- The parsing of the messages is very simple
- The device can send events

XDOP is for devices what the SOAP protocol is for web services. XDOP defines no addressing or discovery mechanism because this is something that depends on the application and can be done by the underlying transport layer.

XDOP Example

A control point sends a text argument for action 1 to a display service in a device:

```
øA1s1øVHello world!øZ
```

The service s1 displays the message "Hello world!" and replies to the control point:

```
øA1s1øZ
```

XDOP and Streams

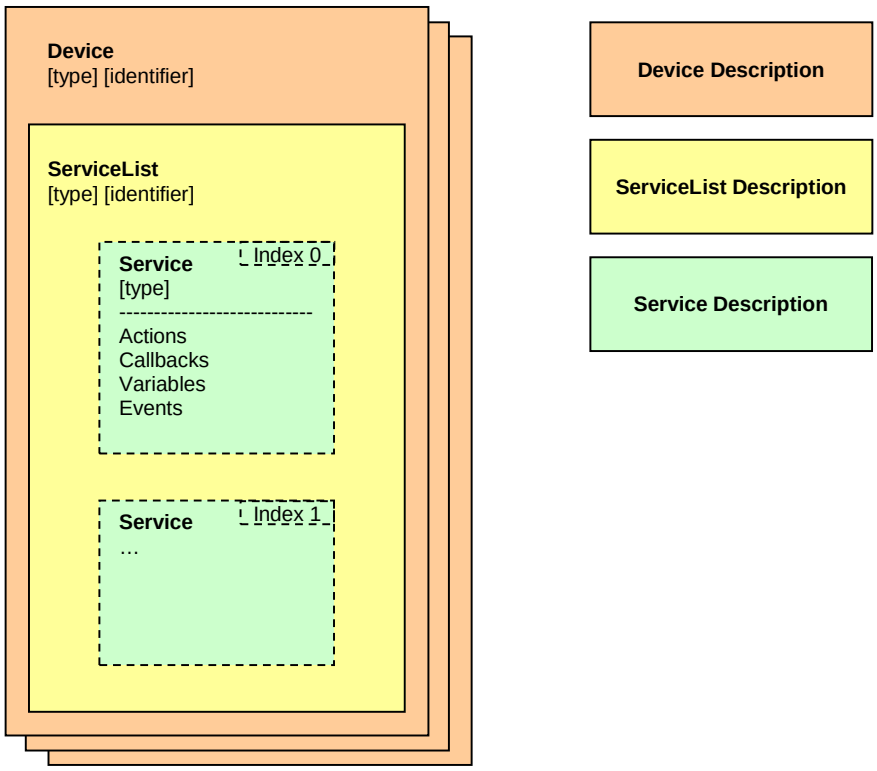
Control and eventing are not enough in many applications. Data streams like audio and video streams are not the scope of XDOP, but XDOP can be used to handle these streams, and XDOP can coexist in a separate logical or physical communication channel.

Device Model

The device model takes an object based approach. There are devices that include one or more services. Like object classes services define actions, callbacks, variables and also events which can be triggered if a service variable changes.

Note that although XDOP devices always show one device interface at a time a single physical device may include multiple logical devices. The activated interface can be changed at the connection establishing phase.

The following diagram shows the relations between device interfaces and services. The device objects are described with very compact descriptions, which can be loaded from the device or a web server.



A substantial characteristic of XDOP is the identification of the objects by indices. The aim of this index is to get a simple access to the object. He does not state anything over the object. Indices consist of integral numbers. The object description gives a mapping between the real object names and their indices. Compared with alphanumeric names a computer can evaluate numbers substantially more simply. In addition numbers offer a more compact representation than names. However these indices will not be transferred as binary numbers, but as hexadecimal character strings.

Further key characteristics of an object beside its index number are an object type and an identifier. See below for details.

Device object

The device object represents a unique instance if a specific device type. The main purpose of a device object is the providing of one or more service interfaces. A physical device can implement more than one logical device. The actual used device interface can be selected by a client at the beginning of the connection.

Service object

The service object models a functional unit of a device. You can imagine it as an object oriented class which can define following elements:

Actions

Actions are just like methods to invoke commands on a service object. The can be used to control a device.

Variables

Variables are just like the object oriented equivalents. They define methods to get or set a current aspect of the object which defines them.

Callbacks

Callbacks generally represent the same as actions. The only difference is that the device initiates them, not the client.

Events

Events are an easy option to inform a client of variable changes without the need of using polling method.

XDOP Stacks

XDOP is designed to meet various requirements for communication between all kinds of devices. It doesn't matter if you have a small microcontroller device connecting with a pc or another microcontroller device. XDOP offers a wide range of protocol features of which you can pick the ones you really need.

It's designed to work as a simple reply & request protocol which is easy to implement if required and also a full grown communication platform which allows more sophisticated features like eventing.

A complete implementation of all XDOP features could be a to heavy burden for some vendors. Either the target microcontroller lacks the necessary resources or the vendors don't have the resources to implement the complete XDOP stack.

Maybe a vendor wants to create a simple to control device which doesn't take much effort in getting familiar with the protocol features and implementing it.

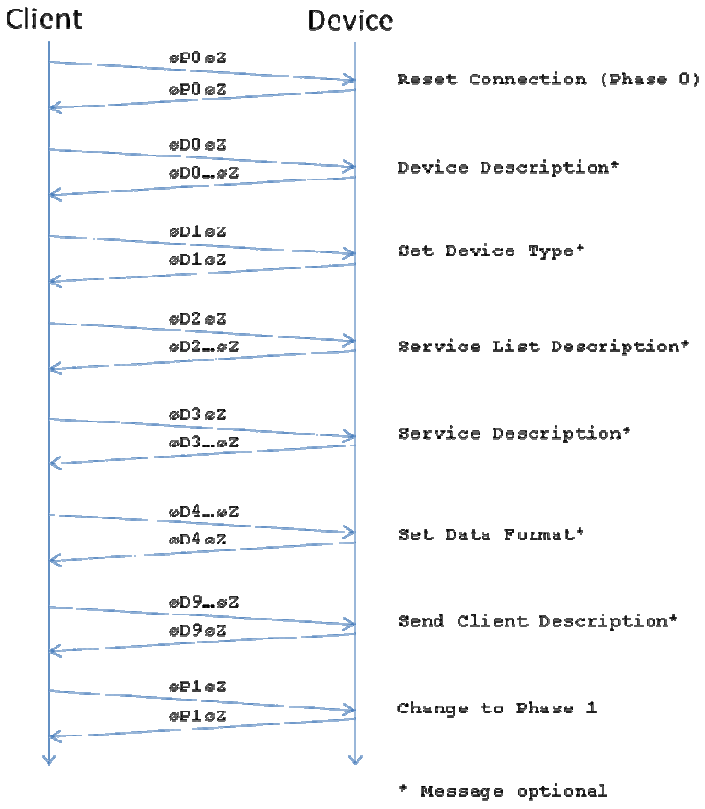
Because of this XDOP offers different stacks which give a vendor the choice which protocol features to use.

XDOP itself defines two stacks, a complete implementation of the protocol and a reduced approach. But a vendor is also free to create his own stack and picking only the protocol features he needs.

	Full XDOP Stack	Reduced XDOP Stack	Vendor Stack
Descriptions	X		?
Connection Establishment	X		?
Actions	X	X	?
Callbacks	X	X	?
Events	X	X	?
Variables	X	x	?

Connection Establishment

A XDOP connection consists of two different phases. At the beginning of a connection a client must send a reset connection message in order to reset the connection to phase 0. In this phase all descriptions can be requested, a device interface can be set and a client description can be sent to the device. Also the used data format for this connection can be set. All these messages are optional. This can be an advantage if a client knows which device it is talking to. At the end of this phase a change to phase 1 must be made which allows controlling of the device. After that no further changes of the device interface or data format are allowed for this connection.



Messages

As mentioned above XDOP messages consist mostly of text. This has many advantages contrary to a binary representation. The messages are human readable and can be processed directly by human beings (e.g. character terminal, SMS, debugging …).

There a three types of messages for normal operation: a request, a response and an event message. There are also special messages for some error conditions. The messages have all the same message format.

Common Message Format (EBNF):

```
XDOPMessage = RootElement, {ChildElement}, EndTag;
RootElement = Element;
ChildElement = Element;
EndTag = "øZ";
Element = "ø", Name, {Attribute}, [":", CharData];
Name = (UppercaseLetter - "Z"), Index;
Index = "0" | NonZeroDigit, {Digit};
Attribute = LowercaseLetter, Index;
CharData = {EscSeq | Char | AllowedSlashedOToken};

(* Slashed o's in binary data have to be replaced with "øX", if they are followed by an uppercase
letter *)
Char = UnicodeChar - "ø";
EscSeq = "øX";
AllowedSlashedOToken = "ø", (UnicodeChar - UppercaseLetter);
```

An essential part of the message format is the combination of a slashed o with an uppercase letter. An XDOP message starts always with such a character pair. The pair "øZ" marks always the end of a message. The pair "øX" is reserved for the escape sequence of the slashed o character. Every slashed o character before an uppercase letter has to be escaped, if this slashed o doesn't belong to a format element.

There are four categories of messages: request, response, event and other special messages. The following list gives an overview of all message types with their corresponding grammar name and the first characteristic message characters. A detailed description of the grammar can be found under the topic "XDOP grammar".

Request

DeviceDescriptionRequest	øD0…
SetDeviceTypeRequest	øD1…
ServiceListDescriptionRequest	øD2…
ServiceDescriptionRequest	øD3…
SetDataFormatRequest	øD4…
ClientDescriptionRequest	øD9…
ActionRequest	øA…
CallbackRequest	øC…
QueryRequest	øQ…
UpdateRequest	øU…
EventSubscribe	øS1…
EventUnsubscribe	øS0…

Response

DeviceDescriptionResponse	øD0…
SetDeviceTypeResponse	øD1…
ServiceListDescriptionResponse	øD2…
ServiceDescriptionResponse	øD3…
SetDataFormatResponse	øD4…
ClientDescriptionResponse	øD9…
ActionResponse	øA…
CallbackResponse	øC…
QueryResponse	øQ…
UpdateResponse	øU…
EventSubscribeResponse	øS1…
EventUnsubscribeResponse	øS0…

Event

EventMessage	øE…

Other

ResetConnectionMessage	øP0…
SwitchToPhaseMessage	øP1…
RejectedMessage	øR0…

All messages are described under the following topics beside the last one. This message is only used in special situations:

The message "RejectedMessage" is used by the device, if it detects an unrecognized or invalid root element or a transport error.

Description

Although a device without any services is allowed, in practice a minimal XDOP device description consists of a device, which has one service. A service can be empty but in practice will define at least one action, variable or event.

XDOP does not use the XML format for the description but simplified indexed tables. The description begins with an empty line, thus the first index number in a description is always the beginning of a new line. Each line ends with CRLF. The label names in the description tables between the colon and the equals sign are optional. The indent character is one space (ASCII 32) per level.

Description: Device description

```
empty line
0:descriptionType=standard:1
1:deviceType=urn:domain-name:device:deviceType:v
2:deviceTypeList=
 0:deviceType= urn:domain-name:device:deviceType:v
 0:deviceType= urn:domain-name:device:deviceType:v
 …
3:deviceId=unique device identifier
4:manufacturer=manufacturer name
5:manufacturerURL=URL to manufacturer site
6:productName=product name
7:productURL=product number
8:serialNumber=manufacturer's serial number
9:softwareVersion=version of the device software
10:platformInfo=info about the platform
11:tempBuildNumber=temporary build number
12:formatSmallIntegers=
 0:format=formatString
 0:format=formatString
 …
13:formatBigIntegers=
 0:format=formatString
 0:format=formatString
 …
14:formatFloats=
 0:format=formatString
 0:format=formatString
 …
15:formatBinary=
 0:format=formatString
 0:format=formatString
 …
```

0:descriptionType
Required. Type of the descriptions provided by the device. Must be ”standard:1”.

1:deviceType
Required. XDOP device type. Must begin with urn:, followed by a domain name owned by a working committee or a vendor, followed by :**device**:, followed by a device type suffix, colon, and an integer version, i.e., urn:*domain-name*:**device**:*deviceType*:*v*. Period characters in the domain name must be replaced with hyphens in accordance with RFC 2141.
The device type suffix defined must be <= 64 chars, not counting the version suffix and separating colon.
The version must be counted up if a newer version of a service type must be present. All lower versions of a device type must be supported by the actual version. If an incompatible change is made a new device type must be chosen.

2:deviceTypeList

If the device has more than one device type interface, all available interfaces (including the current device type) must be listed here. This element is only present if more than one interface is available.

0:deviceType
Required. See deviceType.

3:deviceId
Required. Unique device Identifier. Must be the same over time for a specific device instance (i.e., must survive reboots). The device ID must be <= 64 characters.

4:manufacturer
Required. Manufacturer's name. Specified by a vendor. String. Should be < 64 characters.

5:manufacturerURL
Optional. Web site for Manufacturer. Specified by a vendor. Single URL.

6:productName
Required. Product name. String. Should be < 32 characters.

7:productURL
Optional. Web site for model.

8:serialNumber
Recommended. Serial number. String. Should be < 64 characters.

9:softwareVersion
Recommended.

10:platformInfo
Optional.

11:tempBuildNumber
Optional. Can be used for development purposes to handle and to distinguish intermediate versions of a device description or device implementation without changing the version of the deviceType.

12:formatSmallIntegers
Optional. Can be used to give a client a choice for data representation (of 1 and 2 byte integer types) in XDOP messages. Standard representation is hex. Can be set using "D4" message.
Available formats:
- *decimal* (base 10 ascii string)
- *hex* (base 16 ascii string)
- *bin* (binary with escaping of slashed o, TODO Norbert unsigned vs. signed) LE, BE

13:formatBigIntegers
Optional. Can be used to give a client a choice for data representation (of 4 and 8 byte integer types) in XDOP messages. Standard representation is hex. Can be set using "D4" message.
Available formats:
- *decimal* (base 10 ascii string)
- *hex* (base 16 ascii string)
- *bin* (raw binary representation in little endian or big endian, slashed o must be escaped
  for unsigned types, leading zero bytes can be left out
  for unsigned types, leading sign bytes can be left out (the sign still has to be in the most significant bit)
  the receiver then extends the value (by adding zeroes for unsigned types and by sign extension for signed types)

  examples for 32-bit signed integers:
  1 (0x00000001) can be transported as 0x01
  -1 (0xFFFFFFFF) can be transported as 0xFF
  -256 (0xFFFFFF00) can be transported as 0xFF00 (NOT as 0x00, because then the sign would be lost)
  255 (0x000000FF) can be transported as 0x00FF (NOT as 0xFF, because then the sign would be lost)

  examples for 32-bit unsigned integers:
  255 (0x000000FF) can be transported as 0xFF
  0xFFFFFFFF can only be transported as 0xFFFFFFFF

)

14:formatFloats
Optional. Can be used to give a client a choice for data representation in XDOP messages. Standard representation is hex. Can be set using "D4" message.
Available formats:
- *decimal* (base 10 ascii string which matches %f format of c's printf)
- *hex* (base 16 ascii string which represents IEEE 754 Byte representation of a float) LE, BE
- *bin* (IEEE 754 Byte representation of a float transported as binary with escaping of slashed o) LE, BE

15:formatBinary
Optional. Can be used to give a client a choice for data representation in XDOP messages. Standard representation is hex. Can be set using "D4" message.
Available formats:
- *hex* (base 16 coded ascii string) LE, BE
- *bin* (binary with escaping of slashed o) LE, BE

Description: ServiceList description

The ServiceList description is a list of all available services, which are served by a device.

```
empty line
0:descriptionType=standard:1
1:serviceList=
 0:serviceIndex=service index
 1:serviceType=urn:domain-name:service:serviceType:v
 2:serviceId=unique service identifier
```

0:descriptionType
Required. Type of the Descriptions provided by the device. Must be ”standard:1”.

1:serviceList
Optional. Contains the following sub elements:

0:serviceIndex
Required. Defines the XDOP mapping index. Used to identify this service by index.

1:serviceType
Required. XDOP service type. Must not contain a hash character (#, 23 Hex in UTF-8). Must begin with urn:, followed by a domain name owned by a working committee or a vendor, followed by :**service**:, followed by a service type suffix, colon, and an integer service version, i.e., urn:*domain-name*:**service**:*serviceType*:*v*. Period characters in the domain name must be replaced with hyphens in accordance with RFC 2141.
The service type suffix must be <= 64 characters, not counting the version suffix and separating colon.

2:serviceId
Required. Service identifier. Must be unique within this device description. While the serviceIndex is a unique number to identify a service within the device, the serviceId provides for a clearer and more detailed identification of a service, mainly if there are multiple services of the same type. The serviceId can be used to access a service by name in a control application.
The service ID must be <= 64 characters.

Description: Service description

The description for a service defines actions, callbacks and variables.

Each service may have zero or more actions. Each action may have zero or more arguments. Any combination of these arguments may be input or output parameters. Per definition the order of the parameters is defined by their appearance. All input parameters must be defined before any output parameter.

Each service may have zero or more callbacks. Each callback may have zero or more arguments. Any combination of these arguments may be input or output parameters. If a callback has one or more output arguments, the first of these arguments is the return value.

A service can be empty but should at least define one action, variable or event.

```
empty line
0:descriptionType=standard:1
1:serviceType=urn:domain-name:service:serviceType:v
2:actionList=
 0:actionIndex=action index
 1:name=action name
 2:argumentList=
  0:argumentIndex=argument index
  1:name=argument name
  2:direction=in xor out
  3:simpleType=argument simple data type
  … see variable
  8:shortInfo=short argument info text string
 3:shortInfo=short action info text string
 0:actionIndex=action index
 1:name=action name
 2:argumentList=
  0:argumentIndex=argument index
  1:name=argument name
  2:direction=in xor out
  4:extendedType=
   0:simpleType=argument simple data type
   … see variable
 0:actionIndex=action index
 1:name=action name
 2:argumentList=
  0:argumentIndex=argument index
  1:name=argument name
  2:direction=in xor out
  5:dataTypeStruct=
   0:name=field name
   1:simpleType=field simple data type
   0:name=field name
   2:extendedType=
   … see variable
   0:name=field name
   3:dataTypeStruct=
    0:name=field name
    … see dataTypeStruct
   0:name=field name
   4:dataTypeLink=data type link name
   …
 0:actionIndex=action index
 1:name=action name
 2:argumentList=
  0:argumentIndex=argument index
  1:name=argument name
  2:direction=in xor out
  6:dataTypeLink=data type link name
  8:shortInfo=short argument info text string
 3:shortInfo=short action info text string
3:callbackList=
 0:callbackIndex=callback index
 1:name=action name
 2:argumentList=
 … see action
4:variableList=
 0:variableIndex=variable index
 1:name=variable name
 2:flags=any combination of the letters Q (Query), U (Update) or E (Event)
 3:simpleType=variable simple data type
 7:shortInfo=short variable info text string
 0:variableIndex=variable index
 1:name=variable name
 2:flags=any combination of the letters Q (Query), U (Update) or E (Event)
 3:simpleType= variable simple data type[empty for list or number for array]
 7:shortInfo=short variable info text string
 0:variableIndex=variable index
```

```
 1:name=variable name
 2:flags=any combination of the letters Q (Query), U (Update) or E (Event)
 4:extendedType=
   0:simpleType=variable simple data type
   1:allowedValueRange=
    0:minimum=minimum value
    1:maximum=maximum value
    2:step=increment value
   3:unit=unit string
 7:shortInfo=short variable info text string
 0:variableIndex=variable index
 1:name=variable name
 2:flags=any combination of the letters Q (Query), U (Update) or E (Event)
 4:extendedType=
  0:simpleType= variable simple data type[empty for list or number for array]
  2:enumList=
   0:nameValuePair=name=value
   0:nameValuePair=name=value
   …
  3:unit=unit string
 7:shortInfo=short variable info text string
 0:variableIndex=variable index
 1:name=variable name
 2:flags=any combination of the letters Q (Query), U (Update) or E (Event)
 5:dataTypeStruct=
  0:name=field name
  1:simpleType=field simple data type
  0:name=field name
  2:extendedType=
   … see variable
  0:name=field name
  3:dataTypeStruct=
   0:name=field name
   … see dataTypeStruct
  0:name=field name
  4:dataTypeLink=data type link name
 0:variableIndex=variable index
 1:name=variable name
 2:flags=any combination of the letters Q (Query), U (Update) or E (Event)
 6:dataTypeLink=data type link name
 7:shortInfo=short variable info text string
5:dataTypeLinkList=
 0:name=data type link name
 1:simpleType=simple data type
 0:name=data type link name
 1:simpleType= simple data type[empty for list or number for array]
 0:name=data type link name
 2:extendedType=
  0:simpleType=simple data type
  1:allowedValueRange=
   0:minimum=minimum value
   1:maximum=maximum value
   2:step=increment value
  3:unit=unit string
 0:name=data type link name
 2:extendedType=
  0:simpleType= variable data type[empty for list or number for array]
  2:enumList=
   0:nameValuePair=name=value
   0:nameValuePair=name=value
  …
  3:unit=unit string
0:name=data type link name
 3:dataTypeStruct=
  0:name=field name
  1:simpleType=simple data type
  …
```

0:descriptionType

Required. Type of the description provided by the device. Must be ”standard:1”.

1:serviceType

Required. XDOP service type. Must not contain a hash character (#, 23 Hex in UTF-8). Must begin with urn:, followed by a domain name owned by a working committee or a vendor, followed by :service:, followed by a service type suffix, colon, and an integer service version, i.e., urn:*domain-name*:service:*serviceType*:*v*. Period characters in the domain name must be replaced with hyphens in accordance with RFC 2141.
The service type suffix must be <= 64 characters, not counting the version suffix and separating colon.
The version must be counted up if a newer version of a service type is defined. All lower versions of a service type must be supported by the actual version. If an incompatible change is made a new service type must be chosen.

2:actionList

Required if and only if the service has actions. (Each service may have >= 0 actions.) Contains the following sub elements:

0:actionIndex
Required. Defines the XDOP mapping index. Used to identify this action by index.

1:name
Required. Name of action.

2:argumentList
Required if and only if parameters are defined for action. (Each action may have >= 0 parameters.) Contains the following sub elements:

0:argumentIndex
Required. Defines the XDOP mapping index. Used to identify this action by index.

1:name
Required. Name of argument. May only contain USASCII letters (A-Z, a-z), USASCII digits (0-9), and underscores ("_").

2:direction
Required. Whether argument is an input or output parameter. Must be in xor out. Any in arguments must be listed before any out arguments. The first out argument is the return value.

3:simpleType
Required. See below for details. Either "simpleType”, "extendedType, "dataTypeStruct” or "dataTypeLink” must be given.

4:extendedType
Required. See below for details. Either "simpleType”, "extendedType, "dataTypeStruct” or "dataTypeLink” must be given.

5:dataTypeStruct
Required. See below for details. Either "simpleType”, "extendedType, "dataTypeStruct” or "dataTypeLink” must be given.

6:dataTypeLink
Required. See below for details. Either "simpleType”, "extendedType, "dataTypeStruct” or "dataTypeLink” must be given.

7:unit
Recommended. See variableList sub element unit for more details.

8:shortInfo
Optional.

3:callbackList

Required if and only if the service has callbacks. (Each service may have >= 0 callbacks.) Contains the same sub elements as the actionList

4:variableList

Required if and only if the service has variables. (Each service may have >= 0 variables.) Contains the following sub elements:

0:variableIndex
Required. Defines the XDOP mapping index. Used to identify this variable by index.

1:name
Required. Name of variable. May only contain USASCII letters (A-Z, a-z), USASCII digits (0-9), and underscores ("_").

2:flags
Required. Defines how this variable can be used. The flags are composed of one or more letters of the following list:
Q = this variable can be queried
U = this variable can be updated
E = this variable send events

3:simpleType

Required. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given. Must be one of the simpleType values shown below.

4:extendedType

Required. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given. Contains the following elements:

0:simpleType

One of the simpleTypes. See below for details. Numeric types can be extended by array or list definition.

1:allowedValueRange

Recommended. Defines bounds for legal numeric values; defines resolution for numeric values. Defined only for numeric data types. At most one of allowedValueRange and enumList may be specified and must not be present in conjunction with dataTypeLink. Contains the following sub elements:

0:minimum

Required. Inclusive lower bound.

1:maximum

Required. Inclusive upper bound.

2:step

Recommended. Size of an increment operation, i.e., value of *s* in the operation *v* = *v* + *s*.

2:enumList

Recommended. Defines name-value pairs. Contains one or more of the following sub element:

0:nameValuePair

Required. Defines a pair with a name and a value, which are assigned together with the equals sign.

5:dataTypeStruct

Required. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given. Contains the following elements:

0:name

Required. The name of the struct field.

1:simpleType

Required. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given.

2:extendedType

Required. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given.

3:dataTypeStruct

Required. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given. If this element is given this means that the current struct contains another struct as child element.

4:dataTypeLink

Required. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given.

6:dataTypeLink

Required. See below for details. Either "dataType" or "dataTypeLink" must be given.

7:unit

Recommended. The unit of a value is mainly useful, if there is no additional document about the service functionality or for universal control applications, which are loading the description at runtime and have no static information about the service.

The unit string has no strong grammar and can be freely chosen. But there a few recommendations:

- Use SI units and SI prefixes, if applicable. Look at http://www.bipm.org/en/si/
- Don't use negative exponents but divisors. For example: use **m/s²** in place of **ms^{-2}** or **1/min** in place of **min^{-1}**
- Use SI prefixes for scaling, if applicable. Otherwise use multipliers and divisors. For example: use **10m/h** in place of **dkm/h** or **W/10** in place of **dW** but **dm** in place of **m/10**.

8:shortInfo

Optional.

5:dataTypeLinkList

Optional. Defines all linked dataTypes. Contains the following sub elements:

0:name

Required. Link name used by one of the element dataTypeLink for an argument or a variable description.

1:simpleType

Required. See variableList for details. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given.

2:extendedType

Required. See variableList for details. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given.

3:dataTypeStruct

Required. See variableList for details. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given.

4:dataTypeLink
Required. See variableList for details. Either "simpleType", "extendedType, "dataTypeStruct" or "dataTypeLink" must be given.

4:unit
Recommended. See variableList sub element unit for more details.

simpleType details:

Must be one of the following values, in parentheses is a reference to the XDOP Grammar in EBNF:

ui1 (UnsignedInteger)
: Unsigned 1 Byte int. Same format as int without leading sign. Must be between 0 and 255

ui2 (UnsignedInteger)
: Unsigned 2 Byte int. Same format as int without leading sign. Must be between 0 and 65535

ui4 (UnsignedInteger)
: Unsigned 4 Byte int. Same format as int without leading sign. Must be between 0 and 4294967295

ui8 (UnsignedInteger)
: Unsigned 8 Byte int. Same format as int without leading sign. Must be between 0 and 18446744073709551615

i1 (Integer)
: 1 Byte int. Same format as int. Must be between -128 and 127

i2 (Integer)
: 2 Byte int. Same format as int. Must be between -32768 and 32767

i4 (Integer)
: 4 Byte int. Same format as int. Must be between -2147483648 and 2147483647.

i8 (Integer)
: 8 Byte int. Same format as int. Must be between -18446744073709551616 and 18446744073709551615.

int (Integer)
: Fixed point, integer number. May have leading sign. May have leading zeros. (No currency symbol.) (No grouping of digits to the left of the decimal, e.g., no commas.)

r4 (Float)
: 4 Byte float. Same format as float. Must be between -3.40282347E+38 to -1.17549435E-38 for negative values, and between 1.17549435E-38 and 3.40282347E+38 for positive values.

r8 (Float)
: 8 Byte float. Same format as float. Must be between -1.79769313486232E308 and -4.94065645841247E-324 for negative values, and between 4.94065645841247E-324 and 1.79769313486232E308 for positive values, i.e., IEEE 64-bit (8-Byte) double.

float (Float)
: Floating point number. Mantissa (left of the decimal) and/or exponent may have a leading sign. Mantissa and/or exponent may have leading zeros. Decimal character in mantissa is a period, i.e., whole digits in mantissa separated from fractional digits by period. Mantissa separated from exponent by E. (No currency symbol.) (No grouping of digits in the mantissa, e.g., no commas.)

char (Char)
: Unicode string. One character long.

string (String)
: Unicode string. No limit on length.

date (Date)
: Date in a subset of ISO 8601 format without time data.

time (Time)
: Time in a subset of ISO 8601 format with no date and optional time zone. If time zone is present it indicates the difference between local time and UTC. If no time zone is present the time indicates a local time (recommended).

dateTime (DateTime)
: Date in a subset of ISO 8601 format with optional time and time zone.

boolean (Boolean)
: 0 for false; 1 for true.

bin
: Binary representation of data.

uri (URI)
: Universal Resource Identifier.

uuid (UUID)
: Universally Unique ID. Hexadecimal digits representing octets. Optional embedded hyphens are ignored.

nameValueList
: Pairs of name and value strings, concatenated with "=" and terminated with ";": "name=value;".

XML
: XML text.

The [] may be used to indicate a list or array of elements. An empty [] parentheses signals a list otherwise the number of elements in the array must be given. Decimal or hex representation of values must be delimited with ";", binary transport has no delimiter because fix length of data types will be used.

The **dataTypeLink** element of an **argument** or **variable** definition provides the possibility to centralize a common dataType definition in the dataTypeLinkList.

The **enumList** and **allowedValueRange** elements may be used to indicate optional device capabilities. All defined values in an enum list must be supported be a device implementation and no other values are allowed. Therefore an enumList cannot be combined with an allowedValueRange. If device capabilities are expected to change during device operation, working committees should define separate actions to detect device capabilities rather than embedding capabilities information in the service description.

Description: Client description

```
empty line
0:descriptionType=standard:1
1:manufacturer=manufacturer name
2:manufacturerURL=URL to manufacturer site
3:productName=product name
4:productURL=product number
5:softwareVersion=version of the client software
6:platformInfo=info about the platform
7:tempBuildNumber=temporary build number
```

0:descriptionType
Required. Type of the Descriptions provided by the client. Must be ”standard:1”.
1:manufacturer
Required. Manufacturer's name. Specified by a vendor. String. Should be < 64 characters.
2:manufacturerURL
Optional. Web site for Manufacturer. Specified by a vendor. Single URL.
3:productName
Required. Product name. String. Should be < 32 characters.
4:productURL
Optional. Web site for model.
5:softwareVersion
Recommended.
6:platformInfo
Optional.
7:tempBuildNumber
Optional. Can be used for development purposes to handle and to distinguish intermediate versions of a client description or client implementation without changing the version of the deviceType.

Description: Retrieving a description

For the syntax of the XDOP requests and responses see the chapter Messages.

Device Description

Description request:

```
øD0 øZ
```

Description response:

```
øD0:
device description
øZ
```

Error response:

```
øD0
        øF0:error code
        øF1:error description
øZ
```

Service List Description

Description request:

```
øD2 øZ
```

Description response:

```
øD2:
service list description
øZ
```

Error response:

```
øD2
        øF0:error code
        øF1:error description
øZ
```

Service Description

Description request:

```
øD3sservice indexøZ
```

Description response:

```
øD3sservice index:
service description
øZ
```

Error response:

```
øD3sservice index
        øF0:error code
        øF1:error description
øZ
```

Data Format

Groups:

0 = small Integers (ui1, i1, ui2, i2)
1 = big Integers (ui4, i4, ui8, i8)
2 = floats
3 = binary data

Defined data formats:

integers:

- decimal
- hex.ascii (hexadecimal based ascii string, default serialisation)
- bin.escaped (till 32 bit: little endian, 64 bit: tbd)

floats:

- decimal
- bin.hex (hexadecimal representation of a float value according to IEEE 754, till 32 bit: little endian, 64 bit: tbd, default serialisation)
- bin.escaped (binary data with escaped XDOP character ø representating a float value according to IEEE 754, till 32 bit: little endian, 64 bit: tbd)
- C99_%a

binary data:

- bin.hex (hexadecimal serialization, default serialisation)
- bin.escaped (binary data with escaping of XDOP character ø)
- bin.base64

Set data format request:

```
øD4
 øGgroup index:FormatString
 Other group values here, if any
øZ
```

Set data format response:

```
øD4 øZ
```

Error response:

```
øD4
       øF0:error code
       øF1:error description
øZ
```

Setting a device type interface

For the syntax of the XDOP requests and responses see the chapter Messages.

Setting a device type interface

Setting request:

```
øD1:deviceTypeString øZ
```

Description response:

```
øD1 øZ
```

Error response:

```
øD1
        øF0:error code
        øF1:error description
øZ
```

Client Description

Description request:

```
øD9øZ
```

Description response:

```
øD9:
client description
øZ
```

Error response:

```
øD9
        øF0:error code
        øF1:error description
øZ
```

Connection

Phase 0

Every XDOP connection starts in phase 0 in which only descriptions can be requested and send. In addition the used device interface and data format for this connection can be set.

At the beginning of every xdop communication this request should be send to reset the current connection to phase 0. This is required because some transport layers such as RS232 connections cannot explicitly handle connections where the XDOP stack could be reset when opening a new connection.

In order to control a device a progress to phase 1 has to be made.

Phase 0 request:

```
øP0:øZ
```

Phase 0 response:

```
øP0 øZ
```

Error response:

```
øP0
        øF0:error code
        øF1:error description
øZ
```

Phase 1

This phase is used to control a device. Once in this phase no other phase 1 request must be send. Otherwise a phase 0 request is allowed indicating a connection reset.

Phase 1 request:

```
øP1:øZ
```

Phase 1 response:

```
øP1øZ
```

Error response:

```
øP1
        øF0:error code
        øF1:error description
øZ
```

Control

The remainder of this section explains in detail how control and query messages are formatted.

Control: Action

Control points may invoke actions on a device's services and receive results or errors back.

Control: Action: Invoke

An XDOP message begins with a root element followed by optional sub elements and is finished with an end element

All XDOP elements start with a slashed o character ('ø') followed by an upper case letter. The root element has one attribute for the service index. This attribute is optional and the default is zero. The element and element values are delimited with a colon (':'). If there is no element value, the colon is not needed. One sub-element is finished with the beginning of the next sub-element.

A slashed o character in the element value has to escaped with **øX**.

The last element of a message is **øZ**

White space characters between the XDOP elements are not allowed. Only between the XDOP messages are all characters allowed except **ø[A-Z]** without the escape sequence **øX**.

Values in italics are placeholders for actual values.

IMPORTANT: For a better readability the XDOP elements are separated in individual lines, which are not allowed in a real message!

```
øAaction indexsservice index
       øVin arg value
       other in args and their values go here, if any
øZ
```

Shortest possible action without arguments:

```
øAaction indexøZ
```

Example 1: Action request (action index = 0, service index = 1 and device index = 2) with two input arguments (argument index 0 and 1).

```
øA0s1øV1.0øVHello world!øZ
```

Example 2: Action request or response (action index 1, only one service exists) with one in argument.

```
øA1øV23øZ
```

Control: Action: Response

Normal response:

```
øAaction indexsservice index
       øVout arg value
       other out args and their values go here, if any
øZ
```

Error response:

```
øAaction indexsservice index
       øF0:error code
       øF1:error description
øZ
```

Control: Query for variable

In addition to invoking actions on a device's service, control points may also poll the service for the value of a variable by sending a query message. A query message may query only one variable; multiple query messages must be sent to query multiple variables.

This query message is decoupled from the service's eventing (if any). If a variable is moderated, then querying for the value of the variable will generally yield more up-to-date values than those received via eventing. The section on Eventing describes event moderation.

Control: Query: Invoke

```
øQvariable indexsservice indexddevice indexøZ
```

Control: Query: Response

Normal response:

```
øQvariable indexsservice indexddevice index:variable value øZ
```

Error response:

```
øQvariable indexsservice indexddevice index
       øF0:error code
       øF1:error description
øZ
```

Eventing

The remainder of this section first explains subscription, including details of subscription messages, renewal messages, and cancellation messages. Second, it explains in detail how event messages are formatted and sent to control points, and the initial event message.

Eventing: Subscription

A service has eventing if and only if one or more of the variables are evented.

Below is an explanation of the specific format of requests, responses, and errors for subscription, renewal, and cancellation messages.

Eventing: Subscribing

```
øS1sservice indexddevice index øZ
```

Normal response:

```
øS1sservice indexddevice indexøZ
```

Error response:

```
øS1sservice indexddevice index
       øF0:error code
       øF1:error description
øZ
```

Eventing: Renewing a subscription

Because there is no time limit for a subscription, the renewing of a subscription is not necessary.

Eventing: Canceling a subscription

```
øS0sservice indexddevice index øZ
```

Normal response:

```
øS0sservice indexddevice index øZ
```

Error response:

```
øS0sservice indexddevice index
       øF0:error code
       øF1:error description
øZ
```

Callbacks

The remainder of this section explains in detail how callback messages are formatted.

Device objects may invoke callbacks on connected clients and receive results or errors back. A client is not bound to answer a callback so therefore a device must consider receiving a timeout when sending a callback request.

Callback: Invoke

Callbacks have the same syntax as action messages. The only difference is name of the root element. In case of actions the upper case letter A was used. Callbacks use the letter C.

Values in italics are placeholders for actual values.

IMPORTANT: For a better readability the XDOP elements are separated in individual lines, which are not allowed in a real message!

```
øCcallback indexsservice index
       øVin arg value
       other in args and their values go here, if any
øZ
```

Shortest possible action without arguments:

```
øCcallback indexøZ
```

Example 1: Callback request (callback index = 0, service index = 1) with two input arguments .

```
øC0s1øV1.0øVHello world!øZ
```

Example 2: Callback request or response (callback index 1, only one service exist) with one in argument.

```
øC1øV23øZ
```

Callback: Response

Normal response:

```
øCcallback indexsservice index
       øVout arg value
       other out args and their values go here, if any
øZ
```

Error response:

```
øCcallback indexsservice index
       øF0:error code
       øF1:error description
øZ
```

Eventing: Event messages

A service publishes changes to its variables by sending event messages. These messages contain the indexes of one or more variables and the current value of those variables. Event messages should be sent as soon as possible to get accurate information about the service to subscribers and allow subscribers to display a responsive user interface. If the value of more than one variable is changing at the same time, the publisher should bundle these changes into a single event message to reduce processing and network traffic.

An initial event message is sent when a subscriber first subscribes; this event message contains the indexes and values for all evented variables and allows the subscriber to initialize its model of the state of the service. This message should be sent as soon as possible after the publisher accepts a subscription. This message should always be sent, even if the control point unsubscribes before the message is delivered.

Event messages are tagged with an event key, which has one digit. The event key for a subscription is initialized to 0 when the publisher sends the initial event message. For each subsequent event message, the publisher increments the event key for a subscription, and includes that updated key in the event message. Any implementation of event keys should handle overflow and wrap the event key from 9 back to 1 (not 0). Subscribers must also handle this special case when the next event key is not an increment of the previous key.

To repair an event subscription, e.g., if a subscriber has missed one or more event messages, a subscriber must unsubscribe and re-subscribe. By doing so, the subscriber will get a new initial event message, and a new event key.

```
øEevent keysservice index
       øVvariable index:variable value
       other variables and their values go here, if any
øZ
```

Error Codes

errorCode	errorDescription	Description
401	Invalid Action Index	No action by that index at this service.
402	Invalid Args Count	Could be any of the following: not enough in args, too many in args
600	Argument Value Invalid	The argument value is invalid
601	Argument Value Out of Range	An argument value is less than the **minimum** or more than the **maximum** value of the **allowedValueRange**, or is not in the **allowedValueList.**
603	Out of Memory	The device does not have sufficient memory available to complete the action. This may be a temporary condition; the control point may choose to retry the unmodified request again later and it may succeed if memory is available.
604	Human Intervention Required	The device has encountered an error condition which it cannot resolve itself and required human intervention such as a reset or power cycle. See the device display or documentation for further guidance.
605	String Argument Too Long	A string argument is too long for the device to handle properly.
600-899	*TBD*	Action-specific errors for non-standard actions. Defined by Vendors
450	No root element	End Tag without start Tag
451	Unknown root element	End Tag with unknown start Tag
452	Invalid Request Format	Parsing according to EBNF failed
454	Invalid Service Index	
455	Invalid Variable Index	
456	Invalid Description Index	
457	Query not allowed for Index	
458	Update not allowed for Index	
459	Invalid Reset Index	
900	Message Buffer Overflow	The input buffer of the device was full, before an end Tag has been found.
95x	Detected transport error	

XDOP Grammar (EBNF)

Common

```
(* any Unicode character, excluding the surrogate blocks, FFFE, and FFFF. *)
UnicodeChar = TAB | CR | LF | ? Unicode characters #x20-#xD7FF ? |
              ? Unicode characters #xE000-#xFFFD ? | ? Unicode characters #x10000-#x10FFFF ?;

TAB = ? ASCII TAB character ?;
CR = ? ASCII CR character ?;
LF = ? ASCII LF character ?;
CRLF = CR, LF;

UppercaseLetter =       "A" | "B" | "C" | "D" | "E" | "F" | "G"
                      | "H" | "I" | "J" | "K" | "L" | "M" | "N"
                      | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
                      | "V" | "W" | "X" | "Y" | "Z";

LowercaseLetter =       "a" | "b" | "c" | "d" | "e" | "f" | "g"
                      | "h" | "i" | "j" | "k" | "l" | "m" | "n"
                      | "o" | "p" | "q" | "r" | "s" | "t" | "u"
                      | "v" | "w" | "x" | "y" | "z";

Digit = "0" | NonZeroDigit ;

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

HexDigit = Digit | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f";

EndTag = "øZ";
Element = "ø", Name, {Attribute}, [":", CharData];
Name = (UppercaseLetter - "Z"), Index;
Index = "0" | NonZeroDigit, {Digit};
Attribute = LowercaseLetter, Index;
CharData = {EscSeq | Char | AllowedSlashedOToken};

(* Shlashed o's in character data have to be replaced with "øX", if they are followed by an
uppercase letter *)
Char = UnicodeChar - "ø";
EscSeq = "øX";
AllowedSlashedOToken = "ø", ( UnicodeChar - UppercaseLetter );
```

Data Types

```
XDOPNumericValue =     UnsignedInteger | Integer | Float

 (* All data types except string, these data types are uncritical for transport, because they
don’t contain slashed o's *)
XDOPSimpleValue =      XDOPNumericValue | Date | DateTime | Time |
                       Boolean | BinHex | Char | UUID | Base64Binary;

XDOPStruct = EscapedString {"øT” EscapedString};

 (* Values in unescaped form *)
XDOPValue = XDOPSimpleValue | String;

(* Values in escaped form for XDOP messages*)
XDOPEscapedValue = XDOPSimpleValue | EscapedString | XDOPStruct;
EscapedString = CharData;


UnsignedInteger = {Digit}; (* ui1, ui2, ui4 *)

Integer = [Sign], {Digit}; (* i1, i2, i4, int *)
```

```
Sign = "+" | "-";

Float = [Sign], {Digit}, [".", {Digit}], [ ( "e" | "E" ),[Sign] , {Digit} ]; (* r4, r8, float *)

Date                  = FullDate; (* date *)
DateTime              = FullDate "T" PartialTime | FullTime;
Time                  = PartialTime | FullTime;
Boolean = "0" | "1"; (* boolean *)

BinHex = {HexOctet}; (* bin.hex *)

Char = UnicodeChar; (* char *)

String = {Char}; (* string *)

(*
From RFC4122
changes: made hyphens optional
*)
UUID                          = TimeLow, ["-"], TimeMid, ["-"], TimeHighAndVersion, ["-"],
ClockSeqAndReserved, ClockSeqLow ["-"], Node;
TimeLow                       = 4*HexOctet
TimeMid                       = 2*HexOctet
TimeHighAndVersion  = 2*HexOctet
ClockSeqAndReserved   = HexOctet
ClockSeqLow                   = HexOctet
Node                          = 6*HexOctet
HexOctet                      = HexDigit, HexDigit;

(* datetime is a subset of RFC 3339
changes: fractions of seconds removed, "Z" and "T" must be uppercase
*)
DateFullYear   = 4Digit;
DateMonth      = 2Digit; (* 01-12 *)
DateMday       = 2Digit; (* 01-28, 01-29, 01-30, 01-31 based on month/year *)
TimeHour       = 2Digit; (* 00-23 *)
TimeMinute     = 2Digit; (* 00-59 *)
TimeSecond     = 2Digit; (* 00-58, 00-59, 00-60 based on leap second rules *)
TimeNumOffset  = ("+" | "-"), TimeHour, ":", TimeMinute;
TimeOffset     = "Z" | TimeNumOffset;

PartialTime    = TimeHour, ":", TimeMinute, ":", TimeSecond;
FullDate       = DateFullYear, "-", DateMonth "-", DateMday;
FullTime       = PartialTime, TimeOffset;

(* from:
XML Schema Part 2: Datatypes Second Edition
W3C Recommendation 28 October 2004
changes: included EBNF for Whitespace
*)
Base64Binary   = [WhiteSpace], [ {B64S, B64S, B64S, B64S},
                                ( (B64S, B64S, B64S, B64) | (B64S, B64S, B16S, "=") | (B64S, B04S,
"=", [WhiteSpace], "=") ) ],
                                 [WhiteSpace];

B64S                  = B64, [WhiteSpace];
B16S                  = B16, [WhiteSpace];
B04S                  = B04, [WhiteSpace];
B04                           =  "A" | "Q" | "g" | "w";
B16                           =  "A" | "E" | "I" | "M" | "Q" | "U" | "Y" | "c" | "g" |
                                 "k" | "o" | "s" | "w" | "0" | "4" | "8";
B64                           = UppercaseLetter | LowercaseLetter | Digit | "+" | "/";

WhiteSpace            = WhiteSpaceChar, {WhiteSpaceChar};
WhiteSpaceChar = TAB | CR | LF | " ";
```

Message Types

```
XDOPDetailedMessage    = ActionRequest | ActionResponse |
                         CallbackRequest | CallbackResponse |
                         DeviceDescriptionRequest | DeviceDescriptionResponse |
                         ServiceListDescriptionRequest | ServiceListDescriptionResponse |
                         ServiceDescriptionRequest | ServiceDescriptionResponse |
                         ClientDescriptionRequest | CientDescriptionResponse |
                         SetDeviceTypeRequest | SetDeviceTypeRequest |
                         Phase0Request | Phase0Response |
                         Phase1Request | Phase1Response |
                         SetDataFormatRequest | SetDataFormatResponse |
                         QueryRequest | QueryResponse |
                         UpdateRequest | UpdateResponse |
                         EventSubscribe | EventSubscribeResponse |
                         EventUnsubscribe | EventUnsubscribeResponse |
                         EventMessage | RejectedMessage | ResetConnectionMessage;

ActionRequest  = "øA", ActionIndex, [ "s", ServiceIndex ],
                 { "øV", XDOPEscapedValue }, MessageEnd;
ActionResponse = "øA", ActionIndex, [ "s", ServiceIndex ],
                 ( { "øV", XDOPEscapedValue } | ErrorContent ) , MessageEnd;

CallbackRequest        = "øC", CallbackIndex, [ "s", ServiceIndex ],
                         { "øV", XDOPEscapedValue }, MessageEnd;
CallbackResponse       = "øC", CallbackIndex, [ "s", ServiceIndex ],
                         ( { "øV", XDOPEscapedValue } | ErrorContent ) , MessageEnd;

DeviceDescriptionRequest      = "øD0", MessageEnd;
DeviceDescriptionResponse     = "øD0", ( DeviceDescription | ErrorContent ), MessageEnd;

SetDeviceTypeRequest   = "øD1", deviceTypeString, MessageEnd;
SetDeviceTypeResponse  = "øD1", [ ErrorContent ], MessageEnd;

ServiceListDescriptionRequest = "øD2", [ "s", ServiceIndex ], MessageEnd;
ServiceListDescriptionResponse = "øD2", ( ServiceListDescription | ErrorContent ), MessageEnd;

ServiceDescriptionRequest     = "øD3", [ "d", DeviceIndex ], MessageEnd;
ServiceDescriptionResponse    = "øD3", ( ServiceDescription | ErrorContent ), MessageEnd;

SetDataFormatsDescriptionRequest      = "øD4" { "øG", GroupIndex, ":", FormatString }, MessageEnd;
SetDataFormatsDescriptionResponse     = "øD4", [ ErrorContent ], MessageEnd;

ClientDescriptionRequest      = "øD9", MessageEnd;
ClientDescriptionResponse     = "øD9", ( ClientDescription | ErrorContent ), MessageEnd;

Phase0Request  = "øP0", MessageEnd;
Phase0Response = "øP0", [ ErrorContent ], MessageEnd;

Phase1Request  = "øP1, MessageEnd;
Phase1Response = "øP1", [ErrorContent], MessageEnd;

QueryRequest   = "øQ", VariableIndex, [ "s", ServiceIndex ], MessageEnd;
QueryResponse  = "øQ", VariableIndex, [ "s", ServiceIndex ],
                 ( ( ":", XDOPEscapedValue ) | ErrorContent ), MessageEnd;

UpdateRequest  = "øU", VariableIndex, [ "s", ServiceIndex ], ":",
                 XDOPEscapedValue, MessageEnd;
UpdateResponse = "øU", VariableIndex, [ "s", ServiceIndex ],
                 [ ErrorContent ], MessageEnd;

EventSubscribe =              "øS1", [ "s", ServiceIndex ], MessageEnd;
EventSubscribeResponse        = "øS1", [ "s", ServiceIndex ],
                                [ ErrorContent ], MessageEnd;


EventUnsubscribe      = "øS0", [ "s", ServiceIndex ], MessageEnd;
```

```
EventUnsubscribeResponse = "øS0", [ "s", ServiceIndex ],
                           [ ErrorContent ], MessageEnd;

EventMessage = "øE", EventKey, [ "s", ServiceIndex ],
               { "øV", VariableIndex, ":", XDOPEscapedValue }, MessageEnd;

ResetConnectionMessage = "øR0", MessageEnd;

 (* unrecognized or invalid root element, detected transport error *)
RejectedMessage = "øR1", ErrorContent, MessageEnd;

MessageEnd = { ExtensionElement }, EndTag;

ExtensionElement = Element;

FormatString = DecimalASCII | HexadecimalASCII | Bin_BaseX | Bin_Escaped | C99_Floats];

ActionIndex    = Index;
CallbackIndex  = Index;
ServiceIndex   = Index;
GroupIndex     = [ 0 | 1 | 2 | 3 ];
DeviceIndex    = Index;
VariableIndex  = Index;
EventKey = Digit;

ErrorContent = "øF0:", ErrorCode, [ "øF1:", ErrorDescription ];

ErrorCode = NonZeroDigit, Digit, Digit;
ErrorDescription = CharData;
```

Descriptions

Device Description

```
DeviceDescription = CRLF, DeviceDescriptionHeader;

DeviceDescriptionHeader =       "0:", [ "descriptionType" ], "=standard:1", CRLF,
                                "1:", [ "deviceType" ], "=", URN, CRLF,
                              [ "2:", [ ”deviceTypeList" ], "=", , CRLF, ]
                                                              {DeviceTypeItem},
                                "3:", [ "deviceId" ], "=", String, CRLF,
                                "4:", [ "manufacturer" ], "=", String, CRLF,
                              [ "5:", [ "manufacturerURL" ], "=", URI, CRLF ],
                                "6:", [ "productName" ], "=", String, CRLF,
                              [ "7:", [ "productURL" ], "=", URI, CRLF],
                              [ "8:", [ "serialNumber" ], "=", String, CRLF ],
                              [ "9:", [ "softwareVersion" ], "=", String, CRLF ],
                              [ "10:", [ "platformInfo" ], "=", String, CRLF ],
                              [ "11:", [ "tempBuildNumber" ], "=", string, CRLF ],
                              [ "12:", [ "formatSmallIntegers" ], "=", CRLF ],
                                                              {FormatItem},
                              [ "13:", [ "formatBigIntegers" ], "=", CRLF ],
                                                              {FormatItem},
                              [ "14:", [ "formatFloats" ], "=", CRLF ],
                                                              {FormatItem},
                              [ "15:", [ "formatBinary" ], "=", CRLF ],
                                                              {FormatItem},

DeviceTypeItem =              " 0:”, [ "deviceType” ]. "=”, URN, CRLF;

FormatItem = " 0", [ "format” ], "=”, String, CRLF ;

ItemName =     (UppercaseLetter | LowercaseLetter),
               {UppercaseLetter | LowercaseLetter | Digit | "_" };
```

ServiceList Description

```
ServiceListDescription = CRLF, ServiceListDescriptionHeader, { ServiceListItem };

ServiceListDescriptionHeader =  "0:", [ "descriptionType" ], "=standard:1", CRLF,
                                "1:", [ "serviceList" ], "=", CRLF,


ServiceListItem =       " 0:", [ "serviceIndex" ], "=", Index, CRLF,
                        " 1:", [ "serviceType" ], "=", URN, CRLF,
                        " 2:", [ "serviceId" ], "=", string, CRLF,
```

ClientDescription

```
"0:", ["descriptionType"], "=standard:1", CRLF,
"1",  ["manufacturer"], "=",  String, CRLF,
["2:", ["manufacturerURL"], "=", URI, CRLF,]
"3:", ["productName"], "=", String, CRLF,
["4:", ["productURL"], "=" String, CRLF,]
["5:", ["softwareVersion"], "=" String, CRLF,]
["6:", ["platformInfo"], "=", String, CRLF,]
["7:", ["tempBuildNumber"], "=", String, CLRF]
```

Service Description

```
ServiceDescription = CRLF, ServiceDescriptionHeader;

ServiceDescriptionHeader =      "0:", [ "descriptionType" ], "=standard:1", CRLF,
                                "1:", [ "serviceType" ], "=", URN, CRLF,
                                [ActionList],
                                [CallbackList]
                                [VariableList]
                                [DataTypeLinkList];

ActionList =                    "2:", [ "actionList" ], "=", CRLF,
                                ActionListItem, {ActionListItem};

ActionListItem =                " 0:", [ "actionIndex" ], "=", Index, CRLF,
                                " 1:", [ "name" ], "=", ItemName, CRLF,
                                [ArgumentList];
                                [" 3:", [ "shortInfo" ], "=", String, CRLF]

ArgumentList =                  " 2:", [ "argumentList" ], "=", CRLF,
                                ArgumentListItem, {ArgumentListItem};

ArgumentListItem =              "  0:", [ "name" ], "=", ItemName, CRLF,
                                "  1:", [ "direction" ], "=", ( "in" | "out" ), CRLF,
                                DataTypeInfo,
                                [  7:", [ "shortInfo" ], "=", String;

DataTypeInfo =                  SimpleType | ExtendedType | DataTypeStruct | DataTypeLink;

SimpleType =                    Indentation, "2:", [ "simpleType" ], "=", SimpleDataType, CRLF;

ExtendedType =                  Indentation, "3:", [ "extendedType" ], "=", CRLF,
                                ExtendedTypeValue

ExtenedTypeValue                Indentation, " 0:”, [ "simpleType” ], "=”, SimpleDataType, CRLF,
                                [ AllowedValueRange ],
                                [ enumList ],
                                [Indentation, " 3:”, [ "unit” ], "=”, string, CRLF ];
```

```
DataTypeStruct                  Indentation, "4:", [ "dataTypeStruct" ], "=", CRLF,
DataTypeStructValue;

DataTypeStructValue             Indentation, "0:", [ "name" ], "=", string, CRLF,
                                [Indentation, "1:", [ "simpleType" ], "=", SimpleDataType, CRLF, ]
                                [Indentation, "2:", [ "extendedType" ], "=", CRLF
                                ExtendedTypeValue]
                                [Indentation, "3:", [ "dataTypeStruct" ], "=", CRLF,
                                DataTypeStructValue]

Indentation                    {" "};(* Actual count of indent characters depends on location*)

DataTypeLink                    Indentation, "5:", [ "dataTypeLink" ], "=", String, CRLF;

CallbackList                    "3:", [ "callbackList" ], "=", CRLF,
                                ActionListItem, {ActionListItem};

VariableList =           "4:", [ " variableList " ], "=", CRLF,
                                VariableListItem, {VariableListItem};

VariableListItem =       " 0:", [ "variableIndex" ], "=", Index, CRLF,
                                " 1:", [ "name" ], "=", ItemName, CRLF,
                                " 2:", [ "flags" ], "=", [ "Q" ], [ "U" ], [ "E" ], CRLF,
                                (" 3:", [ "simpleType" ], "=", SimpleDataType, CRLF) |
                                (" 4:", [ "extendedType" ], "=", ExtendedTypeValue, CRLF) |
                                (" 5:", [ "dataTypeStruct" ], "=", DataTypeStructValue, CRLF) |
                                (" 6:", [ "dataTypeLink" ], "=", String, CRLF)
                                [" 7:", [ "shortInfo" ], "=", String, CRLF"]



DataTypeLinkList =              "5:", [ "dataTypeLinkList" ], "=", CRLF,
                                     DataType, {DataType};

DataType =                      " 0:", [ "name"], "=", String, CRLF,
                                [" 1:", [ "simpleType" ], "=", SimpleDataType, CRLF;]
                                [" 2:", [ "extendedType" ], "=", ExtendedTypeValue ],
                                [" 3:", [ "dataTypeStruct" ], "=", DataTypeStructValue ]

EnumList =               Indentation, "2:", [ "enumList" ], "=", CRLF,
                                EnumListItem, {EnumListItem};

EnumListItem =           Indentation, "0:", [ "nameValuePair" ], "=", String, CRLF;

AllowedValueRange =             Indentation, "1:", [ "allowedValueRange" ], "=", CRLF,
                                Indentation, " 0:", [ "minimum" ], "=", XDOPNumericValue, CRLF,
                                Indentation, " 1:", [ "maximum" ], "=", XDOPNumericValue, CRLF,
                               [Indentation, " 2:", [ "step" ], "=", XDOPNumericValue, CRLF];

SimpleDataType = "ui1", "ui2", "ui4", "ui8", "i1", "i2", "i4", "i8", "int", "r4", "r8", "float",
"char", "string", "date", "dateTime", "time", "boolean", "bin", "uri", "uuid", "xml",
"nameValueList";
```

URN

```
(* from RFC 2141 *)
(*
NID = Namespace Identifier
NSS = Namespace Specific String
*)
URN = "urn:", NID, ":", NSS;

NID = NIDRaw - "urn";

NIDRaw = LetDig, LetDigHyp, 30*[ LetDigHyp ];

LetDigHyp = LetDig | "-";

LetDig = UppercaseLetter | LowercaseLetter | Digit;

NSS = URNChar, {URNChar};

URNchar  = Trans | ( "%", HexDigit, HexDigit );

Trans    = LetDig | Other | Reserved;

Other    = "(" | ")" | "+" | "," | "-" | "." |
           ":" | "=" | "@" | ";" | "$" |
           "_" | "!" | "*" | "'";

Reserved = "%" | "/" | "?" | "#";
```

URI

```
(* from RFC 3986 *)
URI           = Scheme, ":", HierPart, [ "?", Query ], [ "#", Fragment ];
URIReference = URI | RelativeRef;
Absolute-URI  = Scheme, ":", HierPart, [ "?", Query ];

HierPart     = ( "//", Authority, PathAbEmpty ) | PathAbsolute | PathRootless | PathEmpty;
RelativeRef  = RelativePart, [ "?", Query ], [ "#", Fragment ];
RelativePart = ( "//", Authority, PathAbEmpty ) | PathAbsolute | PathNoScheme | PathEmpty;
Scheme                = Alpha {Alpha | Digit | "+" | "-" | "."};
Authority     = [ UserInfo, "@" ], Host, [ ":", Port ];
UserInfo      = { Unreserved | PctEncoded | SubDelims | ":" };
Host          = IPLiteral | IPv4Address | RegName;
Port                  = {Digit};

IPLiteral    = "[", ( IPv6Address | IPvFuture  ), "]";
IPvFuture     = "v", HexDigit, {HexDigit}, ".", IPvFuturePart, {IPvFuturePart};
IPvFuturePart  = Unreserved | SubDelims | ":";
IPv6Address   =                                          6*( H16, ":" ), Ls32 |
                                                  "::", 5*( H16, ":" ), Ls32 |
                         [                 H16 ], "::", 4*( H16, ":" ), Ls32 |
                         [   [ H16, ":" ], H16 ], "::", 3*( H16, ":" ), Ls32 |
                         [ 2*[ H16, ":" ], H16 ], "::", 2*( H16, ":" ), Ls32 |
                         [ 3*[ H16, ":" ], H16 ], "::",     H16, ":",   Ls32 |
                         [ 4*[ H16, ":" ], H16 ], "::",                 Ls32 |
                         [ 5*[ H16, ":" ], H16 ], "::",                 H16  |
                         [ 6*[ H16, ":" ], H16 ], "::";
H16           = HexDigit, 3*[HexDigit];
Ls32          = ( H16 ":" H16 ) | IPv4Address;
IPv4Address   = DecOctet, ".", DecOctet, ".", DecOctet, ".", DecOctet;
DecOctet     = Digit              |  (* 0-9 *)
                        NonZeroDigit, Digit |  (* 10-99 *)
                        "1", 2*Digit        |  (* 100-199 *)
                        "20", Digit         |  (* 200-209 *)
                        "21", Digit         |  (* 210-219 *)
                        "22", Digit         |  (* 220-229 *)
                        "23", Digit         |  (* 230-239 *)
                        "24", Digit         |  (* 240-249 *)
                        "250" | "251" | "252" | "253" | "254" | "255";
RegName      = { Unreserved | PctEncoded | SubDelims };
Path          = PathAbEmpty   | (* begins with "/" or is empty *)
                          PathAbsolute  | (* begins with "/" but not "//" *)
                          PathNoScheme  | (* begins with a non-colon segment *)
                          PathRootless  | (* begins with a segment *)
                          PathEmpty;      (* zero characters *)

PathAbEmpty  = { "/", Segment };
PathAbsolute = "/", [ SegmentNz, { "/", Segment } ];
PathNoScheme = SegmentNzNc, { "/", Segment };
PathRootless = SegmentNz, { "/", Segment };
PathEmpty    = ;
Segment       = {PChar};
SegmentNz     = PChar, {PChar};
SegmentNzNc   = (PCharNc), {PCharNc};
                        (* non-zero-length segment without any colon ":" *)
PChar         = Unreserved | PctEncoded | SubDelims | ":" | "@";
PCharNc       = PChar - ":";
Query         = { PChar | "/" | "?" };
Fragment      = { PChar | "/" | "?" };
PctEncoded    = "%", HexDigit, HexDigit;
Unreserved    = Alpha | Digit | "-" | "." | "_" | "~";
Reserved      = GenDelims | SubDelims;
GenDelims     = ":" | "/" | "?" | "#" | "[" / "]" | "@";
SubDelims     = "!" | "$" | "&" | "'" | "(" / ")" | "*" | "+" | "," | ";" | "=";
Alpha          = UppercaseLetter | LowercaseLetter;
```

Glossary

action
: Command exposed by a service. Takes one or more input or output arguments. For more information, see sections on Description and Control.

argument
: Parameter for action or callback exposed by a service. May be in xor out. For more information, see sections on Description and Control.

control point
: Retrieves device and service descriptions, sends actions to services, polls for service variables, and receives events from services.

device
: Logical device. A container. May implement other logical device interfaces. Embeds one or more services. For more information, see section on Description.

device description
: Formal definition of a logical device. For more information, see section on Description.

device type
: Standard device types are denoted by working committees. Vendors may specify additional device types. These are denoted by urn:*domain-name*:device: followed by a unique name assigned by a working committee or a vendor, where *domain-name* is a domain name registered to the vendor. For more information, see section on Description.

event
: Notification of one or more changes in variables exposed by a service. For more information, see section on Eventing.

publisher
: Source of event messages. Typically a device's service. For more information, see section on Eventing.

root device
: A logical device that is not embedded in any other logical device. For more information, see section on Description.

service
: Logical functional unit. Smallest units of control. Exposes actions and models the state of a physical device with variables. For more information, see section on Control.

service description
: Formal definition of a logical service. For more information, see section on Description.

service type
: Standard service types are denoted by working committees. Vendors may specify additional services. These are denoted by urn:*domain-name*:service: followed by a unique name assigned by a working committee or a vendor, colon, and a version number, where *domain-name* is a domain name registered to a working committee or a vendor. For more information, see section on Description.

variable
: Single facet of a model of a physical service. Exposed by a service. Has a name, data type, optional constraints values, and may trigger events when its value changes. For more information, see sections on Description and Control.

subscriber
: Recipient of event messages. Typically a control point. For more information, see section on Eventing.