INTERNATIONAL STANDARD

ISO/IEC 24752-2

FIrst edition
2008-02-15

Information technology — User interfaces — Universal remote console —

Part 2: User interface socket description

Technologies de l'information — Interfaces utilisateur — Console à distance universelle —

Partie 2: Description de "Socket" d'Interface utilisateur

Reference number
ISO/IEC 24752-2:2008(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 24752-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 35, *User interfaces*.

ISO/IEC 24752 consists of the following parts, under the general title *Information technology — User interfaces — Universal remote console*:

— *Part 1: Framework*

— *Part 2: User interface socket description*

— *Part 3: Presentation template*

— *Part 4: Target description*

— *Part 5: Resource description*

Introduction

A user interface socket is an abstract concept that, when implemented, exposes the functionality and state of a target in a machine-interpretable manner. A user interface socket is independent of any specific implementation platform.

A user interface socket contains variables, constants, commands and notifications. The variables include all of the dynamic data a user can perceive and/or manipulate, and may also include additional dynamic supporting data that is not presented to the user. Example variables include the volume of a television, the current floor of an elevator, or an internal variable representing the current state of a transaction that is used to control dynamic features of the interface. Constants represent fixed or constant information known before runtime which may be presented to the user, such as a model number. A command is a core function that a user can request a target to perform and that cannot be represented by a variable. The commands include all target functions that can be called by users. Examples include the 'search' command of an airline reservation system or the 'seek' command of a CD player. A user interface socket does not include commands for accessing the values of the variables. There are typically no commands that simply change the values of variables. An exception would be a 'reset' operation which puts the target into a specific state. The notifications are special states where normal operation is suspended such as an exception state. Notifications are special states triggered by the target. Examples include an announcement made by a public address system in an airport, a clock alarm, or a response to invalid input for a field of a form.

A user interface socket specification is an XML document that uses the constructs defined in this part of ISO/IEC 24752 to describe a user interface socket.

An example user interface socket description is included as Annex B.

NOTE Additional information is needed before the socket can be presented to a user, including natural language labels and help text associated with the elements of the user interface. This information is provided externally to the socket description. Resources reference socket elements using the socket's name (as given in the socket descriptions about URI, see 6.2) and the element ids (see sections 7.2, 10.2 and 11.2). Refer to ISO/IEC 24752-5 for further details.

Information technology — User interfaces — Universal remote console —

Part 2:
User interface socket description

1 Scope

ISO/IEC 24752 is a multi-part International Standard to facilitate operation of information and electronic products through remote and alternative interfaces and intelligent agents.

A user interface socket is an abstract concept that describes the functionality and state of a device or service (target) in a machine interpretable manner. This part of ISO/IEC 24752 defines an extensible Markup Language (XML) based language for describing a user interface socket. The purpose of the user interface socket is to expose the relevant information about a target so that a user can perceive its state and operate it. This includes data presented to the user, variables that can be manipulated by the user, commands that the user can activate, and exceptions that the user is notified about. The user interface socket specification is applicable to the construction or customization of user interfaces.

2 Conformance

An XML file conforms to this part of ISO/IEC 24752 (i.e. is a user interface socket description) if

- it has the MIME type specified in 5.3, if applicable, and
- its root element is the <uiSocket> element as defined in Clause 6.

An XML file does not conform to this part of ISO/IEC 24752 if it uses any element, attribute or value that is not part of this specification.

NOTE 1 Target manufacturers who want to add information to a socket description beyond the elements, attributes and values specified in this document can do so by externally providing (proprietary) resource descriptions that point into the structure of a socket description. Refer to ISO/IEC 24752-5 for details.

NOTE 2 Future versions of this part of ISO/IEC 24752 might add new elements, attributes and values. Also, future versions might drop the policy of strict language conformance in favor of allowing for language extensions. Therefore, URC manufacturers are encouraged to implement their URCs so that unrecognized markup is ignored without failing.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 24752-1, *Information technology — User interfaces — Universal remote console — Part 1: Framework*

ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

ISO/IEC 14977:1996, *Information technology — Syntactic metalanguage — Extended BNF*

ISO 15836:2003, *Information and documentation — The Dublin Core metadata element set*

DCMI Metadata Terms, http://dublincore.org/documents/dcmi-terms/

IETF RFC 2046, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, November 1996, http://www.ietf.org/rfc/rfc2046.txt

IETF RFC 3986, Uniform Resource Identifier (URI): Generic Syntax, January 2005, http://www.ietf.org/rfc/rfc3986.txt

W3C Recommendation: Extensible Markup Language (XML) 1.0 (Third Edition), 04 February 2004, http://www.w3.org/TR/2004/REC-xml-20040204/

W3C Recommendation: Namespaces in XML, World Wide Web Consortium, 14 January 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114/

W3C Recommendation: XML Path Language (XPath) 2.0, W3C Recommendation 23 January 2007, http://www.w3.org/TR/2007/REC-xpath20-20070123/

W3C Recommendation: XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Recommendation 23 January 2007, http://www.w3.org/TR/2007/REC-xpath-functions-20070123/

W3C Recommendation: XML Schema Part 1: Structures Second Edition, W3C Recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

W3C Recommendation: XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 24752-1 and the following apply.

4.1
context element
element to which a dependency pertains

4.2
dependency
expression that defines a relationship between a property of a socket variable, constant, command or notify element and the values of other socket elements

4.3
notify
socket element that represents a notification

4.4
uniform resource identifier
URI
name or address that refers to a resource, as defined in IETF RFC 3986

5 Relation to other standards

5.1 Relation to XML

This specification defines an extensible Markup Language (XML) based language. Markup in XML is case sensitive.

Tag names, and attribute names and values are not localizable, i.e. they are identical for all international languages. However, the text content between tags can be language specific. As with all XML based languages, white space characters immediately surrounding tags are non-significant.

This specification makes use of the XML namespaces concept to enable the import of element and attribute names defined elsewhere.

All element and attribute names used in this document with no namespace prefix are defined by this International Standard and are part of the namespace with URI reference http://myurc.org/ns/uisocketdesc. If not defined as the default namespace, the namespace identifier 'uis' should be used.

Throughout this document, the following namespace prefixes and corresponding namespace identifiers are used for referencing foreign namespaces:

- xsd: The XML Schema namespace (http://www.w3.org/2001/XMLSchema);
- dc: The Dublin Core Metadata Element Set namespace (http://purl.org/dc/elements/1.1/) (Element Set defined by ISO 15836);
- dcterms: The DCMI Metadata Terms namespace (http://purl.org/dc/terms);
- xsi: The XML Schema Instance namespace (http://www.w3.org/2001/XMLSchema-instance).

For an XML Schema definition for the user interface socket description see Annex A.

5.2 XPath expressions

5.2.1 General

This specification uses XML Path Language (XPath) Version 2.0 for addressing elements within the socket. Specifically, XPath is used in describing dependencies between the elements of the socket.

XPath 2.0 syntax is used without XPath 1.0 compatibility.

5.2.2 Use of XPath 2.0 syntax and semantics

This International Standard uses the syntax and semantics of XPath 2.0, with the following additions and exceptions:

- The XPath expressions shall be coded in UCS.
- The static expression context (see section 2.1.1 in XPath 2.0) shall be initialized with the following components:
    - “XPath 1.0 compatibility mode” shall be false.
    - The “statically known namespaces” are the namespace declarations that are in scope for the XML element that contains the XPath expression.

- The "default element/type namespace" shall be the null namespace (which refers to types that are defined in the socket description, see 12).

- The "default function namespace" shall be the standard function namespace of XPath 2.0: http://www.w3.org/2005/xpath-functions.

- The "in-scope schema definitions" shall only contain "in-scope schema types" with the following content: All types of namespace http://www.w3.org/2001/XMLSchema, as specified in section 2.5.1 of XPath 2.0; and the local types defined in a socket description's <schema> part (see 12).

  NOTE 1 XPath 2.0 adds the following pre-defined types to the pre-defined types of XML Schema Definition Part 2: xsd:untyped, xsd:untypedAtomic, xsd:anyAtomicType, xsd:dayTimeDuration, xsd:yearMonthDuration.

- The "in-scope variables" shall be empty.

- The "function signatures" shall be the functions of the namespace http://www.w3.org/2005/xpath-functions, as defined in XQuery 1.0 and XPath 2.0 Functions and Operators, with exceptions as specified in 5.2.4; the constructor functions for all the atomic types in the "in-scope schema definitions"; and the additional functions defined in 5.2.5.

NOTE 2 The following components of the XPath 2.0 static expression context are not used in this International Standard: "context item static type", "statically known collations", "default collation", "base URI", "statically known documents", "statically known collections", "statically known default collection type".

- The dynamic expression context (see section 2.1.2 in XPath 2.0) shall be initialized with the following components:

  - The "context item" shall be the socket set or element that the XPath expression is specified for as dependency.

  - The "variable values" shall be empty.

  - The "function implementations" shall include implementations of the functions of the namespace http://www.w3.org/2005/xpath-functions, as defined in XQuery 1.0 and XPath 2.0 Functions and Operators; the constructor functions for all the atomic types in the "in-scope schema definitions"; and the additional functions as defined in 5.2.5.

  - The "current dateTime" shall be the current time with local timezone of the URC, represented as a value of type xsd:dateTime.

  - The "implicit timezone" shall be the local timezone of the URC.

  NOTE 3 The following components of the XPath 2.0 dynamic expression context are not used in this International Standard: "context item", "context position", "context size", "Available documents", "Available collections", "Default collection".

- There is no Data Model (XDM instance). Expressions and functions that refer to a data model instance shall not be used in socket descriptions. The context item expression (see section 3.1.4 in XPath 2.0) shall not be used. Path expressions (see section 3.2 in XPath 2.0) shall not be used. Node operations such as node comparison (see section 3.5.3 in XPath 2.0), and the union, intersect and except operators (see section 3.3.3 in XPath 2.0) shall not be used.

- The evaluation of logical expressions (AND/OR) shall be strictly from left to right, and shall not evaluate the right operand if the result is already determined by the left operand. I.e. with an expression of the form "A and B", B shall not be evaluated if A is false; and in the case of "A or B", B

shall not be evaluated if A is true. In addition, Boolean operations shall respect the “undefined” value (see 5.2.3).

- The XPath 2.0 implementation shall be based on XML 1.0 and Namespaces in XML.
- The XPath 2.0 implementation may support the namespace axis.

5.2.3 The undefined value

This International Standard adds the “undefined” value as a special value for all types from XPath 2.0 (see 5.2.2) and locally defined types (see 12).

If any part of an XPath expression is undefined, the whole expression shall be undefined. This rule shall not apply, if, based on evaluation logic, the result of an expression is determined without evaluating any undefined part of it.

EXAMPLE The following expression will never evaluate to an undefined result. It yields true if the element with id ‘myvar’ is available and has the value 4, otherwise it yields false.

```
uis:hasDefinedValue(‘myvar’) and uis:value(‘myvar’) eq 4
```

NOTE Implementations may vary as long as the described effect is warranted. For example, an error exception could be internally raised to signal that an XPath expression yields “undefined”.

5.2.4 XPath functions

The following XPath functions may be used:

- Functions of the namespace http://www.w3.org/2005/xpath-functions, as defined in XQuery 1.0 and XPath 2.0 Functions and Operators
- The constructor functions for all atomic types in the “in-scope schema definitions”

with the following exceptions:

- The function string() shall only be used with one argument.
- The function resolve-uri() shall not be used.
- The functions related to QName (section 11 of XQuery 1.0 and XPath 2.0 Functions and Operators), operators on NOTATION (section 13) and Functions and Operators on Nodes (section 14) shall not be used.
- The following context functions (section 13 of XQuery 1.0 and XPath 2.0 Functions and Operators) shall not be used: position, last, default-collation, static-base-uri.

The XPath 2.0 specific rules for implicit conversion between types apply.

5.2.5 Additional functions

5.2.5.1 General

This International Standard defines the following additional functions that may be used in expressing socket dependencies.

NOTE These functions are defined in the namespace http://myurc.org/ns/uisocketdesc. A namespace prefix for this namespace (e.g. “uis”) needs to be declared on any one of the XML elements containing the XPath expression. Note that the namespace prefix for http://myurc.org/ns/uisocketdesc must always be used for these functions since the default

function namespace is http://www.w3.org/2005/xpath-functions (XPath 2.0 function namespace). Using the 'xmlns' attribute to declare a default XML namespace does not change the default function namespace.

5.2.5.2 xsd:anyType uis:value(string path)

This is a function which takes as its argument the path of a socket variable, constant, command, notification or an indexed component of it, and returns the current value of that variable, command or notification (component).

NOTE 1 The type of the return value of uis:value() function is the same as the type of the socket element referred to by the argument 'path'. Note that, although the static return type is xsd:anyType, the dynamic type of the return value is always a more specific one (derived from the static type); it can be queried by the boolean operator 'instance of'.

The following syntax is defined for path (in Extended BNF notation, see ISO/IEC 14977):

path = absolutePath | relativePath | shortcutPath ;

absolutePath = "/" , { setPath , "/" } elementPath ;

relativePath = ("./" | ("../" , { "../" })) , { setPath , "/" } elementPath ;

shortcutPath = elementPath ;

setPath = *setId* , { "[" , *setIndex* "]" } ;

elementPath = normalElementPath | basicCommandPath | timedCommandPath ;

normalElementPath = *elementId* , { "[" , *elementIndex* , "]" } ;

basicCommandPath = *elementId* , { "[" , *elementIndex* , "]" , } ["[state]"] ;

timedCommandPath = *elementId* , { "[" , *elementIndex* , "]" } ["[state]" | "[timeToComplete]"] ;

A path shall be an absolute path, a relative path or a shortcut path.

An absolute path is a slash-separated list of set ids and an element id, walking the path from the root element <uiSocket> (see 6.1) down to a socket element, with indices in square brackets wherever a set or the element has dimensions. An absolute path starts with a slash character ("/") which stands for the root element.

A relative path has as context item (node) the socket element or set that the dependency is defined on. Relative paths that start with "./" have their context item as starting point for the subsequent path. Relative paths that start with "../" refer to the parent <set> of their context item as first segment in the path. Every subsequent "../" in the path refers to the next parent toward the root. No indices shall be specified for any dimensional set or element that is referred to by "./" or "../". At runtime, the missing indices (starting from the root) will be taken from the set/element owning the dependency, if not otherwise specified in the relative path. That means that relative paths occurring on dimensional sets or elements are inherently referencing elements with the same set of indices or a subset thereof (i.e. they are referring to the same "slice" of components).

A shortcut path omits set ids, and uses only an element's id and indices, if any. It does not have a leading slash character ("/"). A shortcut path shall not be used if the path includes a dimensional set.

setId is a placeholder for the 'id' attribute of a <set> element (which is an ancestor of the requested socket element). For dimensional <set> elements (i.e. <set> elements with a non-empty 'dim' attribute) *setIndex* is a placeholder for an index value of a <set> element, with the index value being compatible to the pertaining index type. <set> elements with no dimension have no *setIndex*.

elementId is a placeholder for the 'id' attribute of a <variable>, <constant>, <command> or <notify> element. For dimensional elements, an *elementIndex* shall be used as a placeholder for an index value of the element,

with the index value being compatible to the pertaining index type. Elements with no dimension and <constant> elements have no *elementIndex*.

For <command> elements of type uis:basicCommand or uis:timedCommand, a selector may be added in square brackets at the end of the path. Allowed is for uis:basicCommand: "[state]"; and for uis:timedCommand: "[state]" or "[timeToComplete]". If the selector is missing for those types, "[state]" is assumed.

The path argument shall be a string. The following characters shall be escaped if used as part of *setIndex* or *elementIndex,* as follows. '^' shall be used as the escape character.

- "[" shall be coded as "^["
- "]" shall be coded as "^]"
- "^" shall be coded as "^^"

NOTE 2 For hard-coded paths (i.e. the path is known at authoring time), the author can use a string literal that is enclosed in single (') or double (") quotes. For paths that are computed at runtime (e.g. when referencing variables as index values), the author can use valid XPath string operations (see 5.2) to concatenate a viable path string. See examples below.

The return value is the current value of the specified socket element (or its component if it is a dimensional element or has a dimensional set as ancestor). For command types that don't have state information (uis:voidCommand), an empty string is returned. For a command of type uis:basicCommand or uis:timedCommand and an *elementIndex* of "state" or empty, the state (as string) of the command or its component is returned (see 10.3). For commands of type uis:timedCommand and an *elementIndex* of "timeToComplete", the time to complete (as string in the xsd:duration format) of the command or its component is returned if it is currently defined, otherwise an empty string is returned. For a notification its state (as string) or the state of its component is returned (valid return values are "active", "inactive" and "stacked").

uis:value(string path) shall evaluate to an undefined result for socket elements (or their components) that have an undefined value/state. In this case the whole expression (of which uis:value(path) is part of) shall have an undefined result.

EXAMPLE 1 A variable of type xsd:string with id "var" is nested inside two sets with ids "outerSet" and "innerSet". Neither the variable nor any of the nesting sets are dimensional. One can retrieve its value by either one of the following XPath expressions:

uis:value("/outerSet/innerSet/var")
uis:value("var")

EXAMPLE 2 A constant of type xsd:string with id "const" is nested inside two sets with ids "outerSet" and "innerSet". Neither the constant nor any of the nesting sets are dimensional. (In fact, a constant can never be dimensional). One can retrieve its value by either one of the following XPath expressions:

uis:value("/outerSet/innerSet/const")
uis:value("const")

EXAMPLE 3 A command of type uis:timedCommand with id "cmd" is nested in a set with id "setId". One can retrieve its state by either one of the following XPath expressions:

uis:value("/setId/cmd[state]")
uis:value("/setId/cmd")
uis:value("cmd[state]")
uis:value("cmd")

And one can retrieve the value of its timeToComplete component by either one of the following XPath expressions:

uis:value("/setId/cmd[timeToComplete]")
uis:value("cmd[timeToComplete]")

EXAMPLE 4 A notification with id "notifyId" is nested in a set with id "setId". One can retrieve its state by either one the following XPath expressions:

uis:value("/setId/notifyId")
uis:value("notifyId")

EXAMPLE 5 A variable of type xsd:string with id “var” has one dimension with index type xsd:string. It is nested within a non-dimensional set element with id “setId”.
One can retrieve the value of the component with index “alpha” by either one of the following XPath expressions:
uis:value(“/setId/var[alpha]”)
uis:value(“var[alpha]”)
And the value of the component with index “3*[2^3]” by either one of the following XPath expressions (note that “[“, “^” and “]” need to be escaped):
uis:value(“/setId/var[3*^[2^^3^]]”)
uis:value(“var[3*^[2^^3^]]”)

EXAMPLE 6 Same as in example 5, but now retrieving the value of the component with an index value taken from another variable with id=”index”:
uis:value(concat(“/setId/var[“, uis:value(“index”), ”]”))
uis:value(concat(“var[“, uis:value(“index”), ”]”))

EXAMPLE 7 A command of type uis:timedCommand with id “cmd” has one dimension with index type xsd:integer. It is nested within a non-dimensional set element with id “setId”. One can retrieve the timeToComplete field of the command with index 0 by either one of the following XPath expressions:
uis:value(“/setId/cmd[0][timeToComplete]”)
uis:value(“cmd[0][timeToComplete]”)

EXAMPLE 8 A variable of type xsd:string with id “var” has two dimensions with index types xsd:integer and xsd:string. It is nested within a non-dimensional set element with id “setId”. One can retrieve the value of the component with indices “3” and “none” by either one of the following XPath expressions:
uis:value(“/setId/var[3][none]”)
uis:value(“var[3][none]”)

EXAMPLE 9 A variable of type xsd:string with id “var” has two dimensions with index types xsd:integer and xsd:string. It is nested within a 1-dimensional outer set element with id “outerSet” and index type xsd:boolean, and a non-dimensional inner set element with id “innerSet”. One can retrieve the value of the component with indices “true” (for the outerSet element), “3” and “none” (for the var element) by the following XPath expression:
uis:value(“/outerSet[true]/innerSet/var[3][none]”)

EXAMPLE 10 Same as in example 9, but now using the values of three other variables (with ids “index1”, “index2” and “index3) as index values:
uis:value(concat(“/outerSet[“, uis:value(“index1”), ”]/innerSet/var[“, uis:value(“index2”), ”][“, uis:value(“index3”), ”]”))

EXAMPLE 11 This example illustrates the use of relative paths. The following minimal socket defines a 1-dimensional set with an index type of xsd:integer. For every set instance, a “power” variable and a “dimmer” variable is defined. The “dimmer” value can only be changed if the power of the pertinent light is on. (The relative path “../power” in the write dependency of the dimmer variable refers to the “power” variable, with the index for the parent set being the same as for the “dimmer” variable that contains the write dependency.)

```
<uiSocket name=”http://example.com/lights/socket” id=”socket”

    xmlns="http://myurc.org/ns/uisocketdesc" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <set id=”lights” dim=”xsd:integer”>
        <variable id=”power” type=”xsd:boolean” />
        <variable id=”dimmer” type=”dimmerType”>
            <dependency>
                <write> value(“../power”) </write>
            </dependency>
        </variable>
    </set>
    <xsd:schema>
        <xsd:simpleType name=”dimmerType” id=”dimmerTypeId>
            <xsd:restriction base=”xsd:integer”>
                <xsd:minInclusive value=”0” />
                <xsd:maxInclusive value=”10” />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:schema>
</uiSocket>
```

EXAMPLE 12 This example uses a relative path inside the 'relevant' dependency of a set. For each set instance, one variable instance (with id="isRelevant") inside the set instance is controlling whether the set instance is to be presented to the user or not.

```
<uiSocket name="http://example.com/relevantset/socket" id="socket"
    xmlns="http://myurc.org/ns/uisocketdesc" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <set id="set1" dim="xsd:integer">
        <variable id="isRelevant" type="xsd:boolean" />
        <variable id="someData" type="xsd:string" />
        <dependency>
            <relevant> value("./isRelevant") </relevant>
        </dependency>
    </set>
</uiSocket>
```

5.2.5.3 boolean uis:hasDefinedValue(string path)

This is a function which takes as its argument the path of a socket variable, constant or command and returns "true" if the current value of that variable or constant, or the current state of that command is defined, otherwise "false". For command types that don't have state information (uis:voidCommand), "false" is returned.

The argument path denotes an XPath expression that shall evaluate to a string.

NOTE Since XPath defines the 'or' and 'and' operators so that the right operand is not evaluated if the left operand pre-determines the result, one can check for undefined values/states by calling uis:hasDefinedValue(id) before calling uis:value(id).

EXAMPLE The following expression will never evaluate to an undefined value. It yields true if the command with id 'reset' has an undefined state or the state 'succeeded'.

```
not(uis:hasDefinedValue('reset')) or uis:value('reset') eq 'succeeded'
```

5.2.5.4 boolean uis:isNotifyActive()

This is a Boolean function with no argument; returns true if any notify element in the socket is active, and false if no notify element is active.

NOTE The XPath expression "not(uis:isNotifyActive())" can be useful to check for normal mode (i.e. no notification is active).

5.2.5.5 boolean uis:sessionForward(string type, string uri)

This is a function that states session forwarding (see ISO/IEC 24752-1). The value of the type argument is either "destructive" or "spawn". The value of the uri argument is the name (URI) of the socket that the URC is forwarded to. The return value is "true" if the target has sent the specified session forward event to the URC, otherwise "false". This function can be used in postconditions of socket commands (see 10.10.6), to provide a hint to the URC that the command may trigger a session forwarding.

5.3 MIME type

A user interface socket description shall have a MIME type of "application/urc-uisocketdesc+xml", if applicable.

6 Structure of a socket description

6.1 General

A socket description is defined as an XML document with the following basic structure. A socket description document shall be coded in UCS according to ISO/IEC 10646.

The following sections describe the sets, variables, constants, commands and notifications in more detail.

EXAMPLE

```
<uiSocket

    about="http://example.com/thermometer/socket"
    id="socket"
    xmlns="http://myurc.org/ns/uisocketdesc"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:dcterms="http://purl.org/dc/terms/"
    <dcterms:conformsTo>http://myurc.org/iso24752-2/2007</dcterms:conformsTo>

    <!-- … variables, constants, commands and notify elements, optionally contained in set elements … -->
    <!-- … xsd type schema elements … -->

</uiSocket>
```

6.2 The ‘about’ attribute

The <uiSocket> element shall have an ‘about’ attribute. The ‘about’ attribute references the socket that is being described by the socket description. The value of the ‘about’ attribute is specified as element content and is the name of the socket expressed as a URI. The URI may or may not be resolvable.

NOTE 1 Target manufacturers are encouraged to make the socket descriptions of their products publicly available by posting the socket description at the socket's name URI.

Typically this URI is derived from the target's URI by concatenation.

EXAMPLE http://www.comp.com/target/socket if the target's URI is http://www.comp.com/target.

NOTE 2 The same URI is used as a socket reference in the socket description (‘about’ attribute of <uiSocket> element), and the name of the socket in the target description (name attribute of the corresponding <socket> element). Refer to ISO/IEC 24752-4 for details on the target description.

6.3 The ‘id’ attribute

The <uiSocket> element shall have an ‘id’ attribute that specifies an identifier that is unique among all ids in the socket description.

NOTE The identifier is used to specify external resources for the socket description root element, as exemplified in ISO/IEC 24752-5.

6.4 The ‘sufficient’ attribute

The <uiSocket> element may have a ‘sufficient’ attribute of type Boolean. Its default value is “false”.

The value "true" indicates that all commands in the socket description that have no ‘sufficient’ attribute themselves, are sufficiently specified as described in 10.5.

NOTE The ‘sufficient’ attribute for <uiSocket> is provided for the convenience of the author. It can be overwritten by separate ‘sufficient’ attributes on the individual <command> elements.

6.5 The ‘complete’ attribute

The <uiSocket> element may have a ‘complete’ attribute of type Boolean. Its default value is “false”.

The value "true" indicates that all commands in the socket description, that have no ‘complete’ attribute themselves, are completely specified as described in 10.6.

NOTE The ‘complete’ attribute for <uiSocket> is provided for the convenience of the author. It can be overwritten by separate ‘complete’ attributes on the individual <command> elements.

6.6 The <dcterms:conformsTo> element

The <uiSocket> element shall have a subelement <dcterms:conformsTo> that specifies a reference to an established standard to which the socket conforms. The value, a URI, is provided as element content. The value http://myurc.org/iso24752-2/2007 indicates that the described socket conforms to this International Standard.

EXAMPLE <dcterms:conformsTo>http://myurc.org/iso24752-2/2007</dcterms:conformsTo>

NOTE 1 The value of the <dcterms:conformsTo> element can be used when testing for conformance of a socket description.

NOTE 2 The <dcterms:conformsTo> element is taken from the set of Dublin Core Metadata Terms.

6.7 The <dcterms:modified> element

The <uiSocket> element may include a <dcterms:modified> element. This indicates that the socket description document has been modified from its original version, while still using the same reference socket URI in its ‘about’ attribute (see section 6.2). Its content is of type xsd:date or xsd:dateTime.

EXAMPLE <dcterms:modified>2004-01-30</dcterms:modified>

NOTE The <dcterms:modified> element is taken from the set of Dublin Core Metadata Terms.

A socket description should remain stable wherever possible. A socket description document that is changed shall be assigned a new about URI (as specified in 6.2) or <dcterms:modified> element. If a socket description has multiple versions with the same about URI (but different modification dates), elements that have changed should be assigned a new identifier (see sections 0, 10.2 and 11.2).

6.8 Socket description properties from DCMI

Any element and element refinement (except <dcterms:conformsTo> and <dcterms:modified> which are referenced above) from ISO 15836, Dublin Core Metadata Element Set, or the set of Dublin Core Metadata Initiative (DCMI) Metadata Terms may be used to describe a socket description, if appropriate. Each of them may occur multiple times within a <uiSocket> element.

In particular, the following Dublin Core Metadata Elements may be applied:

- <dc:creator> specifying the author of the socket description
- <dc:publisher> specifying the provider of the socket description
- <dc:contributor> specifying a co-author of the socket description
- <dcterms:hasVersion> specifying the name of an alternate socket for accessing the same or similar functionality

- <dcterms:isVersionOf> specifying the name of the primary socket among a set of alternate sockets for accessing the same or similar functionality

6.9 <variable>, <constant>, <command>, <notify> and <set> elements

The <uiSocket> element shall contain a complete set of <variable> (see 8), <constant> (see 9), <command> (see 10) and <notify> (see 11) elements capturing all of the information that can be perceived or manipulated by a user, and all of the commands available to a user of the target. These elements may be structured via the <set> element (see 7) which may be nested. An implicit ordering is implied by the order in which the elements appear within the socket description document.

6.10 XSD type schema elements

A socket description may contain an XSD type schema declaration. This is used to define specialized types for the socket variables as described in 12.

6.11 Platform-specific mapping information

The <mapping> element may be used any number of times inside a <uiSocket> element to include platform-specific mapping information pertaining to the socket.

A <mapping> element shall have a 'platform' attribute whose value is not restricted by this International Standard.

A <mapping> element may have arbitrary element content and subelements. However, subelements shall be from namespaces other than the uis namespace.

NOTE Socket descriptions that contain platform specific mapping information lose their platform neutrality. Although multiple mappings may be specified in a socket description (one for each platform) it is recommended to consider other mechanisms of specifying the binding to platform-specific technologies. For example, mapping information may be provided in an external file with references to the elements of the socket description.

7 Sets

7.1 General

Sets express an internal hierarchical structure of a user interface socket. Sets may be nested (i.e. a <set> element may appear within another <set> element).

NOTE The <set> element defined in this International Standard is intended to provide a hierarchical structure internal to a socket. "Grouping resources" (as defined in ISO/IEC 24752-5) can be used for specifying presentational grouping structures (which may be non-hierarchical), or presentational grouping can be contained in a user interface implementation description (UIID, see ISO/IEC 24752-1).

7.2 Attribute 'id'

A <set> element shall have an 'id' attribute.

The 'id' attribute shall be of type ID as defined by XML Schema Part 2: Datatypes. It provides an identifier which is used to refer to the element within the socket description and as a means of binding externally defined resources such as labels. The 'id' value shall be unique among all 'id' attributes within the socket description.

EXAMPLE <set id="mainSet">

NOTE 'id' attribute values are not normally presented to users and need not be human comprehensible.

7.3 Attribute ‘dim’

The ‘dim’ attribute specifies a set as dimensional (with one or more dimensions). At runtime, the set is instantiated once for every possible combination of its index values. This results in multiple values for every socket element that is (directly or indirectly) contained in the dimensional set.

NOTE The <insert> dependency specifies whether the URC can change the set of indices at runtime (see 7.4.5).

If dimensional sets are nested, the values for the socket elements multiply based on every possible combination of the total set of index values of all dimensional sets involved, plus the index values of the socket element itself, if it is dimensional.

NOTE 1 In other words, the set of values for a particular socket element results from the product of all index types that occur when walking the path from the <uiSocket> element down to the particular socket element (see definition of path in 5.2.5.2).

NOTE 2 Constants are not allowed to occur within a dimensional set (see 9.1).

NOTE 3 A dimensional set is also called a “repeating set”.

The ‘dim’ attribute may be present for <set> elements. If present, it shall contain a non-empty, ordered, space-separated list of type references that are the set’s index types. The first reference specifies the index type for the first dimension, the second type for the second dimension and so on. Valid index type references are: (a) the name of a type that is defined in the <schema> part of the socket description, or (b) the fully qualified name (QName) of an external type.

NOTE 3 A set may have an index of a type with an infinite set of possible index values. However, at runtime a finite subset of the index type values will be used as actual indices of the dimensional set.

EXAMPLE 1 A sound mixer application mixes two input streams into one output stream. Each stream has 2 audio settings (volume and loudness) and a “break” command that mutes the channel for 5 seconds.

```
<set id=”stream” dim=”streamType”>
    <variable id=”volume” type=”volumeType” />
    <variable id=”loudness” type=”xsd:boolean” />
    <command id=”break” type=”uis:timedCommand” />
</set>
    ….
    <xsd:schema>
        <xsd:simpleType name=”streamType”>
            <xsd:restriction base=”xsd:string”>
                <xsd:enumeration value=”Input1” />
                <xsd:enumeration value=”Input2” />
                <xsd:enumeration value=”Output” />
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name=”volumeType”>
            <xsd:restriction base=”xsd:integer”>
                <xsd:minInclusive value=”0” />
                <xsd:maxInclusive value=”100” />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:schema>
```

References (e.g. from resources) to dimensional sets are treated as references to all instantiations of the set.

EXAMPLE 2 A label resource applies to a dimensional set. The URC can use this label for the set of elements contained in it.

NOTE 4 The ‘dim’ attribute allows for repetition based on a set of indices that are known before runtime or at session opening. More dynamic applications that require a changing set of indices at runtime should use the “spawn forward” approach instead (see ISO/IEC 24752-1).

7.4 Set dependencies

7.4.1 General

A <dependency> element may be present within a <set> element. If present, it shall occur exactly once.

7.4.2 The 'id' attribute

Any of the subelements of <dependency> (see 7.4.3 through 7.4.4) may have an 'id' attribute.

If present, the 'id' attribute shall be of type ID as defined by XML Schema Part 2: Datatypes. It provides an identifier which is used to refer to the specific dependency within the socket description and as a means of binding externally defined resources such as labels. The 'id' value shall be unique among all 'id' attributes within the socket description.

7.4.3 The <relevant> dependency

A <relevant> dependency may be present within a <dependency> element on sets. If present, it shall occur exactly once.

The content of the <relevant> element shall be as specified for the <relevant> dependency of variables (see 8.11.3).

A set shall inherit its <relevant> dependency to its members (applies to sets, variables, commands and notifications), unless the member specifies a <relevant> dependency on its own.

NOTE Inheritance of dependencies provides a convenient way for authors to specify a dependency once (on the set level) that is common to all (or most) of its members.

7.4.4 The <write> dependency

A <write> dependency may be present within a <dependency> element on sets. If present, it shall occur exactly once.

The content of the <write> element shall be as specified for the <write> dependency of variables (see 8.11.3).

A set shall inherit its <write> dependency to its members (applies to sets, variables and commands), unless the member specifies a <write> dependency on its own.

7.4.5 The <insert> dependency

The <insert> dependency specifies whether the user can modify the pertaining set of valid index combinations at runtime (with the pertaining set being the indices specified by the 'dim' attribute on the corresponding <set> element).

The content of the <insert> dependency shall be an XPath expression that evaluates to a Boolean value which may change during a session. If it evaluates to false, then it is inappropriate to add or delete an index combination; otherwise it is appropriate to attempt to add or delete an index combination, although there is no guarantee that it will succeed.

NOTE "Adding an index combination" means to add a component of every socket element contained in the corresponding <set> element, for a particular new index combination within the index space defined by the corresponding 'dim' attribute. Note that the initial values of the new components are determined by the target. "Removing an index combination" means to delete a component of every socket element contained in the corresponding <set> element, for a particular index combination that is currently occurring.

A <insert> dependency may be present within a <dependency> element on dimensional sets (i.e. sets with 'dim' attribute, see 7.3). If present, it shall occur exactly once.

Its default value is "false()".

7.5 Platform-specific mapping information

The <mapping> element may be used any number of times to include platform-specific mapping information for the containing set.

A <mapping> element shall have a 'platform' attribute whose value is not restricted by this International Standard.

A <mapping> element may have arbitrary element content and subelements. However, subelements shall be from namespaces other than the uis namespace.

NOTE Socket descriptions that contain platform specific mapping information lose their platform neutrality. Although multiple mappings may be specified in a socket description (one for each platform) it is recommended to consider other mechanisms of specifying the binding to platform-specific technologies. For example, mapping information may be provided in an external file with references to the sets and elements of the socket description.

7.6 Set properties from DCMI

Any element and element refinement from ISO 15836, Dublin Core Metadata Element Set, or the set of Dublin Core Metadata Initiative (DCMI) Metadata Terms may be used to describe a <set> element, if appropriate. Each of them may occur multiple times within the <set> element.

7.7 Set members

A <set> element shall contain a non-empty set of <set>, <variable>, <constant>, <command> or <notify> elements as members.

8 Variables

8.1 General

Variables are used to expose the state of a target to a URC. The value of a variable may be displayed to the user of the URC, and the user may change the target's state by changing the variable's value.

A variable may occur as subelement of a <set> element (see 7) or as subelement of the <uiSocket> element (see 6.1).

A variable is defined with the markup:

<variable> … </variable>

EXAMPLE Examples of variables include the current channel showing on a television set, and a value (not shown to the user) indicating whether the current user has accepted the terms of a license.

8.2 The 'id' attribute

A <variable> element shall have an 'id' attribute.

The 'id' attribute shall be of type ID as defined by XML Schema Part 2: Datatypes. It provides an identifier which is used to refer to the element within the socket description and as a means of binding externally defined resources such as labels. The 'id' value shall be unique among all 'id' attributes within the socket description.

EXAMPLE id="tvVolume"

NOTE 'id' attribute values are not normally presented to users and need not be human comprehensible.

8.3 The ‘type’ attribute

8.3.1 General

Every <variable> element shall be assigned a ‘type’ attribute.

If the variable has a ‘dim’ attribute (see 8.9) the ‘type’ attribute specifies the type of any one of the variable’s values (“homogenous array”).

8.3.2 Simple types

Any simple type defined in XML Schema Part 2 shall be allowed as variable type, or any other simple type derived from one of these types (including derivation by union).

NOTE The usage of datatype declarations borrowed from XML Schema Part 2 does not imply that the values are coded as defined by XML Schema. This is implementation dependent and not defined by this International Standard. For example, a variable of type xsd:integer may be represented in the socket as a binary value, and not as a string.

8.3.3 Space-Separated Value Lists

Any type that has been derived by list according to XML Schema Part 2 is allowed. If used for a variable, a list type shall be defined in the type definition part of the socket description (see 12). Alternatively, the pre-defined type uis:stringList may be used if the set of string items is not restricted (see 12.4).

For the purpose of expressing dependencies in XPath 2.0 syntax (see 5.2), the value of a list variable shall be treated as a sequence of the atomic type from which it is derived (see 5.2.2).

8.3.4 Comma-Separated Value (CSV) Lists

The pre-defined type uis:csvlist is allowed for a list of string values, represented as concatenation of individual strings separated by single comma (‘,’) characters.

Escaping conventions use the backslash character, '\' (UTF-8 character code 0x5C), as follows: a backslash character (‘\’) is represented as ‘\\’ and a comma (‘,’) as ‘\,’ in individual string entries in CSV lists. Any occurring white space characters are construed as part of the string entries.

The empty string (“”) shall be interpreted as empty list (no entries).

For the purpose of expressing dependencies in XPath 2.0 syntax (see 5.2), the value of a uis:csvlist variable shall be treated as of type xsd:string (see 5.2.2).

EXAMPLE 1 “1,2,3” represents a list with 3 string entries: “1”, “2” and “3”.

EXAMPLE 2 “Smith\, Fred,Jones\, Davey” represents a list of 2 strings: “Smith, Fred” and “Jones, Davey”.

EXAMPLE 3 “ alpha, beta” represents a list of 2 strings, each with 2 leading spaces: “ alpha” and “ beta”.

EXAMPLE 4 “,,” represents a list of 3 empty string entries (each “”).

EXAMPLE 5 “” represents an empty list (no entries).

NOTE 1 The XPath 2.0 provided functions for string manipulations can be used to inspect values of comma-separated list variables, when used in dependency expressions. URC implementations may make the individual values of the comma-separated list available to UIIDs through its API, but this is implementation-specific.

NOTE 2 Type uis:csvlist is an alternative to uis:stringList (see 12.4).

NOTE 3 uis:csvlist is especially useful for UPnP AV environments where some values are conveyed as comma-separated value lists. Note that uis:csvlist, however, does allow for entries of type string only.

NOTE 4 In contrary to this International Standard, the UPnP AV specifications leave it open whether the empty string should be interpreted as an empty list or a list with one empty-string entry.

8.3.5 Stream types

For socket variables that convey streams of text, audio or video, the following pre-defined types may be used as values for the 'type' attribute:

- "uis:textStreamOut": Text stream flowing from the target to the URC (client).
- "uis:textStreamIn": Text stream flowing from the URC to the target.
- "uis:audioStreamOut": Audio stream flowing from the target to the URC.
- "uis:audioStreamIn": Audio stream flowing from the URC to the target.
- "uis:videoStreamOut": Video stream flowing from the target to the URC.
- "uis:videoStreamIn": Video stream flowing from the URC to the target.
- "uis:multiMediaStreamOut": Stream that holds a synchronized combination of video, audio and text flowing from the target to the URC.
- "uis:multiMediaStreamIn": Stream that holds a synchronized combination of video, audio and text flowing from the URC to the target.

The format and transmission protocol of any of these stream types is implementation specific, and may not be known before runtime. However, if the set of target-supported formats is known before runtime, the <dc:format> element from DCMI (see 8.14) should be used (any appropriate number of times) as subelement of <variable> to specify the available formats, each coded as MIME type (see IETF RFC 2046).

Also, if the stream is known to be specific to a natural language before runtime, the <dc:language> element from DCMI (see 8.14) should be used as subelement of <variable> to specify the language.

NOTE 1 It is recommended that implementations of sockets and TUNs allow for streaming format negotiation between a URC and a target at runtime.

NOTE 2 The presence of a stream typed variable in the socket description is for descriptive purposes and does not imply that the pertaining stream "runs through" a socket implementation at runtime. Implementations are free to handle streams in a way that is suitable for their specific requirements. For example, a socket implementation may help to set up a connection for streaming, with the actual stream bypassing the socket implementation for performance reasons.

8.3.6 Socket-defined types

Oftentimes the simple types (see 8.3.2) are not sufficient to express a variable's value space in terms of allowed and disallowed values. In these cases a variable's 'type' attribute may reference socket-defined types, i.e. types that are defined in a separate section of the socket description, as specified in section 11.11.The reference to a socket-defined type shall consist of the type's name, without any namespace prefix.

8.3.7 Imported types

Types defined in other namespaces may be referenced by their full qualified name (QName). In this case the namespace URI shall be fully specified, and the location of the pertinent XML schema definition (XSD) file should be given.

EXAMPLE This variable contains a description of the content of a content directory service. The description is of type DIDLType, as defined by the DIDL ("Digital Item Declaration Language") namespace urn:mpeg:mpeg21:2002:02-DIDL-NS. The pertinent XSD file is available at http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/did/didl.xsd.

```
<uiSocket
      about="http://example.com/cds"
      id="socket"
      xmlns="http://myurc.org/ns/uisocketdesc"
      xmlns:didl=”urn:mpeg:mpeg21:2002:02-DIDL-NS”
      xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
      xsi:schemaLocation=”urn:mpeg:mpeg21:2002:02-DIDL-NS
            http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-21_schema_files/did/didl.xsd” >
            <set id=”firstSet”>…
            <variable id=”content” type=”didl:DIDLType” />
      </set>
      …
</uiSocket>
```

NOTE The socket treats values of imported types internally as of type string, in particular when used inside XPath expressions (see 5.2) for dependencies. No checking against well-formedness or validity should be assumed. However, it may make values of imported types available in a special format through an API. For example, it may offer a DOM-based API for variables of imported types, but this is implementation-specific

8.4 The ‘secret’ attribute

<variable> elements may have a ‘secret’ attribute. It is a Boolean indicating whether a variable's value is sensitive regarding security and privacy, and requires protection to support the user's privacy.

The default value is "false".

EXAMPLE A password could be declared as:

```
<variable id="accountPassword" type="xsd:string" secret="true" />
```

If the variable has a ‘dim’ attribute (see 8.9) the ‘secret’ attribute applies to any one of the variable’s values.

8.5 The ‘sensitive’ attribute

A <variable> element may have a ‘sensitive’ attribute. It is a Boolean indicating whether the variable shall be presented to the user under all circumstances.

The default value is "false".

EXAMPLE A variable marked with sensitive=”true” could represent safety or warranty information that is to be presented to the user.

If the variable has a ‘dim’ attribute (see 8.9) the ‘sensitive’ attribute applies to any one of the variable’s values.

8.6 The ‘timeout’ attribute

A variable may have a ‘timeout’ attribute may be given, holding a Boolean value. The default value is "false". The value “true” shall only occur if the variable has type xsd:duration.

A socket variable that has timeout="true" is a socket timeout variable. A socket timeout variable holds the timeout duration for one or multiple user response timeouts of the target.

The type of a socket timeout variable should indicate the range of allowable values for the timeout (by using constraining facets, as described in 12.3), and the variable should allow adjustment up to 5 times the default value.

EXAMPLE A timeout variable might represent the amount of time that a volume indicator remains on a TV screen after the user has stopped adjusting it. When the volume disappears from the screen, this would not cause a notification to the URC user. The value of the timeout variable could be modified by the URC or its user, based on the user's preferences.

NOTE A notify element may also have a timeout variable attached as a subelement (see 11.9). This notify timeout variable follows the same syntax as socket timeout variables.

If the variable has a 'dim' attribute (see 8.9) the 'timeout' attribute applies to any one of the variable's values.

8.7 The 'optional' attribute

A variable may have an 'optional' attribute of type Boolean. Its default value is "false".

If optional="true", the variable may not be available at runtime due to various constraints. If it is not available at runtime, its value shall be undefined.

EXAMPLE If used in dependency expressions, an optional variable should be checked if it has a defined value before accessing the value itself:
<write> uis:hasDefinedValue('myvar') and uis:value('myvar')=4 </write>

NOTE 1 Examples for constraints that impact the availability of a socket element are: access control, different products using a common socket description but with some implementation variation (such as in UPnP DCPs).

NOTE 2 The availability of a socket element will be indicated at runtime to the URC through TUN-specific means (see ISO/IEC 24752-1).

If the variable has a 'dim' attribute (see 8.9) the 'optional' attribute applies to the variable as a whole, i.e. either it exists as dimensional variable or not at all.

8.8 The 'final' attribute

Some variables are provided initially by the target at session opening, and are never changed during a session. These should be marked with final="true".

A <variable> may have the Boolean 'final' attribute. Its default value is "false".

EXAMPLE 1 <variable id="serialNumber" type="xsd:integer" final="true" />

EXAMPLE 2 <variable id="sessionId" type="xsd:string" final="true" />

NOTE A final variable is different from a constant (see 9). A constant's value is specified in the socket description and thus shared among all targets that have the same socket description. A final variable, however, is specified by the target and can thus have different values for targets that have the same socket description.

If the variable has a 'dim' attribute (see 8.9) the 'final' attribute applies to the variable as a whole, i.e. all of its values are initialized by the target at session opening and will never change during a session.

8.9 The 'dim' attribute

The 'dim' attribute specifies variables as dimensional (with one or more dimensions). At runtime a dimensional variable has multiple values, each one for a particular combination of indices.

NOTE 1 A 1-dimensional variable is an array with one index, and a 2-dimensional variable is a table.

NOTE 2 The set of values for a particular socket variable results from the product of all index types that occur when walking the path from the <uiSocket> element down to the particular socket variable (see definition of path in 5.2.5.2).

The 'dim' attribute may be present for variables. If present, it shall contain a non-empty, ordered, space-separated list of type references that are the variable's index types. The first reference specifies the index type for the first dimension, the second type for the second dimension, and so on. Valid index type references are: (a) the name of a type that is defined in the <schema> part of the socket description, and (b) the fully qualified name (QName) of an external type.

NOTE 3 The index type may specify an infinite number of possible index values (e.g. xsd:integer has infinite values). However, the actual index values at runtime are a finite subset of the values allowed by the index type.

NOTE 4 A dimensional variable is similar to “associative arrays” in some programming languages, e.g. JavaScript.

NOTE 5 The ‘type’ attribute of a dimensional variable specifies the type of every value contained in that variable, and not the overall type of the variable (which is an array).

EXAMPLE 1 The volume of a 5.1 surround sound system can be defined as a vector of 6 integer values, each between 0 and 100. Each value represents the volume for a particular channel.

```
<variable id=”volume” type=”volumeType” dim=”channelType” />
….
<xsd:schema>
    <xsd:simpleType name=”volumeType”>
        <xsd:restriction base=”xsd:integer”>
            <xsd:minInclusive> 0 </minInclusive>
            <xsd:maxInclusive> 100 </minInclusive>
        </xsd:restriction>
    </simpleType>
    <simpleType name=”channelType”>
        <xsd:restriction base=”xsd:string”>
            <xsd:enumeration value=”FrontLeft” />
            <xsd:enumeration value=”FrontCenter” />
            <xsd:enumeration value=”FrontRight” />
            <xsd:enumeration value=”RearLeft” />
            <xsd:enumeration value=”RearRight” />
            <xsd:enumeration value=”SubWoofer” />
        </xsd:restriction>
    </simpleType>
</schema>
```

References to variables with ‘dim’ attributes are treated as references to the whole set (array) of values for this variable, if not otherwise specified. This is true in particular for atomic resources (see ISO/IEC 24752-5).

EXAMPLE 2 A label resource applies to the whole of a dimensional variable. The URC may use this label for any of its items, possibly complemented by the indices for the particular item.

8.10 The ‘includesRes’ attribute

A variable may have an ‘includesRes’ attribute of type Boolean. Its default value is “false”.

A value of “true” indicates that atomic resources for this variable are not available via resource sheets, and that atomic resources are provided by the target at runtime as a value-resource bundle with the actual value(s) of the variable. The atomic resources may change at runtime.

NOTE 1 For conformance, at least one textual label must be provided for the variable by the target as part of the value-resource bundle (see ISO/IEC 24752-1).

EXAMPLE For playing music, the user is requested to pick a player from a list of digital media rendering devices that are currently available in the home network. The list of devices is generated at runtime, with their labels reflecting their friendly names as configured in the individual devices.

```
<variable id="availableRenderers" type="uis:stringList" includesRes="true" />
<variable id="selectedRenderer" type="xsd:string" includesRes="true">
    <selection>
        <selectionSetDynamic id="pickFromRenderers" varRef="availableRenderers"/>
    </selection>
</variable>
```

NOTE 2 This mechanism should only be used for situations in which the resources are only available at runtime. It should not be misused for variables whose labels are known before runtime. The drawback of the “bundle approach” is that no supplemental resources can be provided.

8.11 Variable dependencies

8.11.1 General

Variable dependencies express when a variable is relevant or writeable by the user, and what dependencies between the values of variables are existing.

All dependencies are coded as XPath expressions and evaluated at runtime. Some types of dependency may change during a session.

Variable dependencies are specified by embedding them within a variable element using the markup:

```
<dependency>
    <relevant>expr1</relevant>
    <write>expr2</write>
    <calculate>expr3</calculate>
</dependency>
```

The <dependency> element may be present under <variable> elements. If present it shall occur exactly once.

8.11.2 'id' attribute for <dependency> subelements

Any of the subelements of <dependency> (see 8.11.3 through 8.11.14) may have an 'id' attribute.

If present, the 'id' attribute shall be of type ID as defined by XML Schema Part 2: Datatypes. It provides an identifier which is used to refer to the specific dependency within the socket description and as a means of binding externally defined resources such as labels. The 'id' value shall be unique among all 'id' attributes within the socket description.

8.11.3 The <relevant> dependency

The <relevant> dependency specifies when a variable may be presented to the user, i.e. how relevant it is based on a particular state of the target. This does not necessarily mean that it shall be directly accessible in a user interface derived from the socket, but that there shall be some navigation path by which the user can find it.

The <relevant> element may be present in a <dependency> element. If present, it shall occur exactly once. If not present, the variable inherits the <relevant> dependency from the closest ancestor <set> element that has a <relevant> dependency specified (see 7.4.3).

EXAMPLE A variable representing an enrolment number may be only available if the 'userType' variable has the value "student":

```
<dependency>
    <relevant> uis:value('userType') eq 'student' </relevant>
</dependency>
```

The content of the <relevant> element shall be a valid XPath expression (with allowed extensions as specified in 5.2) that evaluates to a Number or Boolean value. Numbers shall be in the range of [0.0, 1.0] (both 0.0 and 1.0 are included). This allows for any number of ordered degrees of relevance, with higher numbers meaning "more relevant" than lower numbers. 0.0 represents "not relevant", and 1.0 represents "relevant". If the XPath expression evaluates to a Boolean value, "true" shall be interpreted as 1.0, and "false" as 0.0.

NOTE The result of the XPath expression may change during a session.

If no <relevant> dependency is specified (neither on the variable nor on any of its ancestor sets), or if it is empty the <relevant> dependency of the variable shall be always true, even when a notify element is active.

8.11.4 The <write> dependency

The <write> dependency specifies whether the user can change the value of a variable. If the <write> dependency is false, then it is inappropriate to modify the variable's value; otherwise it is appropriate to attempt to set a new value, although there is no guarantee that it will succeed.

The <write> element may be present in a <dependency> element. If present, it shall occur exactly once. If not present, the variable inherits the <write> dependency from the closest ancestor <set> element that has a <relevant> dependency specified (see 8.11.4).

EXAMPLE 1 A hotel reservation system may have a variable for the type of credit payment that can only be written if the user has opted to pay by credit card. This dependency can be expressed as:

```
<variable id=”creditCard” type=”creditCardType”>
    <dependency>
        <write> uis:value(‘creditPayment’) </write>
    </dependency>
</variable>
```

EXAMPLE 2 Dependencies can be used to represent ordering constraints. For example, a service's variables may only become writeable after the user has accepted the terms of a license. The user's acceptance can be represented as a Boolean variable with id="warrantySigned", and the other variables would have write attributes of the form:

```
<variable id=”serviceVar” type=”serviceVarType”>
    <dependency>
        <write> uis:value(‘warrantySigned’) </write>
    </dependency>
</variable>
```

The content of the <write> element shall be either empty or a valid XPath expression (with allowed extensions as specified in 5.2) that evaluates to a Boolean value.

NOTE The result of the XPath expression may change during a session.

If a variable's <write> element is empty or if it inherits an empty <write> dependency from an ancestor set the <write> dependency of the variable shall be unknown.

NOTE The introduction of an unknown value is analogous to a three-valued Boolean logic that allows for uncertainty. If the <write> dependency is unknown, the URC has no other choice than to find out by trial whether it can change the variable or not.

If no <write> dependency is specified (neither on the variable nor on any of its ancestor sets), the <write> dependency of the variable shall be always true, even when a notify element is active.

8.11.5 The <insert> dependency

The <insert> dependency specifies whether the user can modify the pertaining set of valid index combinations at runtime (with the pertaining set being the indices specified by the ‘dim’ attribute on the corresponding <variable> element).

The content of the <insert> dependency shall be an XPath expression that evaluates to a Boolean value which may change during a session. If it evaluates to false, then it is inappropriate to add or delete an index combination; otherwise it is appropriate to attempt to add or delete an index combination, although there is no guarantee that it will succeed.

NOTE 1 “Adding an index combination” means to add a variable component for a particular new index combination within the index space defined by the corresponding ‘dim’ attribute. Note that the initial value of the new variable component is determined by the target. “Removing an index combination” means to delete a variable component for a particular index combination that is currently occurring.

The <insert> element may be present in a <dependency> element only if the corresponding variable has a 'dim' attribute (see 8.9), and is not final (see 8.8). If present, it shall occur exactly once.

Its default value is "false()".

NOTE 2 There is no inheritance from the <insert> dependency of a <set> element to that of a contained <variable> element. Each <insert> dependency pertains only to the set of indices that are specified by the 'dim' attribute of the corresponding level.

EXAMPLE With the following socket description, illustrating a table with rows and columns, the URC can request to add or remove a row (index), but not a column (index).

```
<set id="row" dim="xsd:positiveInteger">
    <dependency>
        <insert> true() </insert>
    </dependency>
    <variable id="column" type="xsd:integer" dim="xsd:positiveInteger">
        <dependency>
        <insert> false() </insert>
        </dependency>
    </variable>
</set>
```

8.11.6 The <calculate> dependency

The calculate dependency applies only to variables, and specifies how the value of a variable depends on the values of other variables.

The <calculate> element may occur in a <dependency> element. If present, it shall occur exactly once.

EXAMPLE If the value of a variable representing a total is the sum of two specific subtotal variables, this dependency can be expressed as:

```
<variable id="num1" type="xsd:double" />
<variable id="num2" type="xsd:double" />
<variable id="total" type="xsd:double">
    <dependency>
        <write> false() </write>
        <calculate> uis:value('num1') + uis:value('num2') </calculate>
    </dependency>
</variable>
```

The content of the <calculate> element shall be a valid XPath expression (with allowed extensions as specified in 5.2) that evaluates to a value of a type that is compatible to the pertinent variable type. It shall not contain cycles. It may change during a session.

A calculate dependency may only be provided for permanently read-only variables (those with a <write> dependency of "false()").

8.11.7 The <length> dependency

The <length> element specifies an XPath expression (with syntax as defined in 5.2) that evaluates to a number. This number specifies the allowed length of the variable at runtime. It shall not change during a session.

NOTE The behavior in case of a changing <length> dependency is undefined. For example, a URC may ignore any change. A conformance checking tool may reject <length> dependencies that reference socket elements other than constants and final variables.

The <length> element may occur in a <dependency> element, but only if the type of the pertaining variable is xsd:string, or derived from xsd:string. If present, it shall occur once.

EXAMPLE 1 The following specifies a length constraint of 5:
<length> 5 </length>

EXAMPLE 2 The following specifies that the value of the (final) variable with id 'lengthVar' be used as length constraint:
<length> uis:value("lengthVar") </length>

The <length> element should only be used if the allowed length has to be determined at session initialization, in relation to other socket elements. If a variable has a length constraint that can be determined before runtime, it should express this constraint through a type that is defined in the <schema> section (see 12) of the socket description. If a variable has a length constraint on both its type and through a <length> element, they shall result in the same number.

8.11.8 The <minLength> dependency

The <minLength> element specifies an XPath expression (with syntax as defined in 5.2) that evaluates to a number. This number specifies the minimum length of the variable at runtime. It shall not change during a session.

NOTE The behavior in case of a changing <minLength> dependency is undefined.

The <minLength> element may occur in a <dependency> element, but only if the type of the pertaining variable is xsd:string, or derived from xsd:string. If present, it shall occur once.

The <minLength> element should only be used if the allowed minimal length is determined at session initialization, in relation to other socket elements. If a variable has a minimal length constraint that can be determined before runtime, it should express this constraint through a type that is defined in the <schema> section (see 12) of the socket description. If a variable has a minimal length constraint on both its type and through a <length> element, the variable shall comply with both (i.e. the higher number is taken as minimal length).

8.11.9 The <maxLength> dependency

The <maxLength> element specifies an XPath expression (with syntax as defined in 5.2) that evaluates to a number. This number specifies the maximum length of the variable at runtime. It shall not change during a session.

NOTE The behavior in case of a changing <maxLength> dependency is undefined.

The <maxLength> element may occur in a <dependency> element, but only if the type of the pertaining variable is xsd:string, or derived from xsd:string. If present, it shall occur once.

The <maxLength> element should only be used if the allowed maximal length is determined at session initialization, in relation to other socket elements. If a variable has a maximal length constraint that can be determined before runtime, it should express this constraint through a type that is defined in the <schema> section (see 12) of the socket description. If a variable has a maximal length constraint on both its type and through a <length> element, the variable shall comply with both (i.e. the lower number is taken as maximal length).

8.11.10 The <pattern> dependency

The <pattern> element specifies an XPath expression (with syntax as defined in 5.2) that evaluates to a string. This string specifies a regular expression that defines a pattern for the allowed values of the variable at runtime. The pattern syntax is as specified for the 'pattern' constraining facet in XML Schema Part 2. The pattern shall not change during a session.

NOTE The behavior in case of a changing <pattern> dependency is undefined.

The <pattern> element may occur once in a <dependency> element, but only if the type of the pertaining variable is xsd:string, or derived from xsd:string.

NOTE 1 If a variable's value space needs to be described by a union of patterns, these can be ORed together by the "|" operator inside a regular expression.

EXAMPLE 1 The following specifies a pattern for the string-based representation of any positive integer:
<pattern> "[1-9][0-9]*" </pattern>

EXAMPLE 2 The following specifies that the value of the (final) variable 'patternVar' be used as pattern:
<pattern> uis:value("patternVar") </pattern>

NOTE 2 If any of the characters '&', '<' or '>' occur inside <pattern> they need to be XML escaped.

EXAMPLE 3 The following specifies a pattern for strings like "A&A", "A&B", etc.
<pattern> "[A-Z]&[A-Z]" </pattern>

The <pattern> element should only be used if the specified pattern is determined at session initialization, in relation to other socket elements. If a variable has a pattern constraint that can be determined before runtime, it should express this constraint through a type that is defined in the <schema> section (see 12) of the socket description. If a variable has pattern constraints on both its type and through <pattern> elements, the variable shall comply with all (i.e. they are ANDed together).

8.11.11 The <minInclusive> dependency

The <minInclusive> element specifies an XPath expression (with syntax as defined in 5.2) that evaluates to a value of the same type as the variable it belongs to. This value specifies the minimum value (inclusively) of the variable at runtime. It shall not change during a session.

NOTE 1 The behavior in case of a changing <minInclusive> dependency is undefined.

The <minInclusive> element may occur in a <dependency> element, but only if the pertaining variable has a type whose value space is totally ordered. If present, it shall occur once.

NOTE 2 In XML, built-in datatypes are unordered (e.g. xsd:string) or only partially ordered (e.g. xsd:time). For these datatypes, <minInclusive> and other runtime constraints that require a total order relation cannot be used.

EXAMPLE 1 The following specifies 10 as an inclusive lower bound for a variable of type xsd:integer:
<minInclusive> 10 </minInclusive>

EXAMPLE 2 The following specifies double of the value of a constant with id 'lowerBound' as an inclusive lower bound for a variable of type xsd:integer:
<minInclusive> 2 * uis:value("lowerBound") </minInclusive>

The <minInclusive> element should only be used if the allowed minimal value is determined at session initialization, in relation to other socket elements. If a variable has a minimal value constraint that can be determined before runtime, it should express this constraint through a type that is defined in the <schema> section (see 12) of the socket description. If a variable has a minimal value constraint on both its type and through a <minInclusive> element, the variable shall comply with both (i.e. the higher value is taken as minimal value).

8.11.12 The <maxInclusive> dependency

The <maxInclusive> element specifies an XPath expression (with syntax as defined in 5.2) that evaluates to a value of the same type as the variable it belongs to. This number specifies the maximum value (inclusively) of the variable at runtime. It shall not change during a session.

NOTE The behavior in case of a changing <maxInclusive> dependency is undefined.

The <maxInclusive> element may occur in a <dependency> element, but only if the pertaining variable has a type whose value space is totally ordered. If present, it shall occur once.

The <maxInclusive> element should only be used if the allowed maximal value is determined at session initialization, in relation to other socket elements. If a variable has a maximal value constraint that can be determined before runtime, it should express this constraint through a type that is defined in the <schema> section (see 12) of the socket description. If a variable has a maximal value constraint on both its type and through a <maxInclusive> element, the variable shall comply with both (i.e. the lower value is taken as maximal value).

8.11.13 The <minExclusive> dependency

The <minExclusive> element specifies an XPath expression (with syntax as defined in 5.2) that evaluates to a value of the same type as the variable it belongs to. This number specifies the minimum value (exclusively) of the variable at runtime. It shall not change during a session.

NOTE The behavior in case of a changing <minExclusive> dependency is undefined.

The <minExclusive> element may occur in a <dependency> element, but only if the pertaining variable has a type whose value space is totally ordered. If present, it shall occur once.

The <minExclusive> element should only be used if the allowed minimal value is determined at session initialization, in relation to other socket elements. If a variable has a minimal value constraint that can be determined before runtime, it should express this constraint through a type that is defined in the <schema> section (see 12) of the socket description. If a variable has a minimal value constraint on both its type and through a <minExclusive> element, the variable shall comply with both (i.e. the higher value is taken as minimal value).

8.11.14 The <maxExclusive> dependency

The <maxExclusive> element specifies an XPath expression (with syntax as defined in 5.2) that evaluates to a value of the same type as the variable it belongs to. This number specifies the maximum value (exclusively) of the variable at runtime. It shall not change during a session.

NOTE The behavior in case of a changing <maxExclusive> dependency is undefined.

The <maxExclusive> element may occur in a <variable> element, but only if the pertaining variable has a type whose value space is totally ordered. If present, it shall occur once.

The <maxExclusive> element should only be used if the allowed maximal value is determined at session initialization, in relation to other socket elements. If a variable has a maximal value constraint that can be determined before runtime, it should express this constraint through a type that is defined in the <schema> section (see 12) of the socket description. If a variable has a maximal value constraint on both its type and through a <maxExclusive> element, the variable shall comply with both (i.e. the lower value is taken as maximal value).

8.12 Selection

8.12.1 General

A list variable (i.e. a variable whose type has been derived by list) may provide a set of values that either restrict a variable's value space (closed selection) or provide suggested values for user input (open selection). The list items shall be of the same type as the restricted variable, or a subtype thereof. The set of values is described by a <selection> element for the restricted variable. Both static and dynamic value sets may be defined, and these may be combined within a single <selection> element.

The <selection> element may be present in a <variable> element. If present, it shall occur once.

EXAMPLE <selection> … </selection>

8.12.2 The 'closed' attribute

The 'closed' attribute of Boolean type may accompany the <selection> element to indicate whether the set of values is closed or open. The default value is "true", i.e. a closed selection.

EXAMPLE <selection closed="false"> … </selection>

The value "false", i.e. an open selection, indicates a selection containing optional suggested user values, i.e. a user may also enter a value not represented within the selection.

8.12.3 Static and dynamic selection sets

8.12.3.1 General

A selection is composed of one or more selection sets which may be either static or dynamic. Each selection set is described with a separate <selectionSetStatic> or <selectionSetDynamic> element, and shall have an 'id' attribute of type xsd:ID, that can be used to refer to the set.

EXAMPLE A closed selection composed of two selection sets: one with a static set of cities (myStaticCities) that is specified at run-time by the variable dynCitiesVariable. Note that staticCityType is derived from uis:stringListItem (see example in 12.4) which is derived from xsd:string (see 12.4); and dynCitiesVariable is supposed to be of type uis:stringList.

```
<selection>
    <selectionSetStatic id="myStaticCities" typeRef="staticCityType"/>
    <selectionSetDynamic id="myDynamicCities" varRef="dynCitiesVariable"/>
</selection>
```

8.12.3.2 The <selectionSetStatic> element

A static selection set (i.e. that does not change at runtime) is denoted by a <selectionSetStatic> element. It shall have a 'typeRef' attribute which contains the name of a type.

The referenced type shall be derived by restriction from a simple type. Unions of enumerations are also valid, and provide a mechanism by which structure within the selection set can be captured. See 12.5 for an example of such a type.

Valid selections are all elements conforming to the given type.

NOTE User agents may render the set of values in a way that reflects the structure of the referenced types and the selection set elements, where such structure is present. For example, dividers could separate different groups of options of a visual combo-box in a GUI environment.

8.12.3.3 The <selectionSetDynamic> element

A dynamic selection set is denoted by a <selectionSetDynamic> with a 'varRef' attribute which shall contain the id of a variable of type uis:csvlist, or the id of a list variable whose member type is compatible with the type of the host variable.

Valid selections are all elements that are currently held by the list variable.

EXAMPLE In a television set, the variable "channel" allows only for values that are contained in the whitespace separated list given by the value of the variable "channelList". During run-time the value of the variable "channelList" may change at any time, thus changing the set of available values of the variable "channel".

```
<variable id="channel" type="xsd:string">
   <selection closed="true">
      <selectionSetDynamic id="currentChannels" varRef="channelList" />
   </selection>
</variable>
<variable id="channelList" type="uis:stringList" />
```

8.13 Platform-specific mapping information

The <mapping> element may be used any number of times to include platform-specific mapping information for the containing variable.

A <mapping> element shall have a 'platform' attribute whose value is not restricted by this International Standard.

A <mapping> element may have arbitrary element content and subelements. However, subelements shall be from namespaces other than the uis namespace.

EXAMPLE UPnP-specific mapping information for a variable may be included as follows:

```
<mapping platform="UPnP">
    …
</mapping>
```

NOTE Socket descriptions that contain platform specific mapping information lose their platform neutrality. Although multiple mappings may be specified in a socket description (one for each platform) it is recommended to consider other mechanisms of specifying the binding to platform-specific technologies. For example, mapping information may be provided in an external file with references to the elements of the socket description.

8.14 Properties from DCMI

Any element and element refinement from ISO 15836, Dublin Core Metadata Element Set, or the set of Dublin Core Metadata Initiative (DCMI) Metadata Terms may be used to describe a socket variable, if appropriate. Each of them may occur multiple times within the <variable> element.

9 Constants

9.1 General

Constants have the same value, given by the socket description, for all sessions of that socket. A constant's value is known before runtime.

NOTE 1 Constants are common to all instances of a particular device if they share the same socket description.

EXAMPLE Examples are the model name of a digital projector, but not its serial number.

A constant may occur as subelement of a <set> element (see 7) or as subelement of the <uiSocket> element (see 6.1).

A constant is defined with the markup:

<constant> … </constant>

NOTE 2 A constant is very similar to a variable, and has many of its attributes and subelements in common.

A constant shall not occur within a dimensional set (see 7.3), neither as a direct nor indirect descendent.

NOTE 3 This restriction is given since, for a constant inside a dimensional set, the author would have to define a separate value of the constant for each set instantiation (and this may be known at runtime only).

9.2 Constant attributes

A <constant> element shall have the following attributes:

- 'id' as defined in 8.2

A <constant> element may have the following additional attributes:

- 'type' as defined in 8.3. If the 'type' attribute is missing, it defaults to xsd:string
- 'sensitive' as defined in 8.5
- 'timeout' as defined in 8.6

9.3 Constant subelements

A <constant> element may have any of the following subelements:

- One <dependency> with one <relevant> element inside, as defined in 8.11.3
- Any number of <mapping> as defined in 8.13
- Any number of DCMI properties, as defined for variables in 8.14

9.4 Constant value

A <constant> element shall include a value which is specified as its content. Type information may be given by the 'type' attribute (see 8.3).

NOTE The text surrounding the value is provided separately as a label for this element.

EXAMPLE A model number could be defined as a constant in the following way:
<constant id="model" type="xsd:string"> XYZ1000 </constant>

If the constant value includes any one of the following characters, they shall be XML encoded in the socket description: '<', '>', '&'.

EXAMPLE The 'firstAddress' constant is of type mytype:addressType and should be specified with the following value:

```
<address>
    <street>1000 Washington Ave.</street>
    <city>Madison</city>
    <zipCode>53706</zipCode>
</address>
```

This is specified in the socket description with the following code:

```
<constant id=”firstAddress” type=”mytype:addressType”>
    &lt;address&gt;
        &lt;street&gt;1000 Washington Ave.&lt;/street&gt;
        &lt;city&gt;Madison&lt;/city&gt;
        &lt;zipCode&gt;53706&lt;/zipCode&gt;
    &lt;/address&gt;
</constant>
```

10 Commands

10.1 General

Command elements are used to capture commands that a user may issue to a target. A command is a core function that a user can request a target to perform and that cannot be represented by a variable. A command should represent some function beyond simply manipulating the value of a single variable.

EXAMPLE The seek button on a CD player and a submit button on an online form.

A command may occur as subelement of a <set> element (see 7) or as subelement of the <uiSocket> element (see 6.1).

A command is defined with the markup:
<command> … </command>

10.2 The ‘id’ attribute

A <command> element shall have an ‘id’ attribute.

The ‘id’ attribute shall be of type ID as defined by XML Schema Part 2: Datatypes. It provides an identifier which is used to refer to the element within the socket description and as a means of binding externally defined resources such as labels. The ‘id’ value shall be unique among all ‘id’ attributes within the socket description.

NOTE ‘id’ attribute values are not normally presented to users and need not be human comprehensible.

EXAMPLE <command id="seek">

10.3 The ‘type’ attribute

10.3.1 General

<command> elements may have a ‘type’ attribute. The type of a command describes the states that the command may have. The state provides information about the last execution (request) of the command.

The ‘type’ attribute takes a QName as value (see XML Schema Part 2: Datatypes). If the namespace prefix is not specified, the namespace http://myurc.org/ns/uisocketdesc is assumed.

EXAMPLE 1 The following type values are semantically the same: type="uis:voidCommand" and type="voidCommand", assuming that the namespace identifier uiSocket is bound to http://myurc.org/ns/uisocketdesc.

Three command types are defined. The default type is uis:voidCommand.

10.3.2 uis:voidCommand

type="uis:voidCommand" - the command is stateless.

A command of type uis:voidCommand shall not have output or input-output parameters.

NOTE This restriction is necessary since a command with output or input-output parameters cannot execute multiple instances of the same command at one time, since there is only one set of output parameters that can receive the results.

10.3.3 uis:basicCommand

type="uis:basicCommand" - the command has the following set of states available: initial, rejected, inProgress, done, succeeded, failed.

The meanings of the states are as follows:

- initial: The command has not been requested for execution
- rejected: The command's execution has been requested, but the target rejected the request
- inProgress: The command is currently being executed as requested
- done: The command has been executed
- succeeded: The command has been executed with success (this is more specific than done and should be preferred if known by the target)
- failed: The command was executed but failed (this is more specific than done and should be preferred if known by the target)

EXAMPLE A pressed floor button in an elevator is a command with state inProgress.

NOTE 1 Upon failure of a command's execution the target would typically convey an error message to the user in a separate way, e.g. by raising a notification.

NOTE 2 The state of a command says nothing about whether this command may be executed (again). This is specified by the 'execute' dependency (see 10.10.4).

NOTE 3 A command of type uis:basicCommand cannot be invoked multiple times on the target unless the previous invocation has completed (see ISO/IEC 24752-1).

10.3.4 uis:timedCommand

type="uis:timedCommand" - the same as uis:basicCommand except that the state inProgress makes an additional timeToComplete field available.

timeToComplete is of type xsd:duration.

The value of timeToComplete may be undefined at any time. An undefined value is represented by an empty string.

If the command is in state inProgress, the target may periodically "count down" the value of timeToComplete until the command switches to another state. The value of timeToComplete (if not the empty string) is a hint from the target to the URC as to how long the estimated time to complete is – no guarantee is provided by the target whatsoever. timeToComplete is read-only for the URC. The value of timeToComplete is invalid in any other state than inProgress.

EXAMPLE A "book" command in an online ticket system may be of type uis:timedCommand and thus convey the estimated time for the booking process in real-time.

NOTE 1 A command of type uis:timedCommand cannot be invoked multiple times on the target unless the previous invocation has completed (see ISO/IEC 24752-1).

NOTE 2 The value of the timeToComplete field can be accessed in dependency expressions by appending "[timeToComplete]" to the path in the XPath function uis:value(path).

10.4 The 'sensitive' attribute

A <command> element may have a 'sensitive' attribute. It is a Boolean indicating whether the command represents a legally sensitive action.

EXAMPLE An emergency call button command whose form of presentation has legal implications would be described with the markup:

```
<command id="emergency" sensitive="true" />
```

The default value is "false".

10.5 The 'sufficient' attribute

The <command> element may have a 'sufficient' attribute of type Boolean.

The value "true" indicates that the command is sufficiently specified. A command is sufficiently specified iff its set of assert dependencies, if any, is sufficient. A set of assert dependencies is sufficient iff whenever all their expressions evaluate to true, the command has completed successfully on the target. The 'assert' dependency is specified in 10.10.6.

NOTE 1 The spelling "iff" above and below should be read "if and only if" and means that the statements before and after "iff" are equivalent. So if any one of them is true, the other is true, too. (Also, if any one of them is false, the other is false, too.)

EXAMPLE For the "floor 4" command button of an elevator, the following postcondition is not sufficient, because the command could fail and the value of the expression could still be true:

```
<dependency>
    <assert> uis:value('currentFloor') > 0</assert>
</dependency>
```

However, the following postcondition is sufficient:

```
<dependency>
    <assert> uis:value('currentFloor') = 4 </assert>
</dependency>
```

NOTE 2 In the case of sufficient descriptions of commands, an intelligent URC can apply advanced inference techniques and thus provide a more usable user interface. For example, it can conclude that a command doesn't need to be invoked if its assert dependencies are already true.

If present, the 'sufficient' attribute of <command> overrides the 'sufficient' attribute of <uiSocket> (see 6.4), but only for that particular command. If not present, the value of the 'sufficient' attribute of <uiSocket> applies to the command.

10.6 The 'complete' attribute

The <command> element may have a 'complete' attribute of type Boolean.

The value "true" indicates that all commands in the socket description are completely specified with regard to their parameters. A command is completely specified iff its input and output parameter sets are complete. Command parameters are specified in 10.12.

A set of command input parameters is complete iff the outcome of a successful execution of the command is guaranteed to be the same whenever the command is executed with the same set of input parameter values.

A set of command output parameters is complete iff no socket variables other than those specified as output parameters are modified as a result of the command's execution for all possible sets of input parameters.

NOTE In the case of complete descriptions of commands, an intelligent URC can apply advanced inference techniques and thus provide a more usable user interface.

If present, the 'complete' attribute of <command> overrides the 'complete' attribute of <uiSocket> (see 6.5), but only for that particular command. If not present, the value of the 'complete' attribute of <uiSocket> applies to the command.

10.7 The 'optional' attribute

A command may have an 'optional' attribute of type Boolean. Its default value is "false".

If optional="true", the command may not be available at runtime due to various constraints.

NOTE 1 Examples for constraints that impact the availability of a socket element are: access control, different products using a common socket description but with some implementation variation (such as in UPnP DCPs).

NOTE 2 The availability of a socket element will be indicated at runtime to the URC through TUN-specific means (see ISO/IEC 24752-1).

10.8 The 'dim' attribute

The 'dim' attribute specifies a command as dimensional (with one or more dimensions). At runtime a dimensional command has multiple states, each one for a particular combination of indices.

The 'dim' attribute may be present for commands. If present, it shall contain a non-empty, ordered, space-separated list of type references that are the command's index types. The first reference specifies the index type for the first dimension, the second type for the second dimension, and so on. Valid index type references are: (a) the name of a type that is defined in the <schema> part of the socket description, and (b) the fully qualified name (QName) of an external type.

NOTE 1 The 'type' attribute of a dimensional command specifies the type of every value contained in that command, and not the overall type of the command.

NOTE 2 The set of values for a particular socket command results from the product of all index types that occur when walking the path from the <uiSocket> element down to the particular socket command (see definition of path in 5.2.5.2).

NOTE 3 The index type may specify an infinite number of possible index values (e.g. xsd:integer has infinite values). However, the actual index values at runtime are a finite subset of the values allowed by the index type.

References to commands with 'dim' attributes are treated as references to the whole set of values for this command, if not otherwise specified. This is true in particular for atomic resources (see ISO/IEC 24752-5).

NOTE The use of a dimensional command is only reasonable if the set of commands is really homogenous, i.e. each command has identical characteristics including label (message) and help text.

10.9 The 'includesRes' attribute

A command with a type of uis:basicCommand or uis:timedCommand may have an 'includesRes' attribute of type Boolean. Its default value is "false".

A value of "true" indicates that atomic resources for this command are not available via resource sheets, and that atomic resources are provided by the target at runtime as a value-resource bundle with the actual state of the command. The atomic resources may change at runtime.

NOTE 1 For conformance, at least one textual label must be provided for the command by the target as part of the value-resource bundle (see ISO/IEC 24752-1).

NOTE 2 The ability for a target to provide labels at runtime should only be used for situations in which the resources are only available at runtime. It should not be misused for commands whose labels are known before runtime. The drawback of the “bundle approach” is that no supplemental resources can be provided.

10.10 Command dependencies

10.10.1 General

Command dependencies express when a command is relevant or executable by the user, and what a command’s effect is (“postcondition”).

Command dependencies are specified by embedding them within a command element using the markup:

```
<dependency>
    <relevant>expr1</relevant>
    <write>expr2</write>
</dependency>
```

The <dependency> element may be present for a <command> element. If present it shall occur exactly once.

10.10.2 ‘id’ attribute for <dependency> subelements

Any of the subelements of <dependency> (see 10.10.3 through 10.10.6) may have an ‘id’ attribute.

If present, the ‘id’ attribute shall be of type ID as defined by XML Schema Part 2: Datatypes. It provides an identifier which is used to refer to the specific dependency within the socket description and as a means of binding externally defined resources such as labels. The ‘id’ value shall be unique among all ‘id’ attributes within the socket description.

10.10.3 The <relevant> dependency

The <relevant> dependency specifies when a command may be presented to a user, i.e. how relevant it is based on a particular state of the target. This does not necessarily mean that it shall be directly accessible in a user interface derived from the socket, but that there shall be some navigation path by which the user can find it.

The <relevant> element may be present in a <dependency> element. If present, it shall occur exactly once. If not present, the command inherits the <relevant> dependency from the closest ancestor <set> element that has a <relevant> dependency specified (see 7.4.3).

EXAMPLE A command to make a travel reservation may only be relevant when the user is currently defining a reservation. This dependency can be expressed using the <relevant> dependency as follows:

```
<command id=”makeReservation” type=”uis:basicCommand”>
            <dependency>
        <relevant> uis:value(‘activity’) eq 'makingReservation' </relevant>
    </dependency>
</command>
```

The content of the <relevant> element shall be as specified for the <relevant> dependency of variables (see 8.11.3).

If no <relevant> dependency is specified (neither on the command nor on any of its ancestor sets) or if it is empty, the <relevant> dependency of the command shall be always true, even when a notify element is active.

10.10.4 The <write> dependency

This dependency for commands specifies whether the user can activate the command. This is called "<write> dependency" or "precondition". If the precondition is false, then it is inappropriate to perform the action; otherwise it is appropriate to attempt to perform the action, although there is no guarantee that it will succeed.

The <write> element may be present in a <dependency> element. If present, it shall occur exactly once. If not present, the command inherits the <write> dependency from the closest ancestor <set> element that has a <write> dependency specified (see 8.11.4).

EXAMPLE An elevator button commanding the elevator to go to level 4 only be triggered if the elevator is not already at level 4. This dependency can be expressed as:

```
<dependency>
    <write> uis:value('currentFloor') != 4 </write>
</dependency>
```

The content of the <write> element shall be as specified for the <write> dependency of variables (see 8.11.4).

If a command's <write> dependency is empty or if it inherits an empty <write> dependency from an ancestor set the <write> dependency of the command shall be unknown.

NOTE The introduction of an unknown value is analogous to a three-valued Boolean logic that allows for uncertainty. If the <write> dependency is unknown, the URC has no other choice than to find out by trial whether it can execute the command or not.

If no <write> dependency is specified (neither on the command nor on any of its ancestor sets), the <write> dependency of the command shall be always true, even when a notify element is active.

10.10.5 The <insert> dependency

The <insert> dependency specifies whether the user can modify the pertaining set of valid index combinations at runtime (with the pertaining set being the indices specified by the 'dim' attribute on the corresponding <command> element).

The content of the <insert> dependency shall be an XPath expression that evaluates to a Boolean value which may change during a session. If it evaluates to false, then it is inappropriate to add or delete an index combination; otherwise it is appropriate to attempt to add or delete an index combination, although there is no guarantee that it will succeed.

NOTE 1 "Adding an index combination" means to add a command component for a particular new index combination within the index space defined by the corresponding 'dim' attribute. Note that the status of an added command component is determined by the target. "Removing an index combination" means to delete a command component for a particular index combination that is currently occurring.

The <insert> element may be present in a <dependency> element only if the corresponding command has a 'dim' attribute (see 10.8). If present, it shall occur exactly once.

Its default value is "false()".

NOTE 2 There is no inheritance from the <insert> dependency of a <set> element to that of a contained <command> element. Each <insert> dependency pertains only to the set of indices that are specified by the 'dim' attribute of the corresponding level.

10.10.6 The <assert> dependency

The <assert> dependency specifies a Boolean expression that is guaranteed to evaluate to true after the command has been successfully executed. This is called "assertion" or "postcondition".

NOTE 1 If the <assert> dependency is guaranteed to be sufficient, one can infer from a true postcondition that the command has succeeded. See 10.5 for details on the 'sufficient' attribute.

For commands of type uis:voidCommand (see 10.3.2) the postcondition is guaranteed to be true for any requested invocation of the command. For commands of type uis:basicCommand (see 10.3.3) or uis:timedCommand (see 10.3.4), the postcondition is guaranteed to be true for all invocations of the command that result in the state "succeeded".

The <assert> element may be present in a <dependency> element. If present, it shall occur exactly once.

EXAMPLE 1 After the successful execution of the "floor 4" button of the elevator the value of the variable 'currentFloor' will be 4. This assertion can be expressed as:

```
<dependency>
    <assert> uis:value('currentFloor') = 4 </assert>
</dependency>
```

EXAMPLE 2 This "startPlay" command will trigger the target requesting the URC for opening a sub-session ("spawn forward request", see ISO/IEC 24752-1) on the "play" socket.

```
<command id="startPlay" type="uis:voidCommand">
    <dependency>
        <assert> uis:sessionForward("spawn", "http://example.com/target/play") </assert>
    </dependency>
</command>
```

NOTE 2 Advanced URCs may exploit knowledge on postconditions for generating more usable user interfaces. Postconditions are especially useful if they are sufficiently specified (see 10.5).

The content of the <assert> element shall be either empty or a valid XPath expression that evaluates to a Boolean value.

If a command's <assert> dependency is empty the <assert> dependency of the command is unknown. If no <assert> dependency is specified, the <assert> dependency for the command shall be always true.

10.11 Platform-specific mapping information

The <mapping> element may be used any number of times for commands to include platform-specific mapping information for the containing command.

A <mapping> element shall have a 'platform' attribute whose value is not restricted by this International Standard.

A <mapping> element may have arbitrary element content and subelements. However, subelements shall be from namespaces other than the uis namespace.

NOTE Socket descriptions that contain platform specific mapping information lose their platform neutrality. Although multiple mappings may be specified in a socket description (one for each platform) it is recommended to consider other mechanisms of specifying the binding to platform-specific technologies. For example, mapping information may be provided in an external file with references to the elements of the socket description.

10.12 Command parameters

10.12.1 General

A command element may have any number of <param> subelements, each defining one distinct parameter pertaining to the command.

NOTE 1 A command parameter's scope is either local to the command or global (in the socket). See 10.12.2 and 10.12.3 below.

EXAMPLE A command to call an elevator has three input parameters: the origin floor, the destination floor and the wait time for the elevator door to stay open while nobody passes the door. The parameters “origin floor” and “destination floor” are local, i.e. they have to be supplied for every call. The wait time is a global parameter, i.e. it can be set as a global setting for all subsequent calls.

```
<command id="callElevator" type=”uis:basicCommand”>
    <param id="originFloor" dir="in" type="xsd:integer" />
    <param id="destinationFloor" dir="in" type="xsd:integer" />
    <param idref=”waitTime” dir=”in” />
</command>
```

NOTE 2 Advanced URCs may exploit knowledge on command parameters for generating more usable user interfaces. This information is especially useful if the command is guaranteed to be completely specified. See 10.6 for details on the ‘complete’ attribute.

10.12.2 The ‘id’ attribute (local parameter)

A <param> element shall have either one ‘id’ attribute or one ‘idref’ attribute (see 10.12.3). A <param> element with an ‘id’ attribute is called “local parameter”.

The value of local parameters shall be available to the user (i.e. has a defined state) only when the command’s <relevant> dependency evaluates to true.

The value of the ‘id’ attribute shall be unique among all ‘id’ attributes within the socket description.

The ‘id’ attribute shall be of type ID as defined by XML Schema Part 2: Datatypes. It provides an identifier which is used to refer to the element within the socket description and as a means of binding externally defined resources such as labels. The ‘id’ attribute is not presented to users and need not be human comprehensible.

NOTE On command invocation, the URC will send the values of all local input parameters (i.e. local parameters with a dir value of “in” or “inout”) to the target. After execution, the target will return the values of all local output parameters (i.e. local parameters with a dir value of “out” or “inout”) to the URC.

10.12.3 The ‘idref’ attribute (global parameter)

A <param> element shall have either one ‘idref’ attribute or one ‘id’ attribute (see 10.12.2). A <param> element with an ‘idref’ attribute is called “global parameter”.

The ‘idref’ attribute shall reference a variable of the same socket, i.e. the value of the ‘idref’ attribute shall be the value of the ‘id’ attribute of a <variable> element in the relevant socket description. As an exception, the ‘idref’ attribute may also reference a set of the same socket by its ‘id’, meaning that all variables contained in the set are used as global parameters for the command.

Local parameters of other commands shall not be referenced by an ‘idref’ attribute.

The ‘idref’ attribute is of type IDREF as defined by XML Schema Part 2: Datatypes. It specifies a constraint between the command it is contained in and a variable: If the value of the ‘dir’ attribute (see 10.12.4) is “in”, the target will use the current value of the referenced variable as input value for the execution of the command. If the value of the ‘dir’ attribute is “out”, the target will, after execution of the command, update the referenced value to reflect a result. If the value of the ‘dir’ attribute is “inout”, the target will read from the referenced variable before execution and write to it after execution of the command.

The ‘idref’ attribute is not presented to users and need not be human comprehensible.

A global parameter shall not have a ‘type’ attribute (see 10.12.4.3). Instead, its type is specified by the referenced socket variable declaration (see 8.3).

NOTE 1 On command invocation, the URC will not send or receive the values of the command's global parameters. Instead, their values will be synchronized between the URC and the target as required for socket variables (as described in ISO/IEC 24752-1), independently from the command invocation.

NOTE 2 Advanced URCs may exploit the knowledge on global parameters to infer dependencies between socket elements and thus build a more usable user interface.

10.12.4 The 'dir' attribute

A <param> element shall have a 'dir' attribute.

Its value shall be either "in", "out" or "inout". A value of "in" marks the parameter as input parameter. A value of "out" marks the parameter as output parameter. A value of "inout" specifies that the parameter is both input and output parameter, i.e. the parameter will be read before execution and updated after execution of the command.

NOTE Commands of type uis:voidCommand cannot have output or input-output parameters (see 10.3.2).

10.12.4.1 Input parameters

A dir value of "in" denotes input parameters, i.e. parameters whose value will be read by the target before execution of a command, to affect the execution and its result(s).

A command may have any number of (local and global) input parameters.

The value of a local input parameter may be changed by the URC or its user when the 'execute' dependency of the command evaluates to true or is unknown.

For global input parameters their relevant and write dependencies (see 8.11.3 and 8.11.4) specify whether they can be accessed or changed by the URC or its user.

NOTE Local input parameters are similar to socket elements, but their values are synchronized with the target only when the command is invoked, and only from the URC to the target.

10.12.4.2 Output parameters

A dir value of "out" denotes output parameters, i.e. parameters whose value is updated by the target after execution of a command, to reflect a result of the execution.

A command may have any number of (local and global) output parameters.

The value of a local output parameter shall not be changed by the URC or its user.

For global output parameters their relevant and write dependencies (see 8.11.3 and 8.11.4) specify whether they can be accessed or changed by the URC or its user.

NOTE Local output parameters are similar to socket elements, but their values are synchronized with the target only after the command's execution has completed, and only from the target to the URC.

10.12.4.3 Input-output parameters

A dir value of "inout" denotes input-output parameters, i.e. variables that are used as input and output parameters for the same command.

A command may have any number of (local and global) output parameters.

The value of a local input-output parameter may be changed by the URC or its user when the 'execute' dependency of the command evaluates to true or is unknown.

For global input-output parameters their relevant and write dependencies (see 8.11.3 and 8.11.4) specify whether they can be accessed or changed by the URC or its user.

NOTE 1 Local input-output parameters are similar to socket elements, but their values are synchronized with the target only in the following two cases: (1) when the command is invoked, from the URC to the target; and (2) after the command's execution has completed, from the target to the URC.

10.12.5 The 'type' attribute

A local parameter shall have a 'type' attribute. A global parameter shall not have a 'type' attribute.

Its value denotes the type of the parameter. Valid types are the same as for the <variable> element (see 8.3).

10.12.6 The 'secret' attribute

A local parameter may have a 'secret' attribute. Its default value is "false".

A global parameter shall not have a 'secret' attribute.

It has the same meaning as for the <variable> element (see 8.4).

10.12.7 The 'sensitive' attribute

A local parameter may have a 'sensitive' attribute. Its default value is "false".

A global parameter shall not have a 'sensitive' attribute.

It has the same meaning as for the <variable> element (see 8.5).

10.12.8 The <selection> subelement

The <selection> element may be present in a <param> element. If present, it shall occur once.

The <selection> element may provide a set of values that either restricts the parameter's value space (closed selection) or provides suggested values for user input (open selection). It follows the same syntax as defined for variables in 8.12.
EXAMPLE A media rendering device lets the user pick from a set of preset configurations. When invoking the command "selectPreset", the user can pick one of the presets given in the list variable "presetNameList".

```
<variable id="presetNameList" use="required" type="uis:csvlist">
    <dc:description>This variable contains a comma-separated list of valid preset names currently supported by this
    device. Its value changes if/when the device changes the set of presets that it supports. This may occur in
    conjunction with a vendor-defined action or some other non-UPnP event. This state variable will include any of
    the predefined presets that are supported by the device.</dc:description>
 </variable>
<command id="selectPreset" use="required" type="uis:basicCommand">
    <dc:description>Set a preset configuration.</dc:description>
    <param id="presetName" type="xsd:string" dir="in">
        <selection closed="true">
            <selectionSetDynamic id="availablePresets" varRef="presetNameList"/>
        </selection>
    </param>
</command>
```

10.12.9 Platform-specific mapping information

The <mapping> element may be used any number of times for local parameters to include platform-specific mapping information. It shall not occur for global parameters.

A <mapping> element shall have a 'platform' attribute whose value is not restricted by this International Standard.

A <mapping> element may have arbitrary element content and subelements. However, subelements shall be from namespaces other than the uis namespace.

NOTE Socket descriptions that contain platform specific mapping information lose their platform neutrality. Although multiple mappings may be specified in a socket description (one for each platform) it is recommended to consider other mechanisms of specifying the binding to platform-specific technologies. For example, mapping information may be provided in an external file with references to the elements of the socket description.

10.12.10 Properties from DCMI

Any element and element refinement from ISO 15836, Dublin Core Metadata Element Set, or the set of Dublin Core Metadata Initiative (DCMI) Metadata Terms may be used to describe a command parameter, if appropriate. Each of them may occur multiple times within a <param> element.

10.13 Properties from DCMI

Any element and element refinement from ISO 15836, Dublin Core Metadata Element Set, or the set of Dublin Core Metadata Initiative (DCMI) Metadata Terms may be used to describe a socket command, if appropriate. Each of them may occur multiple times within a <command> element.

11 Notifications

11.1 General

Notifications are special states where normal operation is suspended, such as exception states. A target may invoke (activate) or dismiss (deactivate) a notification at any time.

EXAMPLE Examples of notifications include an announcement made by a public address system in an airport, a clock alarm, or a response to invalid input for a field of a form.

A notification may occur as subelement of a <set> element (see 7) or as subelement of the <uiSocket> element (see 6.1).

A notify element is defined with the markup:

<notify> … </notify>

A notify element has one of three states: active, inactive, or stacked. The default state is inactive. A target may activate or deactivate the notify element at any time. Some variables and commands may be only readable and writable when a notification is active, as expressed in their relevant, write and execute dependencies and described in sections 8.11.3, 8.11.4, 10.10.2 and 10.10.4.

A target may activate a notification while another notification is already active. In this case, the second notification takes priority over the first. The user agent shall keep a stack of notifications with the most recent on top. All notifications that are not on the top are deemed to be in state “stacked” (this is internal to the user agent, and not shared with the target). User interfaces based on a user interface socket shall respect the order in which notifications become active, so that the most recently activated notification is always the one presented to the user. When the top notification becomes inactive, the next notification on the stack becomes active again and is presented to the user.

When a notification is used to indicate to the user that a timeout has occurred, the notification should provide an option for the user to request more time.

11.2 The ‘id’ attribute

A <notify> element shall have an ‘id’ attribute.

