XDOP™ Device Architecture 1.0

(IndeXed Device Object Protocol)

Version 1.0.3, 10. November 2006

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 a successor of the Simple UPnP (Universal Plug and Play) Proxy Protocol (SUPP). XDOP is derived from UPnP but is not only aligned to UPnP proxies but also suitable to many other device applications, where a universal, simple and lightweight device communication protocol is needed. Two new main application areas are the Universal Serial Bus (USB) and ZigBee. Details about USB and ZigBee can be found in companion documents.

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 text-based and human-readable (no binary data!)
- 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 send a text argument (input argument I1 of action A1) to a display service in a device:

```
*A1s1d1*I1:Hello world!*Z
```

The service s1 in device d1 displays the message “Hello world!” and replies to the control point:

```
*A1s1d1*Z
```

XDOP and Streams

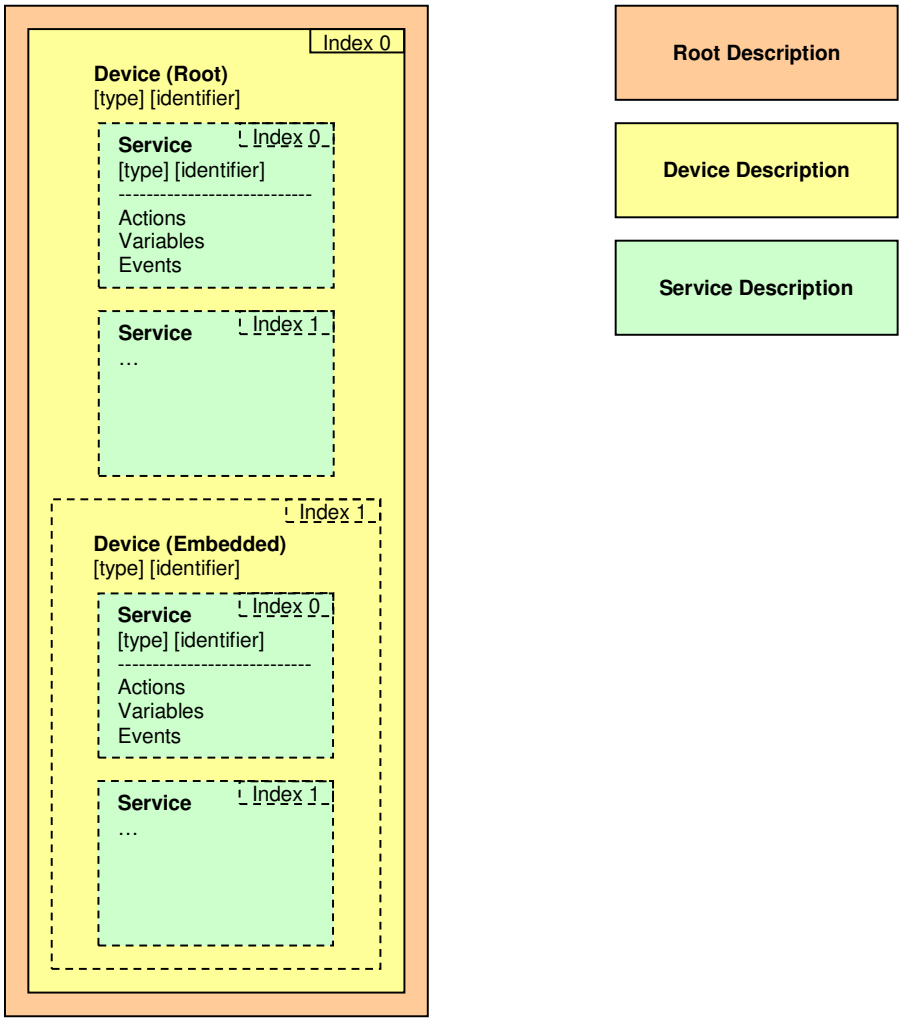
Control and eventing are not enough in many applications. Data streams like audio and video streams are not the scope of XDOP, but XDOP can be used to handle these streams, and XDOP can coexist in a separate logical or physical communication channel.

Device Model

The device model is very similar to the UPnP device model. There are root devices and embedded devices, which include one or more services.

Note that a single physical device may include multiple logical devices. Multiple logical devices can be modeled as a single root device with embedded devices (and services) or as multiple root devices (perhaps with no embedded devices). In the former case, there is one XDOP device description for the root device, and that device description contains a description for all embedded devices. In the latter case, there are multiple XDOP device descriptions, one for each root device.

The following diagram shows the relations between devices (root and embedded) 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 character string.

Every XDOP device has always a root device with a recommended index 0. There is no functional difference between a root and an embedded device. The terms root and embedded are meant only for structure purposes. All other objects except the root device are optional (broken lines in the diagram). In most cases there will be a root device and one or more services.

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. The root device can contain one or more embedded devices, which again can contain further nested embedded devices.

Service object

The service object models a functional unit of a device. Variables can be queried to get a view of the current state. Actions can be invoked to control the device. Besides the polling of the device status also the sending of change events can be activated

Messages

As mentioned above XDOP messages consist of pure 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 = "*Z";
Element = "*", Name, {Attribute}, [":", CharData];
Name = (UppercaseLetter - "Z"), Index;
Index = "0" | NonZeroDigit, {Digit};
Attribute = LowercaseLetter, Index;
CharData = {EscSeq | Char | AllowedAsteriskToken};

(* Asterisks in character data have to be replaced with "*X", if they are followed by an
uppercase letter *)
Char = UnicodeChar - "*";
EscSeq = "*X";
AllowedAsteriskToken = "*", ( UnicodeChar - UppercaseLetter );
```

An essential part of the message format is the combination of an asterisk with an uppercase letter. An XDOP message starts always with such a character pair. The pair “*Z” marks always the end of a message. The pair “*X” is reserved for the escape sequence of the asterisk character. Every asterisk character before an uppercase letter has to be escaped, if this asterisk 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

RootDescriptionRequest	*D0…
DeviceDescriptionRequest	*D1…
ServiceDescriptionRequest	*D2…
ActionRequest	*A…
QueryRequest	*Q…
UpdateRequest	*U…
EventSubscribe	*S1…
EventUnsubscribe	*S0…

Response

RootDescriptionResponse	*D0…
DeviceDescriptionResponse	*D1…
ServiceDescriptionResponse	*D2…
ActionResponse	*A…
QueryResponse	*Q…
UpdateResponse	*U…
EventSubscribeResponse	*S1…
EventUnsubscribeResponse	*S0…

Event

EventMessage	*E…

Other

ResetConnectionMessage	*R0…
RejectedMessage	*R1…

All messages are described under the following topics beside the last two one. These two messages are special messages which are only used in special situations:

The first message “RejectedMessage” is used by the device, if it detects an unrecognized or invalid root element or a transport error. The second message “ResetConnectionMessage” should be sent by the control point after a receive timeout, to reset and purge the device communication stack. It should always be sent at the beginning of a communication session.

Description

Although a root device without any services is allowed, in practice a minimal XDOP device description consists of a root device, which has one service, and this service has one variable. A service has at least one variable but no actions.

XDOP does not use the XML format for the description but simplified indexed tables.

The description begins with an empty 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: Root description

```
empty line
0:descriptionType=simple
1:majorVersion=1
2:minorVersion=0
3:maxRequestLength=maximum length of requests the device can handle
4:rootDevice=
 0:X_DeviceIndex=device index
 1:deviceType=urn:domain-name:device:deviceType:v
 2:UDN=uuid:UUID
 3:deviceList=
  0:X_DeviceIndex=device index
  1:deviceType=urn:domain-name:device:deviceType:v
  2:UDN=uuid:UUID
```

descriptionType
: Required. Type of the Descriptions provided by the device. Must be ”simple”.

majorVersion
: Required. Major version of the root and device description. Must be 1.

minorVersion
: Required. Minor version of the root and device description. Must be 0.

maxRequestLength
: Optional. Maximum message length the device can handle in bytes.

rootDevice
: Required. Information about the root device. Contains the following sub elements:

  X_DeviceIndex
  : Required. Defines the XDOP mapping index. Used to identify this device by index.

  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.

  UDN
  : Required. Unique Device Name. Universally-unique identifier for the device, whether root or embedded. Must be the same over time for a specific device instance (i.e., must survive reboots). Must begin with uuid: followed by a UUID suffix specified by a working committee or a vendor.

  deviceList
  : Required if and only if the device contains embedded devices.

Description: Device description

```
empty line
0:X_DeviceIndex=device index
1:deviceType=urn:domain-name:device:deviceType:v
2:UDN=uuid:UUID
3:friendlyName=short user-friendly title
4:manufacturer=manufacturer name
5:manufacturerURL=URL to manufacturer site
6:modelName=model name
7:modelNumber=model number
8:modelDescription=long user-friendly title
9:modelURL=URL to model site
10:serialNumber=manufacturer's serial number
11:EPC=Electronic Product Code
12:serviceList=
 0:X_ServiceIndex=service index
 1:serviceType=urn:domain-name:service:serviceType:v
 2:serviceId=urn:domain-name:serviceId:serviceID
 3:UTV=uuid:UUID
 VendorExtensionItem
VendorExtensionItem
```

X_DeviceIndex
: Required. Defines the XDOP mapping index. Used to identify this device by index.

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.

UDN
: Required. Unique Device Name. Universally-unique identifier for the device, whether root or embedded. Must be the same over time for a specific device instance (i.e., must survive reboots). Must begin with uuid: followed by a UUID suffix specified by a vendor.

friendlyName
: Required. Short description for end user. Should be localized (cf. ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers). Specified by a vendor. String. Should be < 64 characters.

manufacturer
: Required. Manufacturer's name. May be localized (cf. ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers). Specified by a vendor. String. Should be < 64 characters.

manufacturerURL
: Optional. Web site for Manufacturer. May be localized (cf. ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers). May be relative to base URL. Specified by a vendor. Single URL.

modelName
: Required. Model name. May be localized (cf. ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers). String. Should be < 32 characters.

modelNumber
: Recommended. Model number. May be localized (cf. ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers). String. Should be < 32 characters.

modelDescription
: Recommended. Long description for end user. Should be localized (cf. ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers). String. Should be < 128 characters.

modelURL
: Optional. Web site for model. May be localized (cf. ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers).

serialNumber
: Recommended. Serial number. May be localized (cf. ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers). String. Should be < 64 characters.

EPC
: Optional. Electronic Product Code. Code that identifies the consumer package. Single EPC.

serviceList
: Optional. Contains the following sub elements:

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

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.

serviceId

Required. Service identifier. Must be unique within this device description. Must begin with urn:, followed by a domain name owned by a working committee or a vendor, followed by :**serviceId**:, followed by a service ID suffix, i.e., urn:*domain-name*:**serviceId**:*serviceID*. Period characters in the domain name must be replaced with hyphens in accordance with RFC 2141.

The service ID suffix must be <= 64 characters.

UTV

Optional. Unique Type Variant. Universally-unique identifier for the variant of the service type. If two service descriptions for the same service type differ in one of the following points, they must have a different UTV:

- index mapping
- optional elements

This can be used to cache service descriptions, even if they are from different devices or have different service Ids. Even if the rest of the service descriptions is identical, the UTV may differ.

Must begin with uuid: followed by a UUID suffix specified by a vendor.

Description: Service description

The description for a service defines actions and their arguments, and variables and their data types, ranges, and event characteristics.

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. If an action has one or more output arguments, one of these arguments may be marked as a return value.

Each service must have one or more variables.

In addition to defining non-standard services, vendors may add actions and services to standard devices.

```
empty line
0:X_ServiceIndex=service index
1:serviceType=urn:domain-name:service:serviceType:v
2:serviceId=urn:domain-name:serviceId:serviceID
3:UTV=uuid:UUID
4:majorVersion=1
5:minorVersion=0
6:actionList=
 0:X_ActionIndex=action index
 1:name=action name
 2:argumentList=
  0:X_ArgumentIndex=argument index
  1:name=argument index
  2:direction=in xor out
  3:retval=
  4:relatedVariable=variableName
  VendorExtensionItem
  0:X_ArgumentIndex=argument index
  1:name=argument index
  2:direction=in xor out
  5:dataType=argument data type
  VendorExtensionItem
 VendorExtensionItem
7:variableList=
 0:X_VariableIndex=variable index
 1:name=variable name
 2:dataType=variable data type
 3:defaultValue=default value
 4:flags=any combination of the letters Q (Query), U (Update) or E (Event)
 5:allowedValueList=
  0:allowedValue=enumerated value
  VendorExtensionItem
 VendorExtensionItem
 0:X_VariableIndex=variable index
 1:name=variable name
 2:dataType=variable data type
 3:defaultValue=default value
 4:flags=any combination of the letters Q (Query), U (Update) or E (Event)
 6:allowedValueRange=
  0:minimum=minimum value
  1:maximum=maximum value
  2:step=increment value
  VendorExtensionItem
 VendorExtensionItem
VendorExtensionItem
```

X_ServiceIndex
Required. Defines the XDOP mapping index. Used to identify this service by index.
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.
serviceId
Required. Service identifier. Must be unique within this device description. Must begin with urn:, followed by a domain name owned by a working committee or a vendor, followed by :**serviceId**:, followed by a service ID suffix, i.e., urn:*domain-name*:**serviceId**:*serviceID*. Period characters in the domain name must be replaced with hyphens in accordance with RFC 2141.
The service ID suffix must be <= 64 characters.
majorVersion
Required. Major version of the service description. Must be 1.
minorVersion
Required. Minor version of the service description. Must be 0.
actionList
Required if and only if the service has actions. (Each service may have >= 0 actions.) Contains the following sub elements:

X_ActionIndex
Required. Defines the XDOP mapping index. Used to identify this action by index.
name
Required. Name of action.
argumentList
Required if and only if parameters are defined for action. (Each action may have >= 0 parameters.) Contains the following sub elements:

X_ArgumentIndex
Required. Defines the XDOP mapping index. Used to identify this action by index.
name
Required. Name of argument. May only contain USASCII letters (A-Z, a-z), USASCII digits (0-9), and underscores ("_").
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.
retval
Optional. Identifies at most one out argument as the return value. If included, must be the first out argument. (Element only; no value.)
relatedVariable
Optional. Must be the name of a variable. Case Sensitive. Defines the type of the argument.
dataType
Optional. See below for details. Either “relatedVariable” or “dataType” must be given.
variableList
Required. (Each service must have > 0 variables.) Contains the following sub elements:

X_VariableIndex
Required. Defines the XDOP mapping index. Used to identify this variable by index.
name
Required. Name of variable. May only contain USASCII letters (A-Z, a-z), USASCII digits (0-9), and underscores ("_").
dataType
Required. See below for details.
defaultValue
Recommended. Expected, initial value. Must satisfy allowedValueList or allowedValueRange constraints.
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
allowedValueList
Recommended. Enumerates legal string values. Prohibited for data types other than string. At most one of allowedValueRange and allowedValueList may be specified. Sub elements are ordered (e.g., see NEXT_STRING_BOUNDED). Contains the following sub elements:

allowedValue
Required. A legal value for a string variable. Must be < 32 characters.
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 allowedValueList may be specified. Contains the following sub elements:

minimum
: Required. Inclusive lower bound.

maximum
: Required. Inclusive upper bound.

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

dataType 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

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.

int (Integer)
: 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.

number (Float)
: Same as r8.

fixed.14.4 (Fixed14_4)
: Same as r8 but no more than 14 digits to the left of the decimal point and no more than 4 to the right.

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.

dateTime (DateTime)
: Date in a subset of ISO 8601 format with time but no time zone.

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

time (Time)
: Time in a subset of ISO 8601 format with no date and no time zone.

time.tz (Time_TZ)
: Time in a subset of ISO 8601 format with time zone but no date.

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

bin.base64 (Base64Binary)
: MIME-style Base64 encoded binary BLOB. Takes 3 Bytes, splits them into 4 parts, and maps each 6 bit piece to an octet. (3 octets are encoded as 4.) No limit on size.

bin.hex (BinHex)
: Hexadecimal digits representing octets. Treats each nibble as a hex digit and encodes as a separate Byte. (1 octet is encoded as 2.) No limit on size.

uri (URI)
: Universal Resource Identifier.

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

The **relatedVariable** element of an **argument** definition must be the name of a variable defined in the same service description. **relatedVariable** defines the *type* of the argument; there is not necessarily any semantic relationship between an argument and the relatedVariable used to define its type. **relatedVariable** must specify the **name** of a **variable** in the **variableList** which has the same **dataType**, **allowedValueList**, and **allowedValueRange** as the argument.

The **allowedValueList** and **allowedValueRange** elements may be used to indicate optional device capabilities. Working committees may require all values in the list or range to be supported by all vendors (no options), require a minimum subset with additional values being optional, or allow vendors to entirely decide which portions or the list or range to support. Vendors may add additional, vendor-specific values to the **allowedValueList** by using the "X_" prefix on the vendor-defined **allowedValue**s, if permitted by working committees. However, it should be noted that greater flexibility in optional capabilities reduces the number of values that control points can depend on to be present, with corresponding impacts on interoperability. 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, because the latter requires cancellation of advertisements and readvertisement each time the service description document is changed. If the service description is used to convey capabilities information, the device must omit from the service description any optional elements (actions, allowedValues, etc.) that are not implemented.

Description: Retrieving a description

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

Root Description

Description request:

```
*D0*Z
```

Description response:

```
*D0:
root description
*Z
```

Error response:

```
*D0
        *F0:error code
        *F1:error description
*Z
```

Device Description

Description request:

```
*D1ddevice index*Z
```

Description response:

```
*D1ddevice index:
device description
*Z
```

Error response:

```
*D1ddevice index
        *F0:error code
        *F1:error description
*Z
```

Service Description

Description request:

```
*D2sservice indexddevice index*Z
```

Description response:

```
*D2sservice indexddevice index:
service description
*Z
```

Error response:

```
*D2sservice indexddevice index
        *F0:error code
        *F1:error description
*Z
```

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 an asterisk character ('*') followed by an upper case letter. The root element has two attributes for the service index and the device index. These attributes are optional and the default for both indexes 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.

An asterisk character in the element value has to escaped with ***X**.

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

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

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!

```
*Aaction indexsservice indexddevice index
       *Iargument index:in arg value
       other in args and their values go here, if any
*Z
```

Shortest possible action without arguments:

```
*Aaction index*Z
```

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

```
*A0s1d2*I0:1.0*I1:Hello world!*Z
```

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

```
*A1*I1:23*Z
```

Control: Action: Response

Normal response:

```
*Aaction indexsservice indexddevice index
       *Oargument index:out arg value
       other out args and their values go here, if any
*Z
```

Error response:

```
*Aaction indexsservice indexddevice index
       *F0:error code
       *F1:error description
*Z
```

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

```
*Qvariable indexsservice indexddevice index*Z
```

Control: Query: Response

Normal response:

```
*Qvariable indexsservice indexddevice index:variable value*Z
```

Error response:

```
*Qvariable indexsservice indexddevice index
       *F0:error code
       *F1:error description
*Z
```

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

```
*S1sservice indexddevice index*Z
```

Normal response:

```
*S1sservice indexddevice index*Z
```

Error response:

```
*S1sservice indexddevice index
        *F0:error code
        *F1:error description
*Z
```

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

```
*S0sservice indexddevice index*Z
```

Normal response:

```
*S0sservice indexddevice index*Z
```

Error response:

```
*S0sservice indexddevice index
       *F0:error code
       *F1:error description
*Z
```

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.

```
*Eevent keysservice indexddevice index
        *Vvariable index:variable value
        other variables and their values go here, if any
*Z
```

Error Codes

errorCode	errorDescription	Description
401	Invalid Action Index	No action by that index at this service.
402	Invalid Args	Could be any of the following: not enough in args, too many in args, no in arg by that index, one or more in args are of the wrong data type.
403	*(Do Not Use)*	*(This code has been deprecated.)*
501	Action Failed	May be returned in current state of service prevents invoking that action.
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.**
602	Optional Action Not Implemented	The requested action is optional and is not implemented by the device.
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.
600-699	*TBD*	Common action errors. Defined by UPnP Forum Technical Committee.
700-799	*TBD*	Action-specific errors for standard actions. Defined by UPnP Forum working committee.
800-899	*TBD*	Action-specific errors for non-standard actions. Defined by UPnP vendor.
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
453	Invalid Device Index	
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 = "*Z";
Element = "*", Name, {Attribute}, [":", CharData];
Name = (UppercaseLetter - "Z"), Index;
Index = "0" | NonZeroDigit, {Digit};
Attribute = LowercaseLetter, Index;
CharData = {EscSeq | Char | AllowedAsteriskToken};

(* Asterisks in character data have to be replaced with "*X", if they are followed by an
uppercase letter *)
Char = UnicodeChar - "*";
EscSeq = "*X";
AllowedAsteriskToken = "*", ( UnicodeChar - UppercaseLetter );
```

Data Types

```
XDOPNumericValue =     UnsignedInteger | Integer | Float | Fixed14_4;

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

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

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

Fixed14_4 = 14*[Digit], [ "." , 4*[Digit] ]; (* fixed.14.4 *)

Date                  = FullDate; (* date *)
DateTime              = FullDate "T" PartialTime; (* dateTime *)
DateTime_TZ           = FullDate "T" FullTime; (* dateTime.tz *)
Time                  = PartialTime; (* time *)
Time_TZ               = FullTime; (* time.tz *)

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 |
                         RootDescriptionRequest | RootDescriptionResponse |
                         DeviceDescriptionRequest | DeviceDescriptionResponse |
                         ServiceDescriptionRequest | ServiceDescriptionResponse |
                         QueryRequest | QueryResponse |
                         UpdateRequest | UpdateResponse |
                         EventSubscribe | EventSubscribeResponse |
                         EventUnsubscribe | EventUnsubscribeResponse |
                         EventMessage | RejectedMessage | ResetConnectionMessage;

ActionRequest  = "*A", ActionIndex, ["s", ServiceIndex], ["d", DeviceIndex],
                 {"*I", ArgIndex, ":", XDOPEscapedValue}, MessageEnd;
ActionResponse = "*A", ActionIndex, ["s", ServiceIndex], ["d", DeviceIndex],
                 ( {"*O", ArgIndex, ":", XDOPEscapedValue} | ErrorContent) , MessageEnd;

RootDescriptionRequest = "*D0", MessageEnd;
RootDescriptionResponse       = "*D0", ( RootDescription | ErrorContent ), MessageEnd;

DeviceDescriptionRequest      = "*D1", ["d", DeviceIndex], MessageEnd;
DeviceDescriptionResponse     = "*D1", ( DeviceDescription | ErrorContent ), MessageEnd;

ServiceDescriptionRequest     = "*D2", ["s", ServiceIndex], ["d", DeviceIndex], MessageEnd;
ServiceDescriptionResponse    = "*D2", ( ServiceDescription | ErrorContent ), MessageEnd;

QueryRequest   = "*Q", VariableIndex, ["s", ServiceIndex], ["d", DeviceIndex], MessageEnd;
QueryResponse  = "*Q", VariableIndex, ["s", ServiceIndex], ["d", DeviceIndex],
                 ( (":", XDOPEscapedValue) | ErrorContent), MessageEnd;

UpdateRequest  = "*U", VariableIndex, ["s", ServiceIndex], ["d", DeviceIndex], ":",
                 XDOPEscapedValue, MessageEnd;
UpdateResponse = "*U", VariableIndex, ["s", ServiceIndex], ["d", DeviceIndex],
                 [ErrorContent], MessageEnd;

EventSubscribe = "*S1", ["s", ServiceIndex], ["d", DeviceIndex], MessageEnd;
EventSubscribeResponse = "*S1", ["s", ServiceIndex], ["d", DeviceIndex],
                         [ErrorContent], MessageEnd;

EventUnsubscribe       = "*S0", ["s", ServiceIndex], ["d", DeviceIndex], MessageEnd;
EventUnsubscribeResponse = "*S0", ["s", ServiceIndex], ["d", DeviceIndex],
                           [ErrorContent], MessageEnd;

EventMessage = "*E", EventKey, ["s", ServiceIndex], ["d", DeviceIndex],
               {"*V", VariableIndex, ":", XDOPEscapedValue}, MessageEnd;

ResetConnectionMessage = "*R0", MessageEnd;

 (* unrecognized or invalid root element, detected transport error *)
RejectedMessage = "*R1", ErrorContent, MessageEnd;

MessageEnd = {ExtensionElement}, EndTag;

ExtensionElement = Element;

ActionIndex    = Index;
ServiceIndex   = Index;
DeviceIndex    = Index;
ArgIndex       = Index;
VariableIndex  = Index;
EventKey = Digit;


ErrorContent = "*F0:", ErrorCode, [ "*F1:", ErrorDescription ];

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

Descriptions

Root Description

```
SimpleRootDescription = CRLF, RootDescriptionHeader, DeviceListItem;

RootDescriptionHeader =  "0:", [ "descriptionType" ], "=simple", CRLF,
                         "1:", [ "majorVersion" ], "=", Index, CRLF,
                         "2:", [ "minorVersion" ], "=", Index, CRLF,
                       [ "3:", [ "maxRequestLength" ], "=", Index, CRLF ],
                         "4:", [ "rootDevice" ], "=", CRLF;


DeviceListItem =         {" "}, " 0:", [ "X_DeviceIndex" ], "=", Index, CRLF,
                         {" "}, " 1:", [ "deviceType" ], "=", URN, CRLF,
                         {" "}, " 2:", [ "UDN" ], "=", UDN, CRLF,
                       [ {" "}, " 3:", [ "deviceList" ], "=", CRLF ,
                         DeviceListItem, {DeviceListItem} ];


UDN = "uuid:", TimeLow, ["-"], TimeMid, ["-"], TimeHighAndVersion, ["-"], ClockSeqAndReserved,
ClockSeqLow ["-"], Node;

UTV = "uuid:", TimeLow, ["-"], TimeMid, ["-"], TimeHighAndVersion, ["-"], ClockSeqAndReserved,
ClockSeqLow ["-"], Node;
```

Device Description

```
SimpleDeviceDescription = CRLF, DeviceDescriptionHeader, ServiceList, {VendorExtensionItem};

DeviceDescriptionHeader =       "0:", [ "X_DeviceIndex" ], "=", Index, CRLF,
                                "1:", [ "deviceType" ], "=", URN, CRLF,
                                "2:", [ "UDN" ], "=", UDN, CRLF,
                                "3:", [ "friendlyName" ], "=", String, CRLF,
                                "4:", [ "manufacturer" ], "=", String, CRLF,
                              [ "5:", [ "manufacturerURL" ], "=", URI, CRLF ],
                                "6:", [ "modelName" ], "=", String, CRLF,
                              [ "7:", [ "modelNumber" ], "=", String, CRLF ],
                              [ "8:", [ "modelDescription" ], "=", String, CRLF ],
                              [ "9:", [ "modelURL" ], "=", URI, CRLF],
                              [ "10:", [ "serialNumber" ], "=", String, CRLF ],
                              [ "11:", [ "EPC" ], "=", EPC, CRLF ];

ServiceList =                   "12:", [ "serviceList" ], "=", CRLF,
                                                          {ServiceListItem};

ServiceListItem =               " 0:", [ "X_ServiceIndex" ], "=", Index, CRLF,
                                " 1:", [ "serviceType" ], "=", URN, CRLF,
                                " 2:", [ "serviceId" ], "=", URN, CRLF,
                              [ " 3:", [ "UTV" ], "=", UTV, CRLF ],
                                " ", VendorIndex, ":", [ ItemName ], "=", String, CRLF;

VendorExtensionItem =                VendorIndex, ":", [ ItemName ], "=", String, CRLF;

VendorIndex = "1", Digit, Digit; (* Numbers 100-199 *)

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


EPC = EPCGID-URI | SGTIN-URI | SSCC-URI | SGLN-qURI | GRAI-URI | GIAI-URI | TagURI;
```

Service Description

```
SimpleServiceDescription = CRLF, ServiceDescriptionHeader, {VendorExtensionItem};

ServiceDescriptionHeader =      "0:", [ "X_ServiceIndex" ], "=", Index, CRLF,
                                "1:", [ "serviceType" ], "=", URN, CRLF,
                                "2:", [ "serviceId" ], "=", URN, CRLF,
                              [ "3:", [ "UTV" ], "=", UTV, CRLF ],
                                "4:", [ "majorVersion" ], "=", Index, CRLF,
                                "5:", [ "minorVersion" ], "=", Index, CRLF,
                                [ActionList],
                                [VariableList];

ActionList =                    "6:", [ "actionList" ], "=", CRLF,
                                ActionListItem, {ActionListItem};

ActionListItem =                " 0:", [ "X_ActionIndex" ], "=", Index, CRLF,
                                " 1:", [ "name" ], "=", ItemName, CRLF,
                                [ArgumentList],
                               {" ", VendorExtensionItem};

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

ArgumentListItem =              "  0:", [ "X_ArgumentIndex" ], "=", Index, CRLF,
                                "  1:", [ "name" ], "=", ItemName, CRLF,
                                "  2:", [ "direction" ], "=", ( "in" | "out" ), CRLF,
                              [ "  3:", [ "retval" ], "=", CRLF,],
                                DataTypeInfo,
                               {"  ", VendorExtensionItem};

DataTypeInfo =                  RelatedVariable | DataType;

RelatedVariable =               "  4:", [ "relatedVariable" ], "=", ItemName, CRLF;

DataType =                      "  5:", [ "dataType" ], "=", SimpleDataType, CRLF;

VariableList =          "7:", [ " variableList " ], "=", CRLF,
                                VariableListItem, {VariableListItem};

VariableListItem =      " 0:", [ "X_VariableIndex" ], "=", Index, CRLF,
                                " 1:", [ "name" ], "=", ItemName, CRLF,
                                " 2:", [ "dataType" ], "=", SimpleDataType, CRLF,
                              [ " 3:", [ "defaultValue" ], "=", XDOPValue, CRLF],
                                " 4:", [ "flags" ], "=", [ "Q" ], [ "U" ], [ "E" ], CRLF,
                                [AllowedValueList | AllowedValueRange];
                               {" ", VendorExtensionItem};

AllowedValueList =              " 5:", [ "allowedValueList" ], "=", CRLF,
                                AllowedValueListItem, {AllowedValueListItem};

AllowedValueListItem =          "  0:", [ "allowedValue" ], "=", String, CRLF,
                               {"  ", VendorExtensionItem};

AllowedValueRange =             " 6:", [ "allowedValueRange" ], "=", CRLF,
                                "  0:", [ "minimum" ], "=", XDOPNumericValue, CRLF,
                                "  1:", [ "maximum" ], "=", XDOPNumericValue, CRLF,
                              [ "  2:", [ "step" ], "=", XDOPNumericValue, CRLF],
                               {"  ", VendorExtensionItem};

VendorExtensionItem =  VendorIndex, ":", [ ItemName ], "=", Value, CRLF;

SimpleDataType = "ui1", "ui2", "ui4", "i1", "i2", "i4", "int", "r4", "r8", "number", "fixed14.4",
               "float", "char", "string", "date", "dateTime", "dateTime.tz", "time", "time.tz",
               "boolean", "bin.base64", "bin.hex", "uri", "uuid";
```

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;
```

EPC

```
(*
from:
EPC™ Tag Data Standards Version 1.1 Rev.1.24
Standard Specification
01 April 2004

changes: "Raw Tag URI" and "EPC Pattern URI" are not allowed and have been removed

*)

(* 4.3 Syntax

The syntax of the EPC-URI and the URI forms for related data types are defined by the
following grammar.
*)

(* 4.3.1 Common Grammar Elements *)

NumericComponent = ZeroComponent | NonZeroComponent;
ZeroComponent = "0";
NonZeroComponent = NonZeroDigit, {Digit};

PaddedNumericComponent = Digit, {Digit};

(* 4.3.2 EPCGID-URI *)

EPCGID-URI = "urn:epc:id: gid:", 2*(NumericComponent, "."), NumericComponent;

4.3.3 SGTIN-URI

SGTIN-URI = "urn:epc:id:sgtin:", SGTINURIBody;
SGTINURIBody = 2*(PaddedNumericComponent "."), NumericComponent;

The number of characters in the two PaddedNumericComponent fields must total 13
(not including any of the dot characters).

(* 4.3.4 SSCC-URI *)

SSCC-URI = "urn:epc:id:sscc:", SSCCURIBody;
SSCCURIBody = PaddedNumericComponent, ".", PaddedNumericComponent;

(*
The number of characters in the two PaddedNumericComponent fields must total 17
(not including any of the dot characters).
*)

(* 4.3.5 SGLN-URI *)

SGLN-qURI = "urn:epc:id:sgln:", SGLNURIBody;
SGLNURIBody = 2*(PaddedNumericComponent, "."), NumericComponent;

(*
The number of characters in the two PaddedNumericComponent fields must total 12
(not including any of the dot characters).
*)

(* 4.3.6 GRAI-URI *)

GRAI-URI = "urn:epc:id:grai:", GRAIURIBody;
GRAIURIBody = 2*(PaddedNumericComponent, "."), NumericComponent;

(*
The number of characters in the two PaddedNumericComponent fields must total 12
(not including any of the dot characters).
*)

(* 4.3.7 GIAI-URI *)
```

```
GIAI-URI = "urn:epc:id:giai:", GIAIURIBody;
GIAIURIBody = PaddedNumericComponent, ".", PaddedNumericComponent;

(*
The number of characters in the two PaddedNumericComponent fields must not
exceed 30 (not including any of the dot characters).
*)

(* 4.3.8 EPC Tag URI *)

TagURI = "urn:epc:tag:", TagURIBody;
TagURIBody = GIDTagURIBody | SGTINSGLNGRAITagURIBody | SSCCTagURIBody | GIAITagURIBody;
GIDTagURIBody = GIDTagEncName, ":", 2*(NumericComponent, "."), NumericComponent;
GIDTagEncName = "gid-96";
SGTINSGLNGRAITagURIBody = SGTINSGLNGRAITagEncName, ":", NumericComponent, ".",
    2*(PaddedNumericComponent, "."), NumericComponent;
SGTINSGLNGRAITagEncName = "sgtin-96" | "sgtin-64" | "sgln-96" |
                          "sgln-64"  | "grai-96"  | "grai-64";
SSCCGIAITagURIBody = SSCCGIAITagEncName, ":", NumericComponent, 2*(".", PaddedNumericComponent);
SSCCGIAITagEncName = "sscc-96" | "sscc-64" | "giai-96" | "giai-64";
```

Glossary

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

argument
: Parameter for action 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 embed other logical devices. 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 default value, 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.