Solution Deployment Descriptor Specification Version 2.0

Committee Specification Draft 01 / Public Review Draft 01

10 November 2010

Specification URIs:
This Version:
> http://docs.oasis-open.org/sdd/sdd/v2.0/csprd01/sdd-v2.0-csprd01.html
> http://docs.oasis-open.org/sdd/sdd/v2.0/csprd01/sdd-v2.0-csprd01.doc (Authoritative)
> http://docs.oasis-open.org/sdd/sdd/v2.0/csprd01/sdd-v2.0-csprd01.pdf

Previous Version:
> http://docs.oasis-open.org/sdd/v1.0/os/sdd-spec-v1.0-os.html
> http://docs.oasis-open.org/sdd/v1.0/os/sdd-spec-v1.0-os.doc (Authoritative)
> http://docs.oasis-open.org/sdd/v1.0/os/sdd-spec-v1.0-os.pdf

Latest Version:
> http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-v2.0.html
> http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-v2.0.doc (Authoritative)
> http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-v2.0.pdf

Technical Committee:
> OASIS Solution Deployment Descriptor (SDD) TC

Chair(s):
> Brent Miller, IBM Corporation

Editor(s):
> Merri Jensen, SAS Institute, Inc.

Related work:
> This specification replaces or supercedes:
> - Solution Deployment Descriptor Specification 1.0 OASIS Standard

Declared XML Namespace(s):
> sdd-common=http://docs.oasis-open.org/sdd/ns/common
> sdd-pd=http://docs.oasis-open.org/sdd/ns/packageDescriptor
> sdd-dd=http://docs.oasis-open.org/sdd/ns/deploymentDescriptor

Abstract:
> This specification defines schema for two XML document types: *Package Descriptors* and *Deployment Descriptors*. Package Descriptors define characteristics of a package used to deploy a solution. Deployment Descriptors define characteristics of the content of a solution package, including the requirements that are relevant for creation, configuration and maintenance of the solution content. The semantics of the descriptors are fully defined, allowing software implementations to precisely understand the intent of the descriptor authors and to use the information provided in the descriptors to support solution deployment.

Status:

The OASIS Solution Deployment Descriptor (SDD) Technical Committee approved Working Draft 01 as a Committee Specification Draft on 26 October 2010. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/sdd/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/sdd/ipr.php.

Citation Format:

When referencing this specification the following citation format should be used:

SDD2.0   OASIS Committee Specification Draft 01, *Solution Deployment Descriptor Specification Version 2.0*, 26 October 2010. http://docs.oasis-open.org/sdd/sdd/v2.0/csd01/sdd-v2.0-csd01.doc

Notices

Table of Contents

1 Introduction

The *Solution Deployment Descriptor* (SDD) specification defines a standard, in the form of a schema for XML documents, called *Solution Deployment Descriptors*, or *SDDs*. SDDs define metadata that describes the packaging and deployment characteristics of resources that are relevant for their lifecycle management, including creation, configuration and maintenance.

1.1 Terminology

The following terms are used in this specification in a specialized sense that might differ from definitions elsewhere.

Artifact

Zero or more files and/or metadata used to perform a *deployment lifecycle* operation on a *resource*.

Deployment lifecycle

The stages marking maturation of a *solution*: develop, package, integrate, manufacture, install, configure, evaluate, deploy into production, upgrade and/or update, uninstall.

Host Resource

A resource that provides the execution environment for another resource.

Package

A set of artifacts used to perform *deployment lifecycle* operations on a group of related resources that make up a solution.

Resource

A particular element of a computing environment, such as a computer system, an operating system, a Web server, a software application, or a complex *solution*.

Solution

One or more interrelated *resources* on which *deployment lifecycle* operations can be performed.

Target Resource

A resource that processes *artifacts* to perform *deployment lifecycle* operations on another resource. The *host resource* often serves as the target resource.

Topology

The physical or logical layout of a *solution's resources*.

Update (n.)

A *package* that replaces a limited set of the *resources* in a *solution* instance. An update does not require migration.

Upgrade (n.)

A *package* that replaces all, or a significant portion of, the *resources* used in a *solution*. An upgrade might or might not require migration.

1.2 Purpose

The purpose of this document is to provide the normative specification of the SDD, including concepts, structure, syntax, semantics and usage.

1.3 Scope

This document is the specification for the SDD. It consists of both normative and non-normative prose, diagrams, schema and examples. The document is intended to facilitate an understanding of the SDD concepts, structure, syntax, semantics and usage. This document is not intended to be a tutorial.

This document is the full SDD specification, but it also is augmented with other documents produced by the SDD TC, including the SDD XML Schema and Examples (see Appendix [A]), **[SDDP]**, **[SDDSP]** and the set of SDD profiles (see section [5.3]), as well as documents produced by others (see section [5.3.1]).

1.4 Audience

This document is intended to assist those who require an understanding of the nature and details of the SDD. This includes architects, developers, solution integrators and service/support personnel who generate, consume, or otherwise use SDDs, as well as those who develop tooling and applications for constructing and deploying SDDs.

1.5 How to Read this Document

The various audiences of this specification might have different objectives and purposes when reading the document. You might wish to generally understand the SDD, or learn the details of the SDD to create or consume SDDs, or use the document as a reference.

- If your purpose is to understand the major capabilities and characteristics of the SDD and how they fit together, start by reading the Introductions to the major sections: [3], [4] and [4.1]–[4.14].

- If your purpose is to understand the major elements of the SDD and how they work together to accomplish the goals of this specification, read in addition to the above, the introductions to each of the type sections [3.1]–[3.13] and the type subsections within sections [4.2]–[4.14].

- If your purpose is to understand the syntax of the SDD, look at the tables in each of the Property Summary sections.

- If your purpose is to understand the semantics of the elements and attributes of the SDD, read the Property Usage Notes sections.

- If your purpose is to understand only the package descriptor, subset the above suggestions to focus on the sub-sections within section [3].

- If your purpose is to understand only the deployment descriptor, subset the above suggestions to focus on the sub-sections within section [4].

1.6 Motivation

The motivation for producing this specification is best expressed in this excerpt from the SDD Technical Committee's charter:

> Deployment and lifecycle management of a set of interrelated software, hereinafter referred to as a solution, is a predominantly manual operation because there is currently no standardized way to express installation packaging for a multi-platform environment. Each hosting platform or operating system has its own format for expressing packaging of a single installable unit but, even on these homogeneous platforms, there is no standardized way to combine packages into a single aggregated unit without significant re-creation of the dependency and installation instructions. The problem is compounded when the solution is to be deployed across multiple, heterogeneous, platforms. A standard for describing the packaging and mechanism to express dependencies and various lifecycle management operations within the package would alleviate these problems and subsequently enable automation of these highly manual and error-prone tasks.

> The purpose of this Technical Committee is to define XML schema to describe the characteristics of an installable unit (IU) of software that are relevant for core aspects of its deployment, configuration and maintenance. This document will be referred to as the Solution Deployment Descriptor (SDD).

86 *SDDs will benefit member companies and the industry in general by providing a consistent model*
87 *and semantics to address the needs of all aspects of the IT industry dealing with software*
88 *deployment, configuration and lifecycle management. The benefits of this work include:*

89 • *ability to describe software solution packages for both single and multi-platform*
90 *heterogeneous environments.*

91 • *ability to describe software solution packages independent of the software installation*
92 *technology or supplier.*

93 • *ability to provide information necessary to permit full lifecycle maintenance of software*
94 *solutions.*

1.7 Requirements

96 A summary of requirements satisfied by this SDD specification follows. Detailed requirements that
97 support approved use cases are available at the SDD TC Web page, http://www.oasis-
98 open.org/committees/sdd.

Solution lifecycle management

100 The SDD must provide information to support the complete lifecycle of a software solution.
101 Certain key requirements are applicable to all phases of deployment lifecycle operation: planning,
102 installation, configuration, maintenance, upgrade, migration and uninstallation.

Solution requirements for environment to perform lifecycle
management tasks

105 A deployment lifecycle operation on a target resource is often dependent on a certain set of
106 conditions that must exist on the target. This set of pre-existing conditions is known as the
107 *environment*. If successful deployment lifecycle operations are dependent on a certain set of pre-
108 existing conditions (environment), then the SDD specification must support the ability to specify
109 the required environment.

Projected changes to environment

111 The SDD specification must support the definition of environment changes that become effective
112 once the lifecycle operation is complete.

Solution instance variability

114 The SDD specification must support the definition of the appropriate information for a runtime to
115 vary the ways in which the solution can be deployed. This information is also needed to enable an
116 integrator to control the variability according to the needs of their higher-level solution.

117 This variability includes the information to control (1) the subset of capability that can be
118 deployed; (2) setting the initial configuration of the solution; and (3) varying the topology in which
119 the solution can be deployed.

Solution composition

121 The SDD specification must support the ability for the author to compose solution packages from
122 multiple components, products, or solutions.

Solution and packaging identity

124 The SDD specification must support the definition of identity information for the solution package,
125 resources that make up the solution, and solution itself to support use cases including asset
126 management, license management, support/update entitlement, component reuse during
127 development, reports and queries from a package repository, identifying associated
128 documentation, solution lifecycle management, traceability to build/development environment and
129 problem management systems, correlation into the hosting environment, component reuse, and
130 maintenance history. Also, the SDD specification must support the definition of the identity
131 description information used by a runtime to assist a user in making correct decisions about
132 solution installation. The SDD specification must support the definition of the information that

| 133 | uniquely identifies the SDD descriptor and the ability to identify the version of the SDD. The |
| 134 | customer should be able to identify the solution packages with consistent names. |

Physical packaging

136 Physical packaging information should be contained in a separate media descriptor. The
137 deployment model for a solution should be decoupled from the details of physical packaging. The
138 format and structure of the physical packaging is outside the scope of SDD v1.0-v2.0.

Interoperability with existing software packaging technologies

140 The SDD specification must support the ability for the author to compose solutions from existing
141 software packages that do not have an SDD. This means that the SDD should be able to
142 describe existing software packages.

Conform to external standards

144 The SDD specification must provide for alternative descriptive text to be defined for any images,
145 animations, or audio information contained in the descriptor.

Decision support

147 Requirements to perform lifecycle management operations within various target environments
148 may not be satisfied in the target's current state but might be able to be satisfied with additional
149 operations. For example, successful deployment of a set of Java™[1] components is dependent on
150 the existence of a Java runtime environment that is not included with the solution. The SDD
151 should have the ability to specify information that will assist lifecycle management tools in
152 planning for, accessing and installing these external requirements.

Specification organization

154 The SDD specification must provide the semantic behavior expected by producers and
155 consumers of SDDs. This information allows for the producers to ensure that the consumers of
156 their SDDs will provide the support intended.

Solution metadata

158 The SDD metadata may not encompass all of the information about the solution in all contexts in
159 which the solution can be deployed. Additional metadata that is outside of the scope of the SDD
160 is available at the SDD TC Web page, http://www.oasis-open.org/committees/sdd.

Globalization

162 For all content in the SDD that would be displayed to a user, the specification must support the
163 definition of strings for multiple locales; for example, this content must be localizable.

Align with other standards bodies

165 Satisfying all the requirements listed here calls for extensive standardization in specific areas.
166 The requirements should thus be aligned with other appropriate standards bodies. The SDD
167 reuses existing OASIS and other standards where appropriate and aligns with other standards
168 bodies (for example, **[OGF-ACS]**) that are developing standards in the same domain as SDD.

---

[1] Java is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

1.8 XML Namespaces

170 The XML namespaces defined as part of this specification are:

171 ▪ **sdd-pd**: stands for the package descriptor portion of the SDD namespace.

172 ▪ **sdd-dd**: stands for the deployment descriptor portion of the SDD namespace.

173 ▪ **sdd-common**: stands for the common (shared) types, elements and groups of the SDD namespace.

174 For XML namespaces not defined as part of this specification, conventional XML namespace prefixes are
175 used as follows, regardless of whether a namespace declaration is present in the example:

176 ▪ The prefix **xsd:** stands for the W3C XML Schema namespace **[XSD]**.

177 ▪ The prefix **ds:** stands for the digital signature namespace **[XMLDSIG-CORE]**.

1.9 Notational Conventions

179 Everything in the specification, including the Appendices, is considered normative except for the abstract,
180 examples and any sections or other material marked as non-normative.

181 The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
182 NOT", "RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described
183 in **[RFC2119]**.

184 These keywords are capitalized when used unambiguously to specify requirements or application
185 features and behavior. When these words are not capitalized, they are meant in their natural-language
186 sense.

1.10 General Document Conventions

188 In describing XML elements and attributes of the SDD schema, this document contains many cross-
189 references. Such references appear as the referenced section number inside square brackets, for
190 example, [4.5]. In electronic versions of this specification, the cross-references can act as links the target
191 section.

192 The following property naming convention is used in the schema: Element and type names begin with an
193 uppercase letter and attribute names begin with a lowercase letter.

194 Italics are used to identify element and attribute names, type names and enumerated values defined by
195 an SDD type.

196 In describing the XML schema, each section typically contains the following subsections:

197 ▪ A diagram illustrating the element, group, or type that is specified in the section.

198 ▪ Property Summary: A table listing the schema elements and attributes, along with the data type,
199 cardinality and description for each one.

200 When specified, extension points are listed in the tables with no name and a type of `xsd:any` for
201 element extensions and `xsd:anyAttribute` for attribute extensions. Cardinality is also provided.

202 When a type is an extension of another type, the extended type is listed in the table with no name and
203 prefixed with **[extends]**. The extended type's properties can be referenced from the appropriate
204 section listed in the description column.

205 When the schema specifies a default or fixed attribute value, that value is prefixed with two asterisks,
206 as in ****default value="true"**.

207 ▪ Property Usage Notes: A list of the elements and attributes, along with more detailed prose
208 descriptions of the properties and how they fit into the schema as a whole.

209 ▪ Not all sections contain every one of the preceding subsections.

1.11 Diagram Conventions

211 Sections 3 and 4 of this specification contain diagrams that illustrate the structure of elements, data types
212 and groups used throughout the SDD schema. Figure 1 is an example of this type of diagram.

213
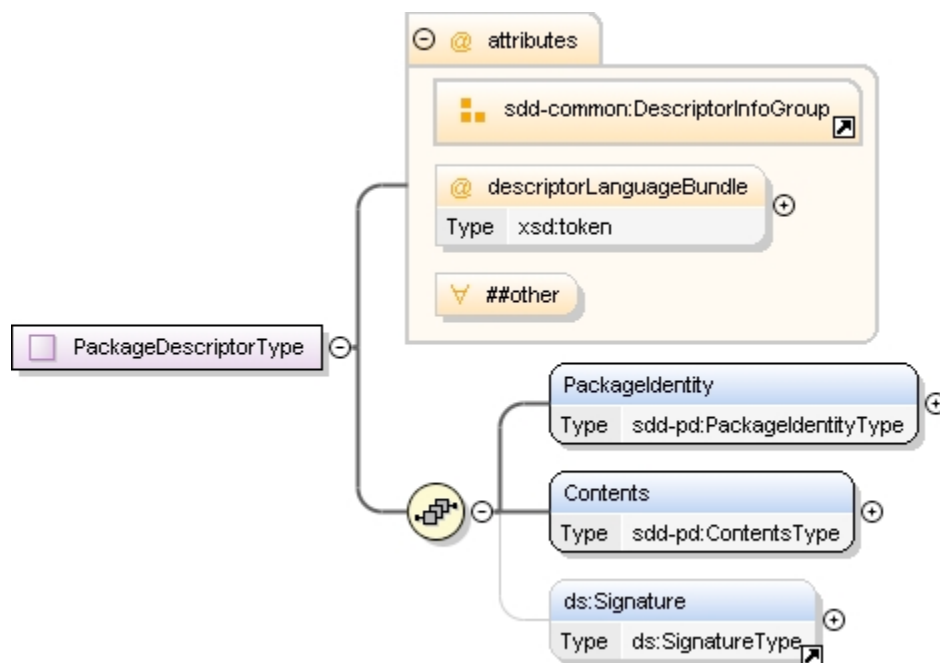
Figure 1: Sample XML structure diagram.

214

215 Elements are represented with the element name in a blue shaded rectangle with four rounded corners.
216 In Figure 1, the elements are *PackageIdentity*, *Contents*, and *ds:Signature*.

217 Attributes are represented within a yellow shaded tab and are individually contained in yellow shaded
218 rectangles with two square corners on the left side and two rounded corners on the right side, and the
219 attribute name is preceeded by a "@" symbol.

220
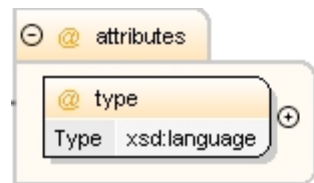
221 For both elements and attributes, a solid black outline around the rectangle indicates that the element or
222 attribute is required by the schema; whereas a grey outline indicates that the element or attribute is
223 optional.

224 References to elements or attributes defined in a separate namespace are represented by a small arrow
225 in the lower right corner of the rectangle:

226
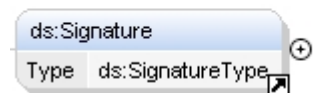
227 Complex types are represented by a purple shaded rectangle with four squared corners and with a purple
228 square followed by the element name:

229
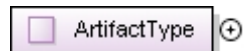
230 Simple types are represented by a purple shaded rectangle with all the corners truncated and with a
231 purple triangle followed by the element name:

232
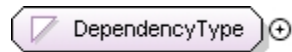
233 A purple shaded tab represents an element type that is extended by the element shown in the figure.

234 Groups are represented by a rectangle with three squared corners and a rounded upper right corner and
235 with three small squares followed by the group name. Element groups are shaded blue and attribute
236 groups are shaded yellow.

237

238 A plus sign on the right border of a component indicates hidden child elements or attributes. When
239 hidden, the child elements are usually described in a separate section.

240 There are two connectors (or compositors) used in the SDD schema diagrams to combine elements:

241 ▪ A sequence of elements is indicated by the following symbol:

242 ▪ A choice among elements is indicated by the following symbol:

243 Where appropriate, the cardinality of an element is indicated by a rectangle with the cardinality listed to
244 the left, using the form "*min..max*". For example, "1..∞" indicates a minimum of one occurrence of the
245 element and an unbounded upper limit:
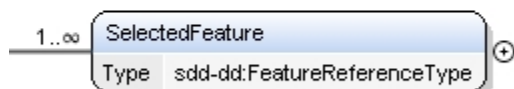
246

247 When cardinality is present to the left of a connector, the sequence or choice that follows is constrained
248 and may have restrictions on the number of times it may be repeated within the element.
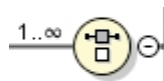
249

250 All XSD schema figures were created with <oXygen/> XML Editor, v10.2.

1.12 Normative References

252	**[CL2_Schema]**	Solution Deployment Descriptor Schema
253		See Appendix [A] for location.
254	**[CONFORM]**	OASIS, *OASIS Conformance Requirements for Specifications 1.0*,
255		http://www.oasis-
256		open.org/committees/download.php/305/conformance_requirements-v1.pdf.
257	**[IANA-CHARSET]**	Internet Assigned Numbers Authority, *Character Sets*,
258		http://www.iana.org/assignments/character-sets, modified December 2006.
259	**[IETF-UUID]**	Internet Engineering Task Force Draft Specification,
260		http://www.ietf.org/rfc/rfc4122.txt.
261	**[ISO639.2]**	Library of Congress, *Codes for the Representation of Names of Languages*,
262		http://www.loc.gov/standards/iso639-2/englangn.html.
263	**[ISO3166]**	International Organization for Standardization, *English Country Names and Code*
264		*Elements*, http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-
265		lists/list-en1.html.
266	**[RFC2119]**	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
267		http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
268	**[RFC3066]**	H. Alvestrand, ed. *RFC 3066: Tags for the Identification of Languages* 1995,
269		http://www.ietf.org/rfc/rfc3066.txt.
270	**[UNIT]**	Bureau International des Poids et Mesures, http://www.bipm.fr.
271	**[XMLDSIG-CORE]**	Bartel et al., *XML-Signature Syntax and Processing*,
272		http://www.w3.org/TR/xmldsig-core/, W3C Recommendation, February 2002.
273	**[XSD]**	W3C Schema Working Group, *XML Schema*, http://www.w3.org/TR/xmlschema-
274		1/, W3C Recommendation, October 2004.
275		

1.13 Non-Normative References

277	**[CL1_Schema]**	Solution Deployment Descriptor Conformance Level 1 Schema
278		See Appendix [A] for location.
279	**[CIM]**	Distributed Management Task Force, Inc., Common Information Model (CIM)
280		http://www.dmtf.org/standards/cim/.
281	**[OGF-ACS]**	Open Grid Forum, Application Contents Service WG (ACS-WG),
282		http://www.ogf.org/gf/group_info/view.php?group=acs-wg.
283	**[SDDEX]**	Solution Deployment Descriptor Examples
284		See Appendix [A] for location.
285	**[SDDP]**	Solution Deployment Descriptor Primer
286		See Appendix [A] for location.
287	**[SDDSP]**	Solution Deployment Descriptor Starter Profile
288		See Appendix [A] for location.
289		

2 Solution Deployment Descriptor Overview

2.1 Package and Deployment Descriptors

The package descriptor defines package content which includes artifacts whose processing results in deployment of the software package. The deployment descriptor defines metadata associated with those artifacts. The SDD package descriptor defines the package identity, the package content and various other attributes of the package. Each SDD consists of exactly one deployment descriptor and one package descriptor. The deployment descriptor is where the topology, selectability, inputs, requirements and conditions of the deployment are described.

2.2 Topology

The SDD's topology describes all the resources that may be required, created or modified when any of the deployment operations supported by the SDD are performed.

Primary identifying characteristics of the resources can be defined in topology. The topology includes identification of hosts–hosted by relationships between resources. It is usual that only a subset of the resources described in topology will play a role in any particular deployment. This is determined by the selection of content elements for the particular deployment. The resources that are required, created or modified by the content elements in scope for the deployment are the ones that will participate in the deployment and so will be associated with resources in the deployment environment.

At deployment time, definitions of the resources that participate in that particular deployment are associated with actual resource instances in the deployment environment. The mechanism for associating resource definitions with resource instances is not defined by the SDD.

The only resource definitions in the SDD are in topology. All other mention of resources in the SDD are references to the resource definitions in the topology.

2.3 Content and Artifacts

Metadata throughout the deployment descriptor is associated with package content in the definition of atomic content elements. The atomic content elements are *InstallableUnit*, *ConfigurationUnit* and *LocalizationUnit*. These are the only content elements that define *Artifacts* elements.

Artifact elements identify an artifact file or set of files defined in package content whose processing will perform all or a portion of the deployment for a particular deployment lifecycle operation. Artifact elements define the inputs and outputs, substitution values and types associated with the artifact files. The content element's target resource, identified by *targetResourceRef*, processes the artifact files with the defined inputs to perform deployment operations. Examples of artifact types include zip files, rpm files and executable install files. Artifact types are not defined by this specification. The artifact types defined in the SDD need to be understood by software that processes the SDD. *Profiles* are used to communicate the artifact types that an implementation is capable of processing [5.3].

Composite content elements organize the content of an SDD but do not define artifacts used to deploy SDD content. There are three types of composite content elements: *CompositeInstallable*, *CompositeUnit* and *CompositeLocalizationUnit*.

CompositeInstallable is used any time that more than one content element is defined in support of one operation on the package; any time aggregation of SDDs is needed; or any time the package includes selectable content. *CompositeInstallable* is the root of a content hierarchy that supports a single deployment lifecycle operation. It can define a base content hierarchy, a localization content hierarchy and a selectable content hierarchy that includes selection criteria. One SDD can have more than one *CompositeInstallable*–each supporting a different operation.

CompositeUnit is used to organize content elements within the base or selectable content hierarchies. *CompositeUnits* can define *InstallableUnits*, *ConfigurationUnits*, *ContainedPackages* and other

335   *CompositeUnits*. Requirements, conditions and variables that are common to all content elements defined
336   by the *CompositeUnit* can be defined in the *CompositeUnit* to avoid repetition. Within the selectable
337   content hierarchy, a *CompositeUnit* can provide an efficient means for selection of a set of related content
338   elements by a *feature.*

339   *CompositeLocalizationUnit* serves the same purposes as *CompositeUnit* within the *LocalizatonContent*
340   hierarchy.

341   SDD packages can aggregate other SDD packages. Metadata about the aggregation is defined in
342   *ContainedPackage*, *ContainedLocalizationPackage* and *Requisite* elements. *ContainedPackage*
343   elements are a content element that can be defined anywhere in the base and selectable content
344   hierarchies. *ContainedLocalizationPackages* are content elements that can be defined in the localization
345   content hierarchy. *Requisites* are packages that can be deployed, if necessary, to satisfy requirements in
346   the aggregating SDD. They are not content of the SDD package. The type of all three of these elements
347   is *ReferencedPackageType*. The term "referenced package" is used in this specification when referring to
348   these elements as a group. The term "referenced SDD" is used when referring to any aggregated SDD.

349   Each referenced package element can further constrain the deployment of the referenced SDD by
350   defining additional requirements; by mapping resources defined in the aggregating SDD to those defined
351   in the referenced SDD; and by determining feature selections for deployment of the referenced SDD.

2.4 Resulting and Changed Resources
352

353   Deployment of an SDD package creates or modifies software resources. These resources are included in
354   the topology definition and described in more detail in *ResultingResource* and *ResultingChange*
355   elements.

356   The SDD author can choose to model resulting and modified resources at a very granular level, at a very
357   coarse level; at any level in between, or not at all. An example of modeling resulting resources at a
358   granular level would be modeling every file created by the deployment as a resulting resource. An
359   example of modeling resulting resources at a very coarse level would be modeling the software product
360   created by deployment as a single resulting resource. The choice depends on the needs of the solution
361   deployment. If a resource is not modeled in the SDD, no requirements can be expressed on it, no
362   conditions can be based on it and no variables can be set from values of its properties. It cannot play any
363   of the roles described for resources in the *ResourceType* section of this document [4.2.2].

2.5 Base, Selectable and Localization Content Hierarchies
364

365   Each *CompositeInstallable* element can define three types of content hierarchies. Base content is the
366   default content for the deployment lifecycle operation associated with the *CompositeInstallable*. This is
367   content that will be deployed whenever the associated operation is performed on the SDD package. Base
368   content may be conditioned on characteristics of the deployment environment but it is not selectable by
369   the deployer.

370   The SDD author can define selectable subsets of optional content in the selectable content hierarchy.
371   The selection criteria include features and groups of features that select content from the selectable
372   content hierarchy. Selectability, as used in the SDD, is a characteristic of the deployment lifecycle
373   operation and the package. For example, the decision to provide selectability for one operation in one
374   package has no semantic relationship to the selectability provided in another package related to the same
375   software. It also has no semantic relationship to the selectability provided for a different operation within
376   the same package.

377   Localization content is the third type of content hierarchy. Localization refers to enabling a particular piece
378   of software for support for one or more languages. Anything that needs to be deployed to provide support
379   for a particular language in that software is considered localization content. Translated materials are a
380   primary, but not the only, example of localization content.

381   Localization content is similar in many ways to other content, but there are important differences in how
382   localization content is selected for deployment that lead to the need for a separate content hierarchy and
383   separate types. There are two criteria for determining that localization content is in scope for a particular
384   deployment.

385 ▪ The first criterion has to do with the language or languages supported by the localization content. At
386   least one of the languages must be in scope for the content to be selected.
387 ▪ The second criterion has to do with the availability of the resources to be localized–the localization
388   base. The localization base may be a resource deployed by base or selectable content, or it may be
389   a resource previously deployed and found in the deployment environment.

2.6 Constraints

391 The SDD author needs to communicate constraints on resources for a variety of purposes.

392 • Some constraints must be met for the requirements of a content element to be met.

393 • Other constraints must be met for a resource to serve as the required base for an update.

394 • Still others must be met to satisfy a condition that determines the applicability of a content element or
395   completion action.

396 The Constraint types are:

397 ▪ *CapacityConstraint*

398 ▪ *ConsumptionConstraint*

399 ▪ *PropertyConstraint*

400 ▪ *VersionConstraint*

401 ▪ *UniquenessConstraint*

402 ▪ *RelationshipConstraint*

403 ▪ *AuthorizationConstraint*

2.7 Requirements

405 A Requirement is an environmental necessity that a resource must have fulfilled in order for an artifact to
406 be deployed successfully into that environment. Requirements are defined by content elements. A
407 requirement consists of resource constraints that the SDD author states MUST be met prior to successful
408 deployment or use of the software described by the SDD package. Each requirement definition lists one
409 or more deployment lifecycle operations to which the requirement applies. When the requirement is
410 specified in an atomic content element, the operation associates the requirement with artifacts within the
411 atomic content element.

412 When a requirement can be satisfied in more than one way, alternatives can be defined within a
413 requirement. A requirement is considered met when any one of the alternatives is satisfied.

2.8 Conditions

415 Conditions are expressed on characteristics of resources in the deployment environment. Conditions are
416 used to indicate when particular elements of the SDD are applicable, or when they should be ignored.
417 Conditions are not requirements. Failure to satisfy a condition does not indicate a failure; it simply means
418 the conditioned element should be ignored. Conditions are used to:

419 ▪ determine if a content element is applicable

420 ▪ choose from among values for a variable

421 ▪ determine when a feature is applicable

422 ▪ determine when a particular result is applicable

423 ▪ determine if a particular completion action is necessary.

424 Because conditions are always based on the characteristics of resources, they are expressed using
425 resource constraints.

426 ## 2.9 Variables

427 Variables provide a way to associate user inputs, resource property values, fixed strings and values
428 derived from these with input arguments for artifacts and with constraints on resources.

3 Package Descriptor

A package descriptor is an XML document that provides information about the identity and the contents of a software package. A software package is a bundle of one or several content elements that deploy or remove computer software; add features to existing software; or apply maintenance to existing software. Each package descriptor is associated with a deployment descriptor.

3.1 PackageDescriptor
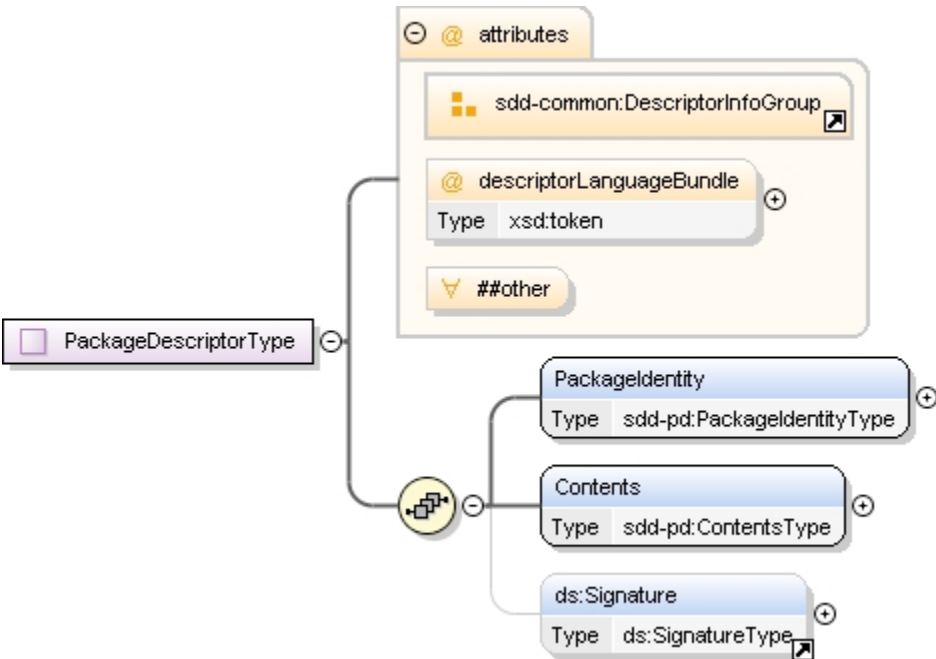
Figure 2: PackageDescriptor structure.

The root element of a package descriptor XML document is *PackageDescriptor*. *PackageDescriptor* includes elements that describe the package identity and the contents that make up the package. The *PackageDescriptor* includes the associated deployment descriptor XML document by defining a *Content* element with a *purpose* attribute set to *deploymentDescriptor*.

3.1.1 PackageDescriptor Property Summary

Name	Data Type	*	Description
PackageIdentity	PackageIdentityType	1	Human-understandable identity information for the software package.
Contents	ContentsType	1	A list of package contents.
ds:Signature	ds:SignatureType	0..1	A signature for the package descriptor.
schemaVersion	xsd:string	1	The descriptor complies with this version of the Solution Deployment Descriptor Specification. **fixed value="2.0"
descriptorID	UUIDType	1	Identifier of a particular package's descriptor.
lastModified	xsd:dateTime	1	The time the descriptor was last modified.

descriptorLanguageBundle	xsd:token	0..1	The root name of language bundle files containing translations for display text elements in the PackageDescriptor.
	xsd:anyAttribute	0..*	

3.1.2 PackageDescriptor Property Usage Notes

- **PackageIdentity**: The *PackageIdentity* element provides identity information about the software package that can be used by the consumer of the package for deployment planning or aggregation of the package into a larger solution.

  See the *PackageIdentityType* section for structure and additional usage details [3.3].

- **Contents**: The *Contents* element defines a list of one or more *Content* elements describing all the files that are part of the package. All files in the package MUST be defined in *Contents*.

  See the *ContentsType* section for structure and additional usage details [3.11].

- **ds:Signature**: The package descriptor and each file in the package MAY be digitally signed. It is RECOMMENDED that they be digitally signed by using an XML-Signature **[XMLDSIG-CORE]**.

  The signature element is an enveloped signature over the SDD package. Note that each *Content* element included in the package is digitally signed indirectly via this digest. Files can also be individually signed in the *Content* element.

- **descriptorLanguageBundle**: Language translations for elements of *DisplayTextType* in the descriptor MAY be included in the solution package. Note that these are not translations for the software deployed by the package, but rather translations only for the text in the descriptors themselves. The root name of the files containing these translations MUST be specified in the *descriptorLanguageBundle* attribute, which is an instance of xsd:token. Language bundles are associated with specific locales at run time using Java-style resource bundle resolution; that is, the bundle filenames SHOULD take the form *languageBundle_locale*, where *locale* consists of optional language, location (country) and variant codes, separated by an underscore character. Language codes consist of two lowercase letters **[ISO639.2]** and location codes consist of two uppercase letters **[ISO3166]**.

  For example, if *descriptorLanguageBundle* is set to "SampleStrings", then "SampleStrings_en_US" refers to the United States English version of the SampleStrings bundle and "SampleStrings_ja" identifies the Japanese version of the same bundle.

  See the *DisplayTextType* section for structure and additional usage details [4.14.3].

- **schemaVersion**, **descriptorID**, **lastModified**: See the *DescriptorInfoGroup* section for structure and additional usage details [3.2].

3.2 DescriptorInfoGroup
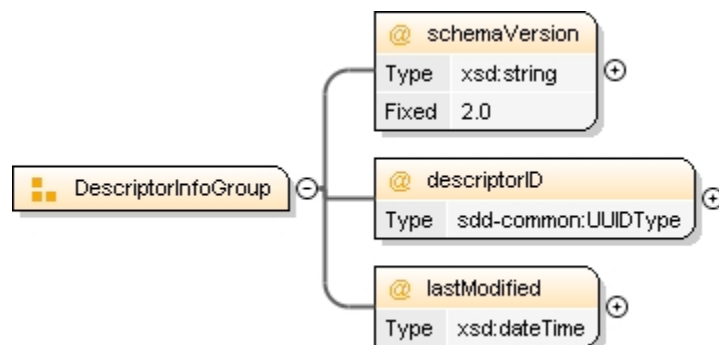
Figure 3: DescriptorInfoGroup structure.

The attributes defined by *DescriptorInfoGroup* are included in both *PackageDescriptor* and *DeploymentDescriptor*.

3.2.1 DescriptorInfoGroup Property Usage Notes

476

477   ▪   **schemaVersion**: The *schemaVersion* attribute identifies the Solution Deployment Descriptor
478   specification version to which the descriptor conforms. It MUST have a fixed value of "2.0".

479   ▪   **descriptorID**: The *descriptorID* attribute, combined with the *lastModified* attribute value, provides a
480   unique identifier for the descriptor. The *descriptorID* value MUST be unique within the scope of use of
481   the deployment descriptor or package descriptor. The *descriptorID* attribute is an instance of
482   *UUIDType*, which is based on `xsd:hexBinary` with length 16. This enables use of a 128-bit UUID
483   **[IETF-UUID]**. The *descriptorID* value supports descriptor updates by allowing updated descriptors to
484   be correctly associated with an earlier version of the same descriptor.

485   For example, if a descriptor contains errors, it may be replaced by an error-free version using the
486   same *descriptorID* value but a different *lastModified* value.

487   ▪   **lastModified**: The *lastModified* value can be used to differentiate between different versions of the
488   same descriptor, for example, the descriptor for one particular package. Comparison of *lastModified*
489   values can be used to determine which descriptor is newer.

490   The *lastModified* attribute MUST be defined as a value that conforms to the `xsd:dateTime` type as
491   defined in **[XSD]** and MUST match the following lexical representation: `[-]CCYY-MM-`
492   `DDThh:mm:ss[Z|(+|-)hh:mm]`. This is a combination of a complete date and time of day, where
493   the time zone can be specified as Z (UTC) or (+|-)hh:mm.

494   For example, the following are valid values for the *lastModified* attribute:

495   ▪   2001-10-26T21:32:52

496   ▪   2001-10-26T21:32:52+02:00

497   ▪   2001-10-26T19:32:52Z

498   ▪   2001-10-26T19:32:52+00:00

499   ▪   -2001-10-26T21:32:52

500   ▪   2001-10-26T21:32:52.12679

501   However, the following values would be invalid:

502   ▪   2001-10-26

503   ▪   2001-10-26T21:32

504   ▪   01-10-26T21:32

505   ▪   2001-10-26T25:32:52+02:00

506   The first three invalid examples do not specify all the required parts, and the fourth includes an
507   out of range hours part, "25".

508

3.3 PackageIdentityType
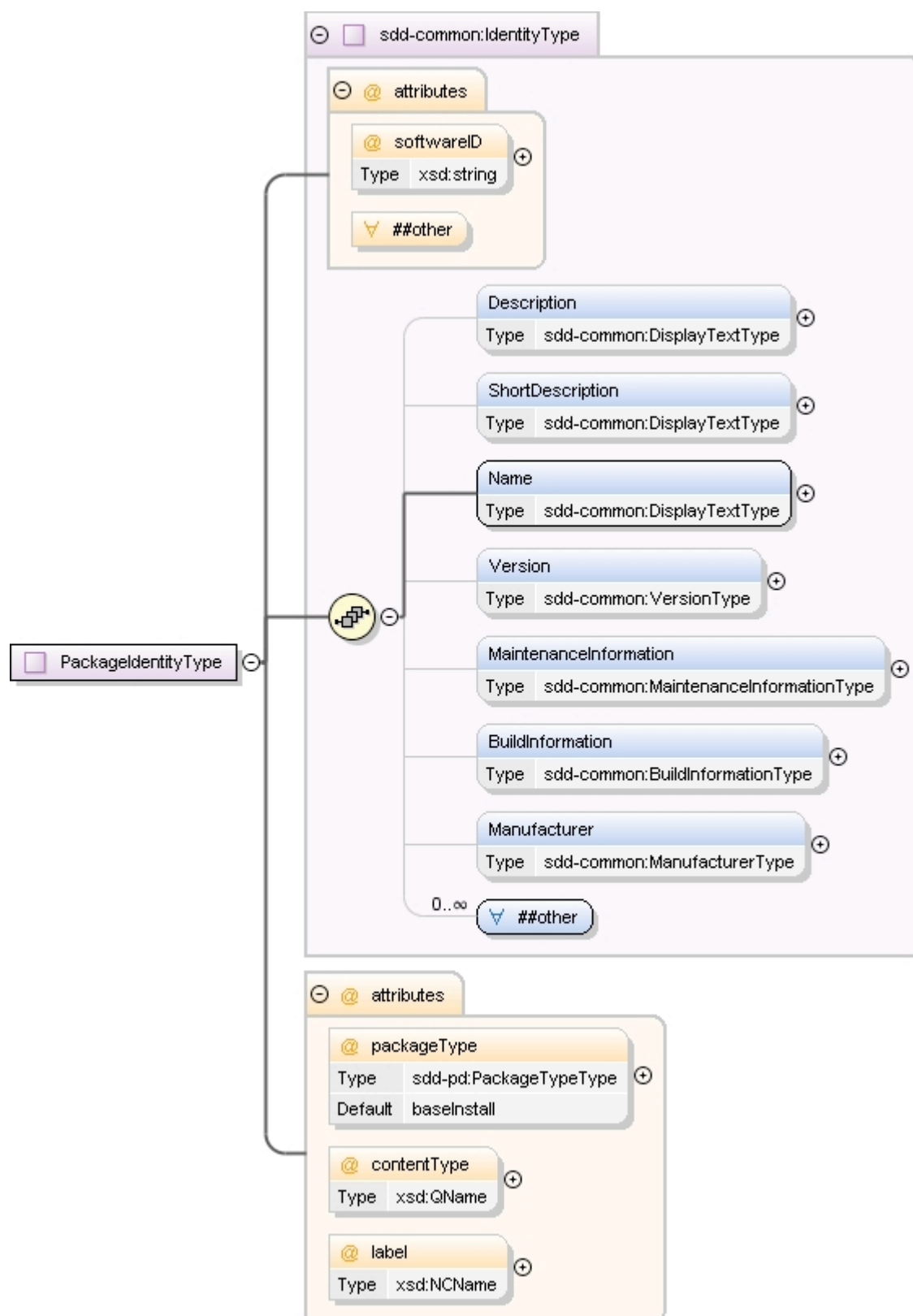
Figure 4: PackageIdentityType structure.

512 The software package described by the SDD can be identified for humans and package management
513 software using the properties in *PackageIdentity*. The *PackageIdentity* is not to be confused with the
514 identity of the deployed software, which is described in the resulting resource elements of the deployment
515 descriptor; see the *ResultingResourceType* section [4.8.1].

3.3.1 PackageIdentityType Property Summary
516

Name	Data Type	*	Description
	[extends] IdentityType		See the IdentityType section for additional properties [3.4].
packageType	PackageTypeType	0..1	The type of the package, for example, "baseInstall" or "maintenance". **default value="baseInstall".
contentType	xsd:QName	0..1	The type of content provided by this package, for example, BIOS.
label	xsd:NCName	0..1	A programmatic label for this package.
	xsd:anyAttribute	0..*	

3.3.2 PackageIdentityType Property Usage Notes
517

518 See the *IdentityType* section for details about the inherited attributes and elements [3.4].

519 ▪ **packageType**: The package type is provided to aid consumer understanding of the type of content
520 contained in the package. A package can contain more than one type of content. In this case, a single
521 *packageType* value should be selected that represents the primary content type as determined by the
522 SDD author. The SDD defines a set of enumeration values in *PackageTypeType* which are
523 extendable by the SDD author.

524 The enumerated types defined by the SDD are as follows:

525 • **baseInstall**: The value *baseInstall* indicates that the package provides a complete installation
526 of the solution. This package type is associated with deployment descriptors that contain
527 installable units with installation artifacts that install the primary solution resources.

528 When *packageType* is not specified, this is the default value.

529 • **baseUninstall**: The value *baseUninstall* indicates that the package provides a complete
530 uninstallation of the solution. This package type is associated with deployment descriptors
531 that contain installable units with uninstall artifacts that remove the primary solution
532 resources.

533 • **configuration**: The value *configuration* indicates that the package configures the solution.
534 This package type is associated with deployment descriptors that contain configuration units
535 with configuration artifacts that configure the solution.

536 • **maintenance**: The value *maintenance* indicates that the package fixes one or more problems
537 in the solution. This package type is associated with deployment descriptors that contain
538 installable units with update artifacts.

539 • **modification**: The value *modification* indicates that the package modifies the function of the
540 solution in some way such as by adding new function. This package type is associated with
541 deployment descriptors that contain installable units with update artifacts.

542 • **replacement**: The value *replacement* indicates that the package installs a solution that
543 replaces a previous version of the solution. Replacement MAY be associated with migration
544 of data into the new solution and/or with deletion of the replaced solution. When associated
545 with migration of data, installation or configuration artifacts within the solution package would
546 perform the migration. When associated with deletion of the replaced solution, uninstall
547 artifacts within the solution package would perform the deletion. This package type is
548 associated with deployment descriptors that contain installable units with installation artifacts

549 that deploy a set of resources that replace the set of resources associated with a previous
550 version of the solution.

- • **localization**: The value *localization* indicates that the package contains materials that
552 localize deployed software for one or more languages.

- ▪ **contentType**: The value of *contentType* is determined by the SDD manufacturer to communicate a
554 characteristic of the package that MAY be used in the manufacturer's package management system
555 or other manufacturer-specific tools that use the SDD. The SDD author chooses the values; they are
556 not defined in this specification.

- ▪ **label**: The label MAY be used as an index in a package management system. The SDD author
558 chooses the values; they are not defined in this specification.

3.4 IdentityType

560
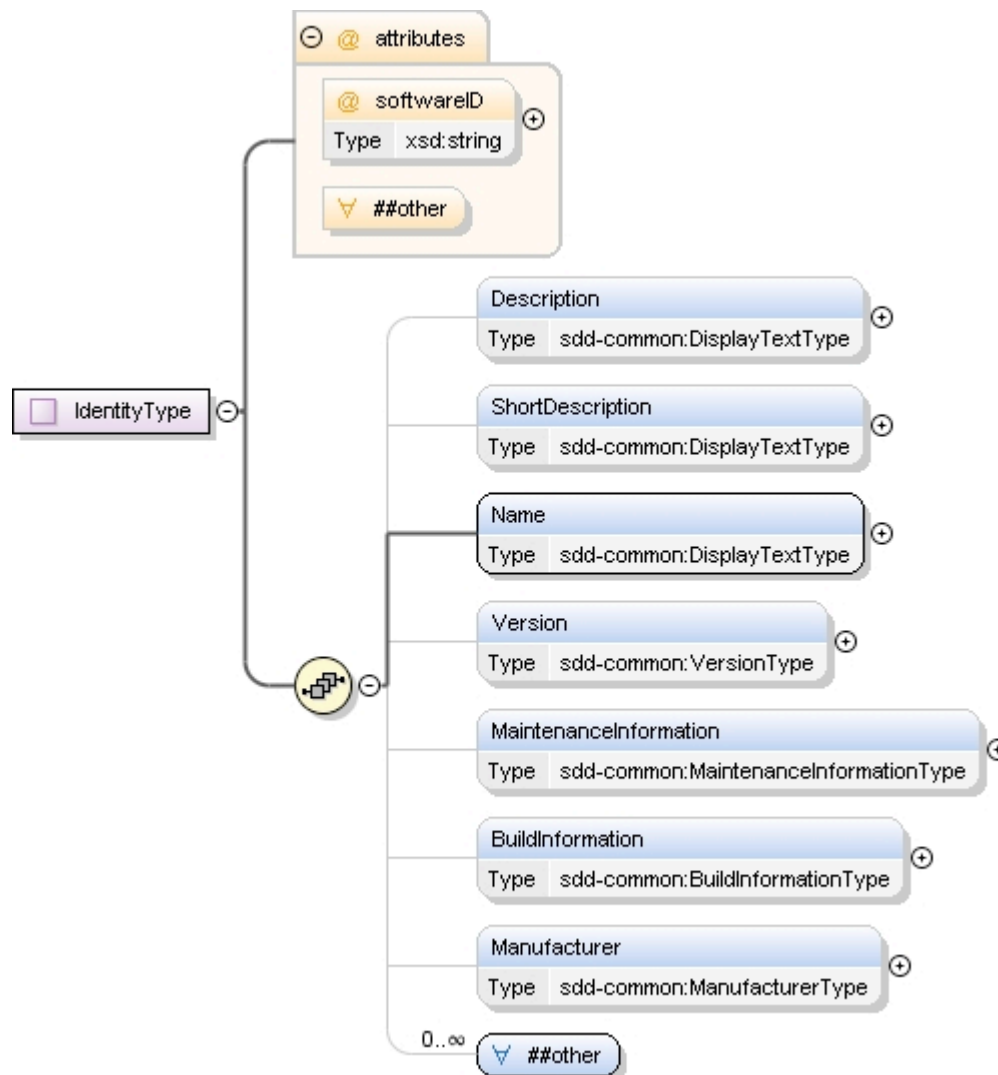
Figure 5: IdentityType structure.

562 This complex type provides identity information for the package as a whole, as well as for content
563 elements, which are portions of the package. Content elements are the *InstallableUnit, LocalizationUnit,*
564 *ConfigurationUnit, CompositeUnit* and *CompositeInstallable* elements defined in the deployment
565 descriptor.

3.4.1 IdentityType Property Summary

Name	Data Type	*	Description
Description	DisplayTextType	0..1	A verbose description of the package or content element.
ShortDescription	DisplayTextType	0..1	A limited description of the package or content element.
Name	DisplayTextType	0..1	A human-readable, translatable, name for the package or content element.
Version	VersionType	0..1	The package or content element version.
MaintenanceInformation	MaintenanceInformationType	0..1	Information about package or content element content used when the package contains maintenance.
BuildInformation	BuildInformationType	0..1	A manufacturer identifier for the build of this package or content element. This property can be extended with additional manufacturer-specific information about the build.
Manufacturer	ManufacturerType	0..1	Information about the manufacturer of the package or content element.
	xsd:any	0..*	
softwareID	xsd:string	0..1	A manufacturer's identification number for the software created or updated by the package or content element.
	xsd:anyAttribute	0..*	

3.4.2 IdentityType Property Usage Notes

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the package.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DisplayTextType* section for structure and additional usage details [4.14.3].

- **Name**: When the manufacturer of the SDD has a package management system, *Name* in *PackageIdentity* should correspond to the name of the package as known in the package management system. *Name* in a content element's *Identity* should correspond to the name of the unit of packaging, if it is known in the package management system.

  When the *PackageIdentity* element is defined, *Name* MUST be defined.

  Software packages that create software often have the same name as the deployed software. Software packages that update software often have a name that reflects the fact that the package is a maintenance package, differentiating it from the base deployed software. The author of the software package that is described by *PackageIdentity* determines whether the *Name* is the same as or different from the *Name* of the deployed software.

  See the *DisplayTextType* section for structure and additional usage details [4.14.3].

- **Version**: This is a packaging version. In *PackageIdentity*, it is the version of the package as a whole. In content element identities, this is the version of the unit of packaging represented by the content element. In either case, the SDD author MAY choose to make this version correspond to the version of a resulting or changed resource, but it should not be confused with resource versions.

  In the case of a base install, version MAY be the same as the top level resulting resource. In the case of a configuration package, version SHOULD NOT be the same as the top level resulting resource.

  See the *VersionType* section for structure and additional usage details [3.10].

| 590 | ▪ | **MaintenanceInformation**: This is used when the package or content element describes the |
| 591 | | deployment of maintenance. |

| 592 | See the *MaintenanceInformationType* section for structure and additional usage details [3.5]. |

593	▪	**BuildInformation**: In *PackageIdentity*, this describes the build of the package as a whole. In content
594		element *Identity*, this describes the build of the artifact(s) and the content element describing the
595		artifact.

| 596 | See the *BuildInformationType* section for structure and additional usage details [3.7]. |

| 597 | ▪ | **Manufacturer**: See the *ManufacturerType* section for structure and additional usage details [3.8]. |

598	▪	**softwareID**: The software identified by *softwareID* is the software whose deployment is described by
599		the SDD. When the manufacturer maintains software identifiers within a sales and distribution
600		system, the *softwareID* SHOULD correspond to an identifier for the software within that system. If a
601		format for software identifiers is not pre-existing within the manufacturer's systems, a UUID SHOULD
602		be used for *softwareID*. When a UUID is used, it MUST be unique within the domain in which the
603		described software is used.

604 3.5 MaintenanceInformationType

605
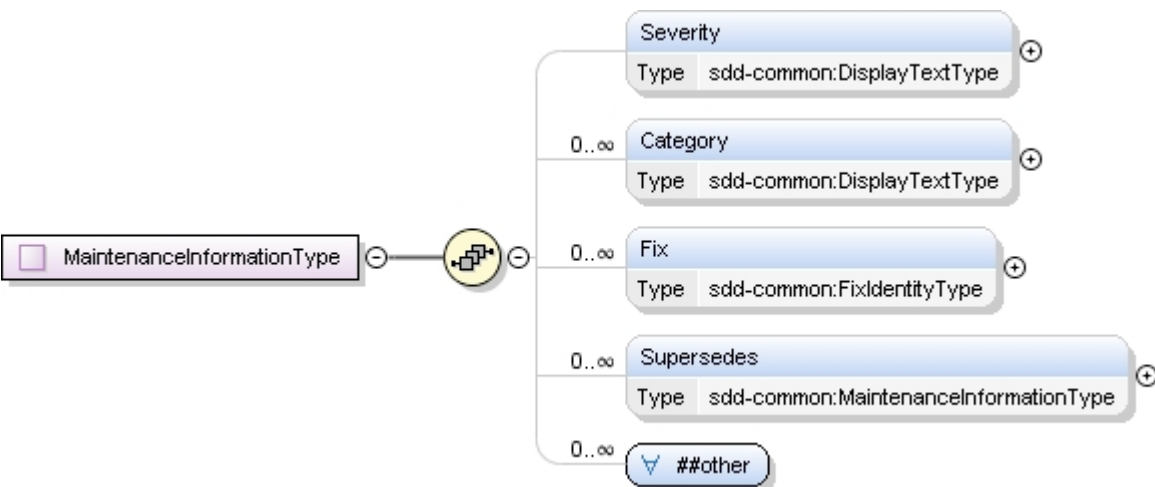
606 Figure 6: MaintenanceInformationType structure.

607 If the package provides maintenance for deployed software, *MaintenanceInformation* declares information
608 about the fix or fixes provided. If the package content is a single fix, *MaintenanceInformation* describes
609 the information about that one fix. If the content is a collection of fixes—for example, a fix pack—
610 *MaintenanceInformation* describes each of the fixes provided by the fix pack.

611 3.5.1 MaintenanceInformationType Property Summary

Name	Type	*	Description
Severity	DisplayTextType	0..1	Severity of the maintenance content.
Category	DisplayTextType	0..*	Category of the maintenance content.
Supersedes	MaintenanceInformationType	0..*	A previously released fix that is superseded by application of this maintenance.
Fix	FixIdentityType	0..*	An included fix.
	xsd:any	0..*	

3.5.2 MaintenanceInformationType Property Usage Notes

- **Severity**: This value SHOULD correspond to a severity value used within the SDD provider's support system. It serves as a hint to the deployer about the urgency of applying the described maintenance.

  See the *DisplayTextType* section for structure and additional usage details [4.14.3].

- **Category**: These values SHOULD correspond to maintenance categories within the SDD provider's support system.

  See the *DisplayTextType* section for structure and additional usage details [4.14.3].

- **Supersedes**: Superseded fixes are ones that fix a problem also fixed by the superseding maintenance package or content element and therefore need not be applied.

  This element does not indicate whether or not the superseded fix needs to be removed. To indicate that the previous fix must be removed before the superseding maintenance can be applied successfully; the SDD author can create a requirement stating that the fix must not be present.

  Superseded fixes MAY include all the information defined in *MaintenanceInformationType*. At a minimum, a superseded fix MUST include at least one *Fix* element with the name of the superseded fix defined.

- **Fix**: *Fix* elements provide information about individual fixes provided by the maintenance content.

  See the *FixIdentityType* section for structure and additional usage details [3.6].

3.6 FixIdentityType
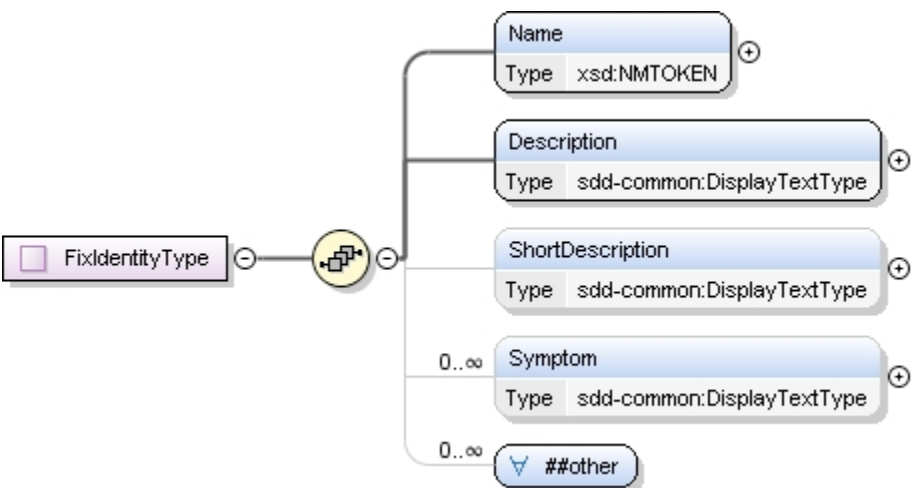
Figure 7: FixIdentityType structure.

Elements of *FixIdentityType* describe fixes that will be applied when the package is deployed or the content element is applied.

3.6.1 FixIdentityType Property Summary

Name	Type	*	Description
Name	xsd:NMTOKEN	1	A name for the fix which is, at a minimum, unique within the scope of the resource fixed.
Description	DisplayTextType	1	A complete description of the fix.
ShortDescription	DisplayTextType	0..1	An abbreviated description of the fix.
Symptom	DisplayTextType	0..*	A symptom of the problem fixed.
	xsd:any	0..*	

3.6.2 FixIdentityType Property Usage Notes

- **Name**: The *Name* element MUST provide a value that uniquely identifies a fix within a scope defined by the manufacturer. This is a name provided by the manufacturer that corresponds to the fix name as understood in the deployment environment.

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the fix.

    The *Description* element MUST be defined if the *ShortDescription* element is defined.

    See the *DisplayTextType* section for structure and additional usage details [4.14.3].

- **Symptom**: Symptom strings can be used to correlate a fix with one or more experienced problems.

    See the *DisplayTextType* section for structure and additional usage details [4.14.3].

3.7 BuildInformationType
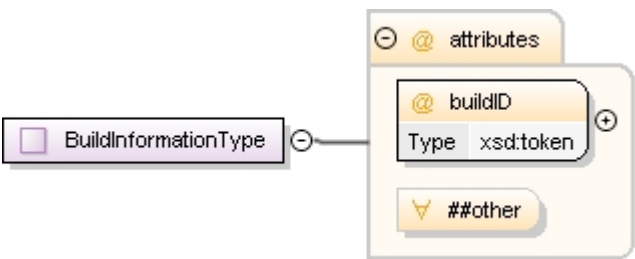
Figure 8: BuildInformationType structure.

BuildInformationType provides the type definition for the *BuildInformation* element in package and content element identity. *BuildInformation* provides information about the creation of the package and its parts.

3.7.1 BuildInformationType Property Summary

Name	Type	*	Description
buildID	xsd:token	1	Identifies the build of the package or package element.
	xsd:anyAttribute	0..*	

3.7.2 BuildInformationType Property Usage Notes

- **buildID**: The *buildID* attribute is an identifier provided by the manufacturer and meaningful to developers that can be used to identify a build of the defining element. This information MUST correspond with information known in the manufacturer's build environment. It is traditionally used during problem determination to allow maintainers of the software to determine the specifics of package creation. Inclusion of *buildID* in the SDD allows the end user to provide this information to package maintainers, enabling them to correlate the deployed software with a particular known build of the software.

659   3.8 ManufacturerType

660
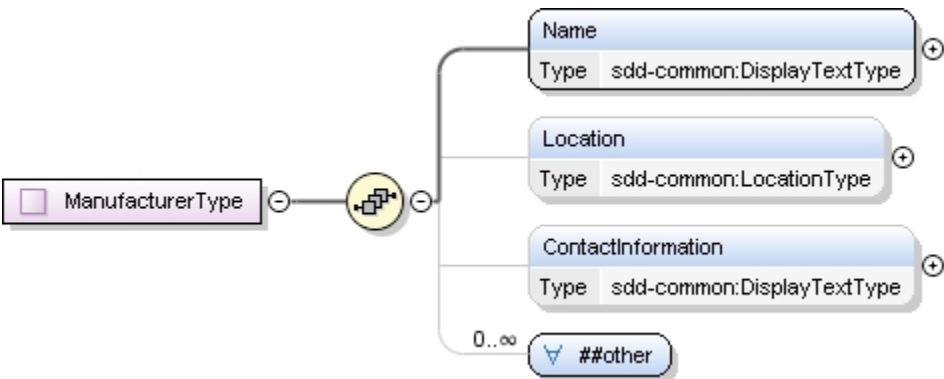
Figure 9: ManufacturerType structure.

662   The SDD author can include information about the package manufacturer that includes name, location
663   and contact information such as the address of the manufacturer's Web site or telephone number.

664   3.8.1 ManufacturerType Property Summary

Name	Type	*	Description
Name	DisplayTextType	1	A translatable name for the manufacturer.
Location	LocationType	0..1	The address and country of the manufacturer.
ContactInformation	DisplayTextType	0..1	Contact information for the manufacturer.
	xsd:any	0..*	

665   3.8.2 ManufacturerType Property Usage Notes

666   ▪ **Name**: The value provided in the *Name* element MUST be an identifiable name of the manufacturer
667   of the SDD.

668   See the *DisplayTextType* section for structure and additional usage details [4.14.3].

669   ▪ **Location**: See the *LocationType* section for structure and additional usage details [3.9].

670   ▪ **ContactInformation**: This element MAY provide additional contact information for the named
671   manufacturer, such as a support Web site address or a technical support telephone number.

672   See the *DisplayTextType* section for structure and additional usage details [4.14.3].

673   3.9 LocationType

674
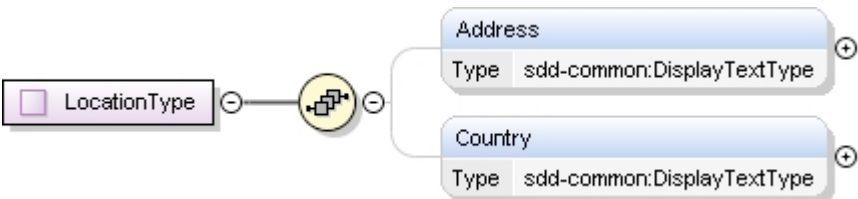
Figure 10: LocationType structure.

676   *LocationType* supports inclusion of the manufacturer's address and country in package and content
677   element identity.

3.9.1 LocationType Property Summary

Name	Type	*	Description
Address	DisplayTextType	0..1	The manufacturer's address.
Country	DisplayTextType	0..1	The manufacturer's country.

3.9.2 LocationType Property Usage Notes

- **Address**: This is the mailing address or the physical address.

    See the *DisplayTextType* section for structure and additional usage details [4.14.3].
- **Country**: Recording the manufacturer's country in the SDD provides information that may be of interest in relation to import and export of software.

    See the *DisplayTextType* section for structure and additional usage details [4.14.3].

3.10 VersionType

VersionType provides the type definition for version elements in the package descriptor and deployment descriptor. It is a simple type that is based on `xsd:string` with no further restrictions. This means that versions in the SDD are represented simply as strings. Because resource versions exist in the deployment environment, their formats and semantics vary widely. For this reason, the format and semantics of versions are not defined by this specification.

3.11 ContentsType
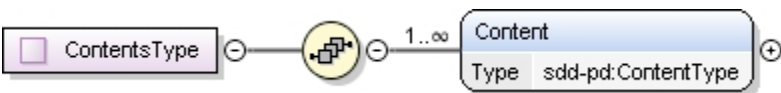
Figure 11: Contents structure.

ContentsType is used in *PackageDescriptor* to provide a list of one or more *Content* elements.

3.11.1 ContentsType Property Summary

Name	Type	*	Description
Content	ContentType	1..*	Describes the physical contents of the software package.

3.11.2 ContentsType Property Usage Notes

- **Content**: A *PackageDescriptor* MUST contain a *Contents* element that is a list of one or more *Content* elements.

    See the *ContentType* section for structure and additional usage details [3.12].

700 3.12 ContentType
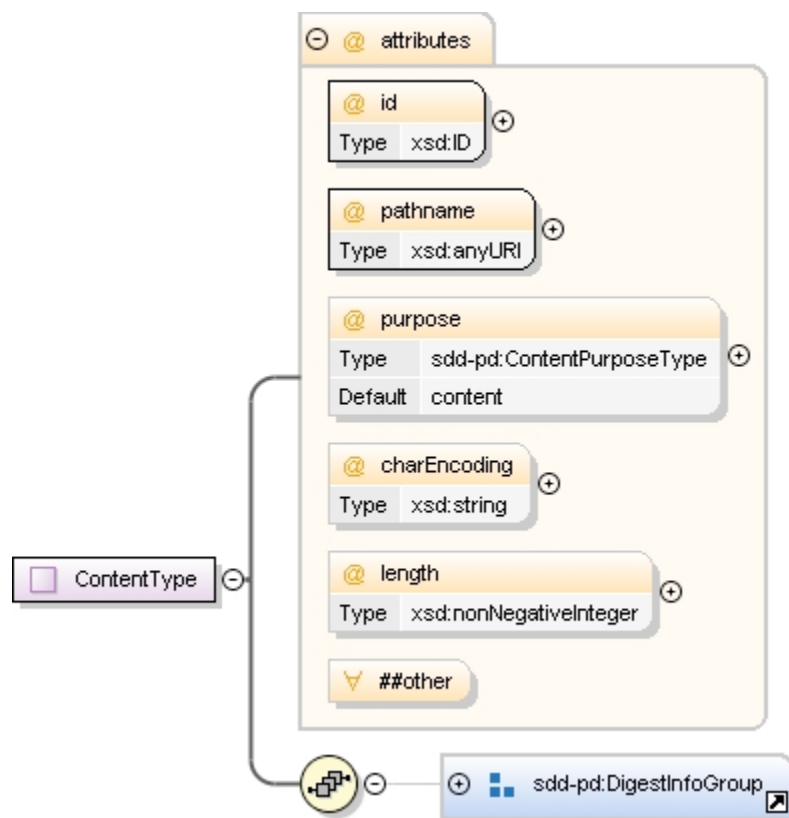
702 **Figure 12: ContentType structure.**

703 A software package includes one or more content files. *ContentType* defines the properties of a content
704 file included in the package descriptor. Content defined in the package descriptor as part of the software
705 package does not need to be physically co-located. Each element MUST be in a location that can be
706 identified by a URI. The *pathname* attribute of each content file defines a URI for accessing the file.
707 Characteristics of the content files—such as their length, purpose and character encoding—MAY be
708 declared in the package descriptor. It is RECOMMENDED to list only content files that need to be
709 accessed by the runtime when processing the SDD.

710 3.12.1 ContentType Property Summary

Name	Data Type	*	Description
ds:DigestMethod	ds:DigestMethodType	0..1	Specifies the digest method applied to the file.
ds:DigestValue	ds:DigestValueType	0..1	Specifies the Base64-encoded value of the digest of the file.
id	xsd:ID	1	An identifier used in deployment descriptors to refer to the file definition in the associated package descriptor.
pathname	xsd:anyURI	1	The absolute or relative path of the content file including the file name.
purpose	ContentPurposeType	0..1	Associates a purpose classification with a file. **default value="content"
charEncoding	xsd:string	0..1	Specifies the character encoding of the contents of the file.
length	xsd:nonNegativeInteger	0..1	Specifies the size of the file in bytes.

	xsd:anyAttribute	0..*	

3.12.2 ContentType Property Usage Notes

711

712 ▪ **ds:DigestMethod, ds:DigestValue**: These values MAY be used to assist with file verification.

713 See the *DigestInfoGroup* section for structure and additional usage details [3.13].

714 ▪ **id**: This is the identifier for the content that is used as a reference in artifact elements in the
715 deployment descriptor.

716 The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
717 log and trace messages.

718 ▪ **pathname**: *pathname* is used to access content in the package. The path of the file MUST be a URI
719 that specifies an absolute path or a path relative to the location of the package descriptor. It MUST
720 include the file name. For a *pathname* that has a *purpose* of *descriptorLanguageBundle*, the SDD
721 author SHOULD include the URI to the root resource bundle only.

722 ▪ **purpose**: The *purpose* attribute enables the *PackageDescriptor* author to associate a classification
723 with a file. The classification identifies the file as having a specific purpose. *ContentPurposeType*
724 defines a union of *SDDContentPurposeType* with `xsd:NCName`. The *purpose* value MAY be chosen
725 from one of the following values enumerated in *SDDContentPurposeType* or be a valid NCName
726 value provided by the SDD author. If *purpose* is not specified, the default value is *content*.

727 Enumerated values for *purpose* are:

728 • **readMe**: A file with information about the package. An implementation may choose to display
729 this to a user as part of the deployment process.

730 • **endUserLicenseAgreement**: A file containing an end user license agreement. An
731 implementation may choose to display this to a user as part of the deployment process.

732 • **responseFile**: A file that contains input values for an operation.

733 • **deploymentDescriptor**: An XML file containing the *DeploymentDescriptor* definition
734 associated with the *PackageDescriptor*. A valid *PackageDescriptor* MUST have exactly one
735 *Content* element with a *purpose* value of *deploymentDescriptor*.

736 • **packageDescriptor**: Supports aggregation of packages. This is used to reference a
737 *packageDescriptor* of an aggregated package.

738 • **descriptorLanguageBundle**: A file containing translations of text defined directly in the
739 package descriptor or its associated deployment descriptor. See **[SDDP]** for an example that
740 includes *descriptorLanguageBundle* content.

741 • **content**: A file used during deployment of solution content. This is the default value for
742 purpose.

743 ▪ **charEncoding**: This attribute need only be used for files that a run-time is required to render.
744 Common *charEncoding* values include "ASCII", "UTF-8", "UTF-16" and "Shift_JIS". For an extensive
745 list of character encodings, see **[IANA-CHARSET]**.

746 ▪ **length**: The file length MAY be used for simple file verification.

3.13 DigestInfoGroup

747
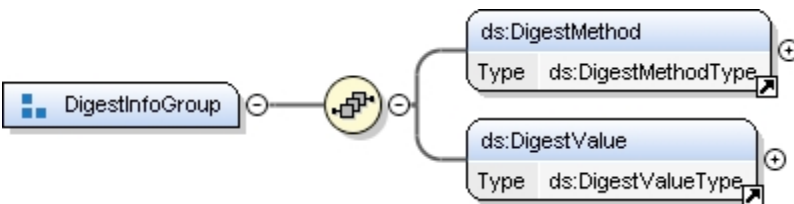
748
749 **Figure 13: DigestInfoGroup structure.**

750 When digest information is used to sign a content file, both the digest method and the digest value MUST
751 be provided.

3.13.1 DigestInfoGroup Property Usage Notes

753 ▪ **ds:DigestMethod, ds:DigestValue**: *ds:digestMethod* and *ds:digestValue* MAY be used to digitally
754 sign individual files. If files are signed, the digest value MUST be calculated over the whole of each
755 file.

756 See **[XMLDSIG-CORE]** for details on the usage of *ds:DigestMethod* and *ds:DigestValue*.

4 Deployment Descriptor

757

758 A solution package contains a deployment descriptor in addition to a package descriptor. The deployment
759 descriptor describes the topology, selectability, inputs, requirements and conditions of the deployment.
760 The deployment descriptor is associated with a package descriptor and refers to content files in that
761 package descriptor.

4.1 DeploymentDescriptor

762

763
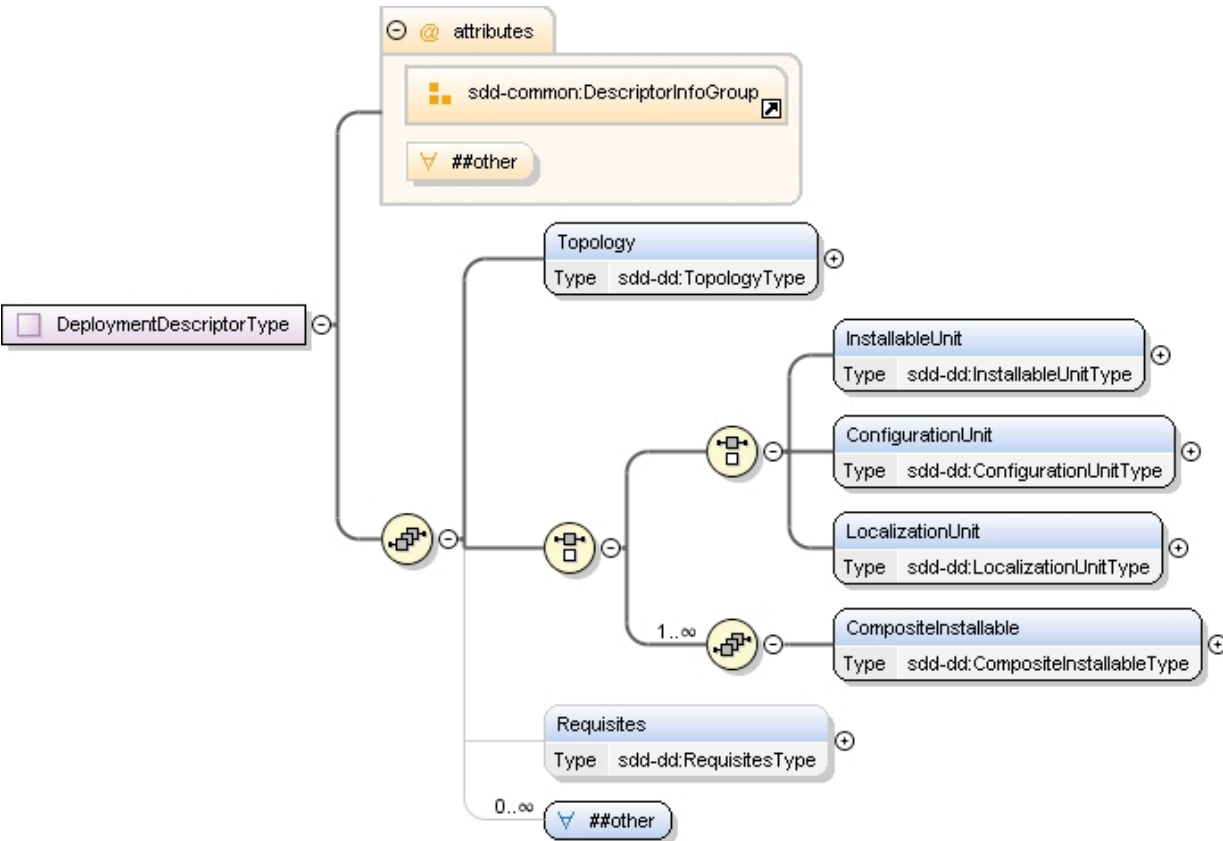
Figure 14: DeploymentDescriptor structure.

764

765 *DeploymentDescriptor* is the top level element of a deployment descriptor. The *DeploymentDescriptor*
766 defines the information required to support deployment of the package contents. This includes the
767 *Topology*, which declares all of the resources that may participate in deployment. It also includes one
768 atomic content element or one or more *CompositeInstallable* content elements. Atomic content elements
769 are *InstallableUnit*, *ConfigurationUnit*, or *LocalizationUnit*. Atomic content elements define artifacts that
770 can be processed to deploy software resources. They are atomic because they cannot aggregate other
771 content elements. A *CompositeInstallable* element is the root of a content element hierarchy that defines
772 content that performs the one deployment operation supported by the *CompositeInstallable*. A
773 *CompositeInstallable* can define base, selectable and localization content as well as the aggregation of
774 other content elements.

4.1.1 DeploymentDescriptor Property Summary

775

Name	Type	*	Description
Topology	TopologyType	1	Defines resources that are required, created or modified by

			deployment.
InstallableUnit	InstallableUnitType	0..1	Defines content that installs, updates and/or uninstalls resources. When an InstallableUnit is defined, no ConfigurationUnit, LocalizationUnit or CompositeInstallable elements can be defined.
ConfigurationUnit	ConfigurationUnitType	0..1	Defines content that configures resources. When a ConfigurationUnit is defined, no InstallableUnit, LocalizationUnit or CompositeInstallable elements can be defined.
LocalizationUnit	LocalizationUnitType	0..1	Defines content that installs, updates and/or uninstalls translated materials. When a LocalizationUnit is defined, no InstallableUnit, ConfigurationUnit or CompositeInstallable elements can be defined.
CompositeInstallable	CompositeInstallableType	0..*	Defines a hierarchy of base, selectable and/or localization content used to perform one deployment lifecycle operation. When one or more CompositeInstallable elements are defined, no InstallableUnit, ConfigurationUnit or LocalizationUnit elements can be defined.
Requisites	RequisitesType	0..1	A list of references to SDD packages that can optionally be deployed to satisfy deployment requirements of the defining SDD.
	xsd:any	0..*	Describes completion actions such as restart and the conditions under which the action is applied.
schemaVersion	xsd:string	1	The descriptor complies with this version of the Solution Deployment Descriptor Specification. **fixed value="2.0"
descriptorID	UUIDType	1	Identifier of the deployment descriptor for a particular set of deployable content.
lastModified	xsd:dateTime	1	The time the descriptor was last modified.
	xsd:anyAttribute	0..*	

4.1.2 DeploymentDescriptor Property Usage Notes

- **Topology**: *Topology* provides a logical view of all resources that may participate in any particular deployment. A resource can participate by being required, created or modified by the deployment. A required resource MAY also play the role of target resource, meaning that it can process artifacts to perform some portion of the deployment. The resources that actually participate in a particular deployment are determined by the user inputs, selections and resource bindings provided during that deployment.

  See the *TopologyType* section for structure and additional usage details [4.2.1].

- **InstallableUnit, ConfigurationUnit, LocalizationUnit, CompositeInstallable**: A simple software deployment that uses a single artifact for each supported deployment operation MAY be described using an SDD that defines a single atomic content element–*InstallableUnit*, *ConfigurationUnit* or *LocalizationUnit*.

  A software deployment that requires multiple artifacts, aggregates other deployment packages or has selectable content MAY be described using an SDD that defines one or more *CompositeInstallable* elements. Each *CompositeInstallable* MUST describe one deployment lifecycle operation for the package.

792 See the respective sections (*InstallableUnitType* [4.3.1], *ConfigurationUnitType* [4.3.2],
793 *LocalizationUnitType* [4.13.2] and *CompositeInstallableType* [4.9.1]) for structure and additional
794 usage details.

795 ▪ **Requisites**: When the package author chooses to provide deployment packages for required
796 software, those packages are described by *Requisite* elements in *Requisites*.

797 Including requisite packages in the SDD package MAY provide a convenient way for the deployer to
798 satisfy one or more SDD requirements.

799 See the *RequisitesType* section for structure and additional usage details [4.10.5].

800 ▪ **schemaVersion, descriptorID, lastModified**: These attributes can be useful to tooling that
801 manages, creates or modifies deployment descriptors.

802 See the *DescriptorInfoGroup* section for structure and additional usage details [3.2].

4.2 Topology

804 The SDD's topology describes all the resources that may be required, created or modified when any of
805 the deployment operations supported by the SDD are performed.

806 Primary identifying characteristics of the resources can be defined in topology. Constraints beyond these
807 primary characteristics are not defined in topology; they are defined in content elements that reference
808 the resource definitions in topology.

809 The topology includes identification of *hosts–hostedBy* relationships between resources. When both
810 resources in that relationship participate in a particular deployment, the relationship is considered a
811 requirement for that deployment.

812 It is possible that only a subset of the resources described in topology will play a role in a particular
813 deployment. This is determined by the selection of content elements for the particular deployment. The
814 resources that are required, created or modified by the content elements in scope for the deployment are
815 the ones that will participate in the deployment and so are associated with resources in the deployment
816 environment.

817 At deployment time, definitions of the resources that participate in that particular deployment are
818 associated with actual resource instances in the deployment environment. The mechanisms for
819 associating resource definitions with resource instances are not described by the SDD. The SDD
820 metadata describes the characteristics of the participating resources. Whether associations of resource
821 instances with matching characteristics are made by user choice or entirely by software does not affect
822 the success of the deployment. Resource characteristics used when making this association include
823 those defined in topology plus all those defined in constraints on the resource in the content elements that
824 are in scope for the particular deployment.

825 Some topologies are variable. That is, a particular set of logical resources of the same type in the
826 topology might be associated with different physical resource instances or the same physical resource
827 during deployment. In this case, a separate logical resource definition is created in topology for each
828 possible physical resource instance. Uniqueness constraints can then be used to describe the conditions
829 under which the separate resources can be associated with a single resource.

830 All resource definitions in the SDD are in topology. All other descriptions of resources in the SDD are
831 references to the resource definitions in the topology.

4.2.1 TopologyType
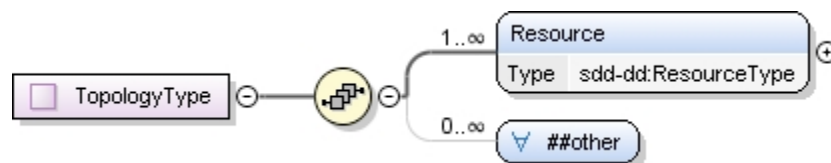
834 **Figure 15: TopologyType structure.**

835 The *Topology* element defines one or more hierarchies of resource specifications that describe the
836 resources that MAY play a role in the deployment of the contents of the solution package. These resource
837 specifications do not identify specific resource instances in a specific deployment environment. Instead,
838 they are logical specifications of resources that can be associated with specific resource instances in the
839 deployment environment for a particular deployment based on the described resource identity
840 characteristics. These resources have a role in a particular solution deployment only when they are
841 required, created or modified by a content element, or referred to by a variable, in that particular solution
842 deployment.

843   4.2.1.1 TopologyType Property Summary

Name	Type	*	Description
Resource	ResourceType	1..*	The root of a tree of resources that play a role in the solution.
	xsd:any	0..*	

844   4.2.1.2 TopologyType Property Usage Notes

845   ▪ **Resource**: The SDD author's decision to model a resource in the deployment environment as a
846   resource in the SDD depends on the need to know about that resource when planning for
847   deployment, aggregating, deploying and managing the resource lifecycle using the SDD. All
848   resources required by the solution SHOULD be included. For all *Requirements* declared in the SDD,
849   resources MUST be specified. Resources referred to by *ResultingResource* or *ResultingChange*
850   elements MUST also be included. The more complete the SDD is, the more useful it will be in guiding
851   successful deployment.

852   See the *ResourceType* section for structure and additional usage details [4.2.2].
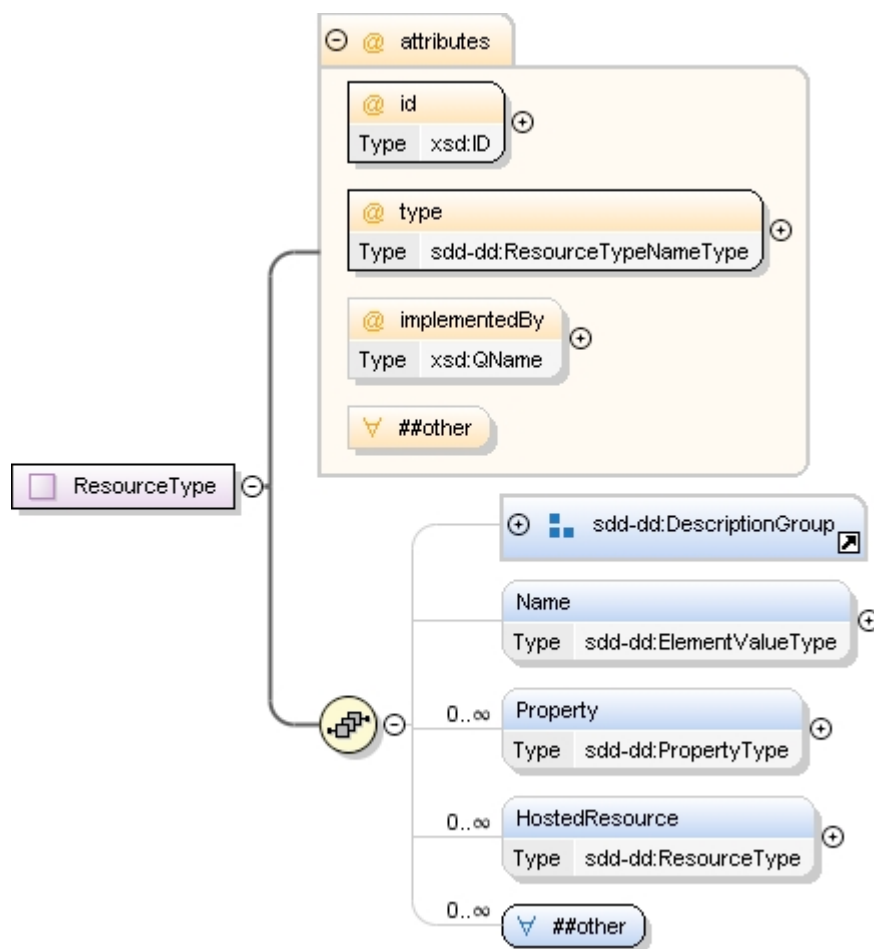
4.2.2 ResourceType

Figure 16: ResourceType structure.

Elements of *ResourceType*—both the top level *Resource* elements and the *HostedResource* elements within the resource hierarchy—make up the topology of an SDD. Each *Resource* element declares, at a minimum, the type of the resource. Values for resource type are not defined by this specification. A core assumption of this specification is that an understanding of specific resource types and resource characteristics are shared by the deployment descriptor author and the deployment software. Therefore, if the deployment descriptor author declares a new resource type, then deployment software operating on the SDD needs to understand how to handle that resource type.

In addition to defining type, the resource elements MAY specify other identity properties that can be used to identify instances of the resource in the deployment environment. The resource identity element, *Property,* is optional and MAY be specified in content elements rather than in topology. Identity properties used in the resource specification in topology MUST be those that do not change during deployment, even when the resource is updated.

> For example, during an update, software may change its version string, thus the version string is not an appropriate identity property.

ResourceType provides the type definition for the *Resource* and *HostedResource* elements defined in *Topology*. All resources MAY nest resource definitions for resources that they host. To host a resource means to provide the execution environment for that resource.

> For example, an operating system provides the execution environment for software, and a database engine provides the execution environment for a database table. The operating system hosts the software and the database engine hosts the database table.

Each resource in these hierarchies may play a role in solution deployment.

877 Content elements determine a resource's participation and role(s) in a particular solution deployment.
878 Content elements can refer to resources in *Topology* in several ways. A resource can be identified via
879 `xsd:IDREF`:

- 880 ▪ as the target of the content element's artifacts. A target resource is a resource that is capable of
881 processing a particular artifact. A target resource is often, but not always, the host of the
882 resources created by the artifacts it processes.

    883 For example, an operating system may be the target resource of an artifact that is a zip file
    884 containing a J2EE application. However, when the J2EE application is deployed, a J2EE
    885 server is the host resource of the application. Thus, the OS hosts the artifact and the J2EE
    886 server hosts the J2EE application.

    887 See the *targetResourceRef* attribute in the *InstallableUnitType* [4.3.1], *ConfigurationUnitType*
    888 [4.3.2] and *LocalizationUnitType* [4.13.2] sections.

- 889 ▪ as the required base for an update applied by the artifact referenced by the content element.

    890 See the *RequiredBaseType* section [4.7.8].

- 891 ▪ as the resource that will be created by deploying the artifact referenced by the content element.

    892 See the *ResultingResourceType* section [4.8.1].

- 893 ▪ as the resource that will be changed by deploying the artifact referenced by the content element.

    894 See the *ResultingChangeType* section [4.8.2].

- 895 ▪ as the localization base for translated materials. The localization base is the resource that is
896 localized by deploying the translated materials.

    897 See the *LocalizationBase* element in the *LocalizationUnitType* section [4.13.2].

- 898 ▪ as a required resource named in the content element's *Requirements*.

    899 See the *RequirementsType* section [4.7.1].

- 900 ▪ to establish a variable value from a resource property.

    901 See the *ResourcePropertyType* section [4.6.18].

902 One resource MAY be referred to by any number of content elements and can be identified to play any or
903 all of the roles just listed. When a content element participates in a particular solution deployment, the
904 resources it references participate in that solution deployment and are associated with resource instances
905 in the deployment environment.

4.2.2.1 ResourceType Property Summary

906

Name	Type	*	Description
Description	DisplayTextType	0..1	A description of the resource and its role in the solution described by the SDD.
ShortDescription	DisplayTextType	0..1	A short description of the resource and its role.
~~Name~~	~~VariableExpressionType~~	~~0..1~~	~~The name of the resource as known in the deployment environment.~~ [DEPRECATED in SDD v2.0]
Property	PropertyType	0..*	An identity property of the resource.
HostedResource	ResourceType	0..*	A resource that participates in the solution and that is hosted by the defining resource.
	xsd:any	0..*	
id	xsd:ID	1	An identifier of the resource scoped to the descriptor.
implementedBy	xsd:QName	0..1	A reference to another hosted resource in topology.

type	ResourceTypeNameType	1	A well-known resource type.
	xsd:anyAttribute	0..*	

4.2.2.2 ResourceType Property Usage Notes

- **Description, ShortDescription**: If used, these elements MUST provide a human-readable description of the resource.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

- ~~**Name**: The resource name is an identifying characteristic of the resource that correlates with a name for the resource in the deployment environment.~~

  ~~The type of the *Name* element, *VariableExpressionType*, allows the resource name to be expressed as a simple string or in terms of a user input parameter or other variable.~~

  ~~An example of a good use of a variable expression in *Resource.Name* is to make sure that the installation directory is hosted on a file system that has sufficient space available for deployment. In this example, the file system resource element would define a *HostedResource* element for the directory. The *Name* of the directory would be expressed as a variable expression that refers to a user input parameter for installation location. Content elements that use the installation directory would express a requirement on the directory and on the file system with the additional constraint that the file system have a certain amount of available space (to satisfy the consumption constraints). The fact that both resources are required and that they are defined with a *hosts–hostedBy* relationship in *Topology,* means that the directory that is used must be the installation directory and it must be hosted by a file system that meets the consumption constraint for available space.~~

  ~~Only the *Variable* elements defined in a top level content element can be used to define a resource *Name,* because these are the only variables visible within *Topology*.~~

  ~~If the name of a resource is changed during deployment, for example, during an update, then the resource name SHOULD NOT be included in the resource specification. Instead, the pre-update resource name SHOULD be specified in the *RequiredBase* element of the installable unit that provides the update, and the post-update name SHOULD be specified in the *ResultingResource* element of the same installable unit.~~

  ~~See the *VariableExpressionType* section for structure and additional usage details [4.6.1].~~

  [Starting with SDD v2.0, *Name* has been deprecated. See the *Property* element below for the appropriate method for specifying a resource identifier.]

- **Property**: *Property* elements MUST be used to identify the resource instance(s). Each property included represents an identifying characteristic necessary for accurate run-time resolution of the resource instance(s).

  If a resource can be identified by a property that represents the name for that resource, the SDD author SHOULD include a *Property* element and MUST set the value of *PropertyName* to "Name".

  See the *PropertyType* section for structure and additional usage details [4.2.3].

- **HostedResource**: A *Resource* MAY define *HostedResource* elements. Each *HostedResource* element is an instance of *ResourceType*. When both the host and the hosted resource participate in a particular solution deployment, the associated resource instances selected for use during that deployment must have a *hosts* relationship.

  For example, a Web application declared to be hosted on a Web server must be hosted on the instance of the Web server that is selected for use during the deployment.

  If only the host resource is identified by the *DeploymentDescriptor's* content elements as participating in the solution, then there is no assumption that the hosted resource exists.

951 ▪ **id**: The *id* attribute uniquely identifies the resource element within the *DeploymentDescriptor*. This *id*
952   value is used by other elements in the *DeploymentDescriptor* to refer to this resource. This value is
953   created by the descriptor author.

954   The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
955   log and trace messages.

956 ▪ **implementedBy**: The *implementedBy* attribute is useful for type casting a resource to another
957   defined hosted resource.

958   For example, an SDD producer needs to deploy a database. The only property with which the
959   producer is concerned is the destination path property. It is cumbersome to require the producer
960   and the runtime to both handle the hosted resource as database specific when a generic resource
961   type will suffice for the operation.

962   In this case it is useful for the SDD producer to define a hosted resource as a generic resource
963   type and use it to typecast the specific database resource type. This allows subsequent operations
964   to recognize that a database has been deployed to the hosting environment even though a generic
965   resource type was used to perform that deployment.

966   To do this, the SDD producer MUST define a generic hosted resource. This is done by defining a
967   hosted resource with the type attribute being a `xsd:QName` reference to a generic resource type
968   from the profile. The SDD producer then defines the specific hosted resource. To typecast the
969   specific hosted resource, the previously defined generic hosted resource should be specified in the
970   *implementedBy* attribute.

971   See **[SDDEX]** for an example that demonstrates this use of the *implementedBy* attribute.

972 ▪ **type**: The *type* attribute defines the class of resource. The value of *type* correlates with the resource
973   type known for the resource in the deployment environment. *ResourceTypeNameType* restricts *type*
974   to valid `xsd:QNames`. The values for *type* are not defined by this specification. Creators of
975   *DeploymentDescriptors* rely on knowledge of resource types that are understood by supporting
976   infrastructure in the target environment. To honor the descriptor author's intent, the deploying
977   infrastructure must be able to discover the existence of resources of the types defined in the SDD; the
978   values of the resource's properties; and the existence and type of resource relationships. The
979   deploying infrastructure also needs to understand how to use the artifact types associated with the
980   resource type to create, modify and delete the resource.

4.2.3 PropertyType

982
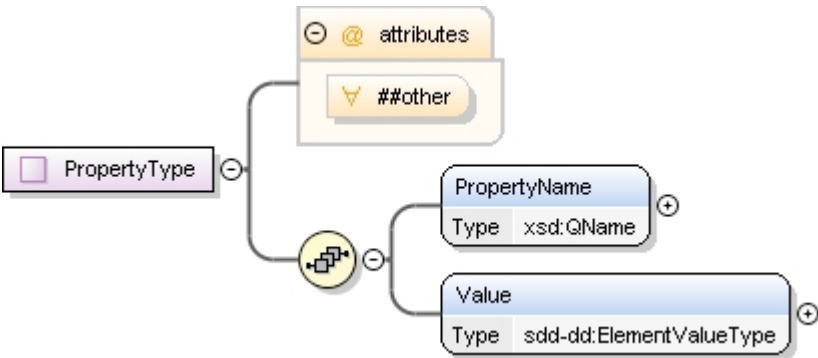
Figure 17: PropertyType structure.

984 *PropertyType* provides the type definition for elements used to declare an identity property of a resource,
985 namely, the *Property* elements of *Resource* and *HostedResource* in *Topology.* It also provides the type
986 definition for *Property* elements in *Relationship* and *RelationshipConstraint*.

4.2.3.1 PropertyType Property Summary

Name	Type	*	Description

Name	Type	*	Description
PropertyName	xsd:QName	1	The property name.
Value	ElementValueType	1	The property value.
	xsd:anyAttribute	0..*	

4.2.3.2 PropertyType Property Usage Notes

- **PropertyName**: The *PropertyName* MAY be used to provide identification for the resource in the deployment environment.

  The *PropertyName* MAY be used to provide constraints on the configuration of a resource.

- **Value**: Evaluation of the *Value* expression provides the value of the property.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

4.2.4 ResultingPropertyType
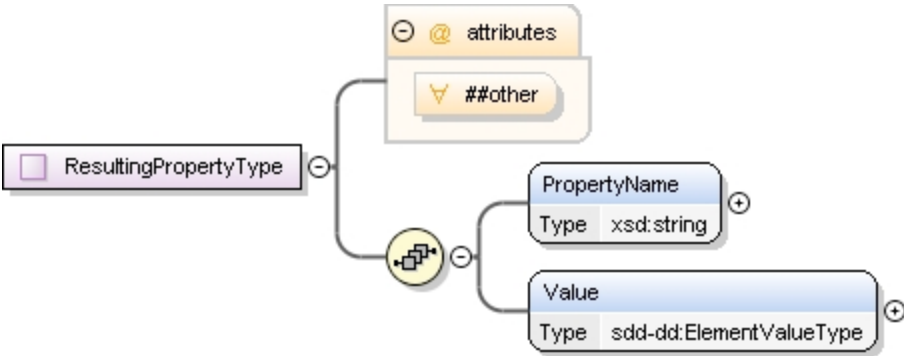
Figure 18: ResultingPropertyType structure.

ResultingPropertyType provides the type definition for elements used to declare an identity property of a resulting resource or to declare a configuration change to a resource property which results from deployment of an artifact.

4.2.4.1 ResultingPropertyType Property Summary

Name	Type	*	Description
PropertyName	xsd:string	1	The resulting property name.
Value	ElementValueType	1	The resulting property value.
	xsd:anyAttribute	0..*	Additional attributes of the resulting property.

4.2.4.2 ResultingPropertyType Property Usage Notes

- **PropertyName**: The *PropertyName* MAY be used to provide additional identification for the resource in the deployment environment.

  The *PropertyName* MAY be used to declare a configuration change to a resource.

- **Value**: Evaluation of the *Value* expression provides the value of the resulting property.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

4.3 Atomic Content Elements

The package descriptor defines package content that includes artifacts whose processing results in deployment of the software package. The deployment descriptor defines metadata associated with those

1010 artifacts. The metadata includes conditions, requirements, results, inputs, outputs and completion actions.
1011 Metadata throughout the deployment descriptor is associated with package content in the definition of
1012 atomic content elements. The atomic content elements are *InstallableUnit*, *ConfigurationUnit* and
1013 *LocalizationUnit*. These are the only content elements that define *Artifacts* elements.

1014 *Artifact* elements identify an artifact file or set of files defined in package content whose processing will
1015 perform all or a portion of the deployment for a particular deployment lifecycle operation. The name of the
1016 artifact element indicates the operation supported by the artifact. Names of the artifact elements are
1017 created by prefixing "Artifacts" with the operation name. The artifacts defined for use in the SDD are
1018 *InstallArtifact*, *UpdateArtifact*, *UndoArtifact*, *UninstallArtifact*, *RepairArtifact* and *ConfigArtifact*.

1019 *Artifact* elements define the inputs and outputs, substitution values and types associated with the artifact
1020 files. The content element's target resource, identified by *targetResourceRef*, processes the artifact files
1021 with the defined inputs to perform deployment operations. Examples of artifact types include zip files, rpm
1022 files and executable install files. Artifact types are not defined by this specification. The artifact types
1023 defined in the SDD need to be understood by software that processes the SDD.

1024 There MAY be multiple atomic content elements within a composite installable that describe the
1025 deployment of multiple resources as part of a single software deployment or there MAY be a single
1026 atomic content element (singleton) in the deployment descriptor that describes the entirety of a simple
1027 deployment. When an atomic content element is used in a *CompositeInstallable*, it MUST define exactly
1028 one artifact. When an atomic content element is a singleton, it MUST define at least one artifact element
1029 and MAY define one of each type of artifact element allowed for its type. The inclusion of an artifact
1030 element in a singleton atomic content element implies support for the associated operation.

1031 For example, a singleton *ConfigurationUnit* that defines a *ConfigArtifact* associates a configure
1032 operation with the *ConfigArtifact*. Similarly, an SDD with a singleton *InstallableUnit* that defines an
1033 *InstallArtifact* and an *UpdateArtifact* associates an *install* operation with the *InstallArtifact* and an
1034 *update* operation with the *UpdateArtifact*.

1035 When an atomic content element is defined within a *CompositeInstallable* hierarchy, its one artifact MUST
1036 support the single top level operation associated with the *CompositeInstallable*. The single artifact defined
1037 need not be an artifact for the operation defined for the *CompositeInstallable*.

1038 For example, in a *CompositeInstallable* that defines metadata for an *update* operation, there may be
1039 one *InstallableUnit* that defines an *InstallArtifact* element and another *InstallableUnit* that defines an
1040 *UpdateArtifact* element. Both of these artifacts are used when performing the overall *update* operation
1041 defined for the *CompositeInstallable*.

4.3.1 InstallableUnitType
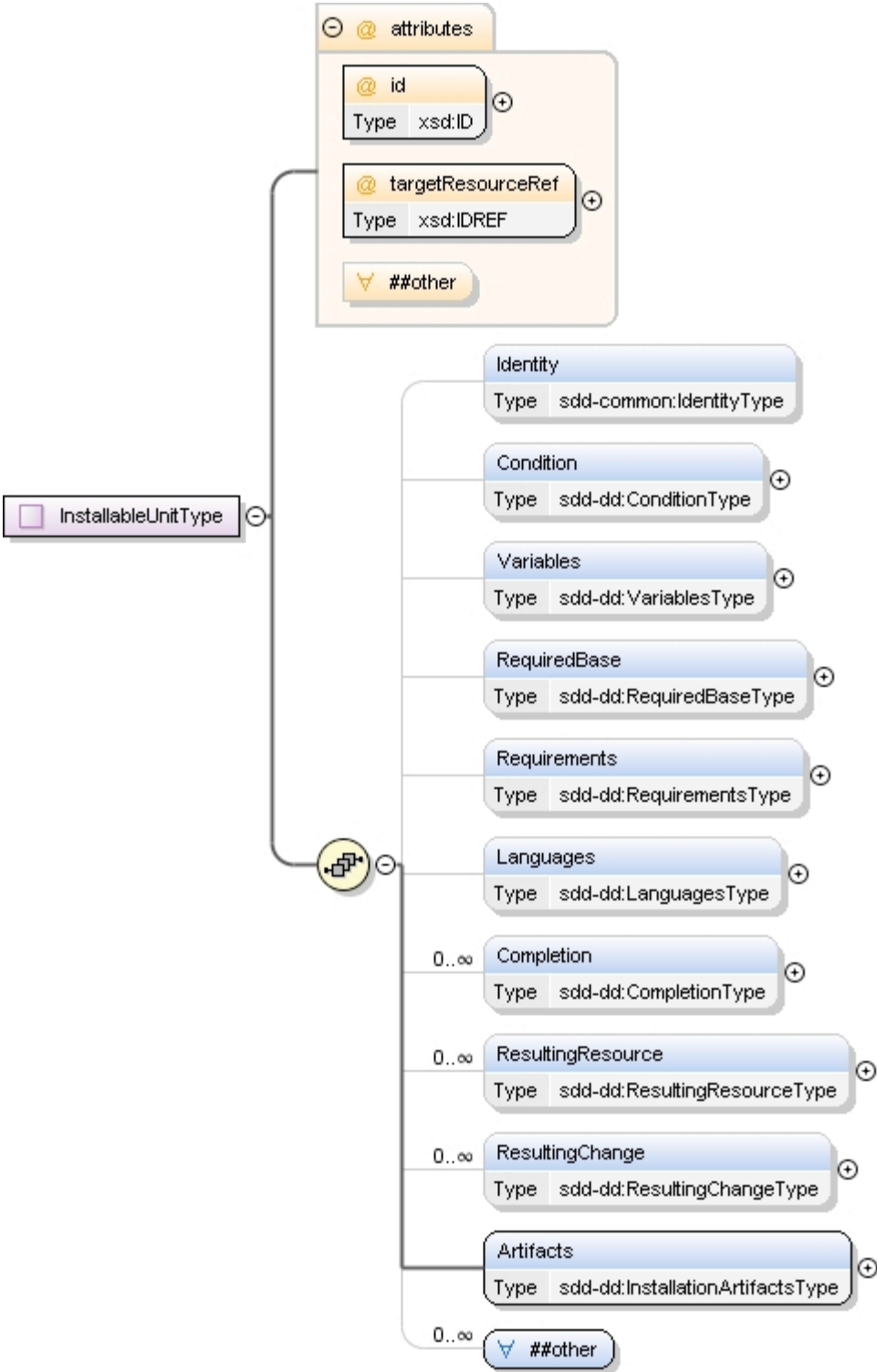
Figure 19: InstallableUnitType structure.

The *InstallableUnit* element is an atomic content element that defines artifacts that install or update
software and defines requirements for applying those artifacts. It may also define artifacts that undo an
update or that uninstall or repair existing software.

4.3.1.1 InstallableUnitType Property Summary

Name	Type	*	Description
Identity	IdentityType	0..1	Human-understandable identity information about the InstallableUnit.
Condition	ConditionType	0..1	A condition that determines if the content element is relevant to a particular deployment.
Variables	VariablesType	0..1	Variables for use within the InstallableUnit's requirements and artifact definitions.
RequiredBase	RequiredBaseType	0..1	A resource that will be updated when the InstallableUnit's UpdateArtifact is processed.
Requirements	RequirementsType	0..1	Requirements that must be met prior to successful processing of the InstallableUnit's artifacts.
Languages	LanguagesType	0..1	Languages supported by the InstallableUnit.
Completion	CompletionType	0..*	Describes completion actions such as restart and the conditions under which the action is applied.
ResultingResource	ResultingResourceType	0..*	A resource that will be installed or updated by processing the InstallableUnit's artifacts.
ResultingChange	ResultingChangeType	0..*	A resource that will be configured by processing the InstallableUnit's artifacts.
Artifacts	InstallationArtifactsType	1	The set of artifacts associated with the InstallableUnit.
	xsd:any	0..*	
id	xsd:ID	1	An identifier for the InstallableUnit scoped to the deployment descriptor.
targetResourceRef	xsd:IDREF	1	Reference to the resource that can process the InstallableUnit's artifacts.
	xsd:anyAttribute	0..*	

4.3.1.2 InstallableUnitType Property Usage Notes

- **Identity**: The *InstallableUnit's Identity* element defines human-understandable information that reflects the identity of the solution as understood by the end user of the solution.

  If the *InstallableUnit* defines a resulting resource, the *Identity* of the *InstallableUnit* SHOULD reflect the identity of the resulting resource.

  When the *InstallableUnit* is the only content element in the deployment descriptor, its *Identity* MAY define values that are the same as the corresponding *PackageIdentity* element values.

1056   This would be useful, for example, in a case where the package is known by the same name as
1057   the resource created by the *InstallableUnit*.

1058   See the *IdentityType* section for structure and additional usage details [3.4].

1059   ▪ **Condition**: A *Condition* is used when the *InstallableUnit's* content should be deployed only when
1060   certain conditions exist in the deployment environment.

1061   For example, one *InstallableUnit* may be applicable only when the operating system resource is
1062   resolved to a Linux®[2] operating system during deployment. The *InstallableUnit* would define a
1063   *Condition* stating that the type of the operating system must be Linux for the *InstallableUnit* to be
1064   considered in scope for a particular deployment.

1065   See the *ConditionType* section for structure and additional usage details [4.5.1].

1066   ▪ **Variables**: An *InstallableUnit's Variables* element defines variables that are used in the definition of
1067   the *InstallableUnit's* requirements and in parameters and properties passed to the *InstallableUnit's*
1068   target resource.

1069   When the deployment descriptor defines a single *InstallableUnit* at the top level, that is, not inside a
1070   *CompositeInstallable*, the variables it defines MAY be referred to by any element under *Topology*.

1071   See the *VariablesType* section for structure and additional usage details [4.6.5].

1072   ▪ **Languages**: When translated materials are deployed by the *InstallableUnit's* artifacts, the languages
1073   of the translations are listed in *Languages*.

1074   See the *LanguagesType* section for structure and additional usage details [4.13.6].

1075   ▪ **RequiredBase**: When an *InstallableUnit* can be used to update resources, the *RequiredBase*
1076   element identifies the resources that can be updated.

1077   See the *RequiredBaseType* section for structure and additional usage details [4.7.8].

1078   ▪ **Requirements**: *Requirements* specified in an *InstallableUnit* identify requirements that must be met
1079   prior to successful processing of the *InstallableUnit's* artifacts.

1080   See the *RequirementsType* section for structure and additional usage details [4.7.1].

1081   ▪ **Completion**: A *Completion* element MUST be included if the artifact being processed requires a
1082   system operation such as a reboot or logoff to occur to function successfully after deployment or if the
1083   artifact executes a system operation to complete deployment of the contents of the artifact.

1084   There MUST be an artifact associated with the operation defined by a *Completion* element.

1085   For example, if there is a *Completion* element for the *install* operation, the *InstallableUnit* must
1086   define an *InstallArtifact*.

1087   See the *CompletionType* section for structure and additional usage details [4.3.14].

1088   ▪ **ResultingResource**: An *InstallableUnit's ResultingResource* element identifies the resources in
1089   *Topology* that will be installed or updated when the *InstallableUnit's* artifacts are processed.

1090   See the *ResultingResourceType* section for structure and additional usage details [4.8.1].

1091   ▪ **ResultingChange**: Multiple content elements within the SDD MAY specify the same resource in their
1092   *ResultingChange* elements. In this case each content element is capable of modifying the
1093   configuration of that resource.

---

[2] Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

1094 An example use of the *ResultingChange* element is to understand whether or not one content
1095 element can satisfy the requirements specified in another content element.

1096 See the *ResultingChangeType* section for structure and additional usage details [4.8.2].

1097 ▪ **Artifacts**: When the *InstallableUnit* is a singleton defined outside of a *CompositeInstallable*, it MUST
1098 define at least one artifact element and MAY define one of each type of artifact element allowed for its
1099 type. The inclusion of an artifact element in a singleton *InstallableUnit* implies support for the
1100 associated operation.

1101 When the *InstallableUnit* is defined within a *CompositeInstallable*, it MUST define exactly one artifact.
1102 The artifact defined MAY be any artifact allowed in an *InstallableUnit* and it MUST support the single
1103 top level operation defined by the *CompositeInstallable*. This does not mean the operation associated
1104 with the artifact has to be the same as the one defined by the *CompositeInstallable*.

1105 For example, an update of a resource may be required to support an install of the overall solution,
1106 in which case the *InstallableUnit* would define an *UpdateArtifact* to support the top level *install*
1107 operation.

1108 See the *InstallationArtifactsType* section for structure and additional usage details [4.3.4].

1109 ▪ **id**: The *id* attribute is referenced in features to identify an *InstallableUnit* selected by the feature and
1110 *Dependency* elements to indicate a dependency on processing of the content element.

1111 The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
1112 log and trace messages.

1113 ▪ **targetResourceRef**: The *targetResourceRef* attribute identifies the resource that will process the
1114 *InstallableUnit's* artifacts.

1115 The resources created or modified by artifact processing are frequently, but not necessarily, hosted
1116 by the target resource.

1117 For example, an operating system may be the target resource of an artifact that is a zip file
1118 containing a J2EE application. However, when the J2EE application is deployed, a J2EE server is
1119 the host resource of the application. Thus, the OS hosts the artifact and the J2EE server hosts
1120 the J2EE application.

1121 This value MUST match an *id* of a resource element in *Topology*.

1122 The target may be a resource that has not yet been created. In this case, there is a dependency on
1123 the complete installation of the target resource prior to applying the *InstallableUnit*. This dependency
1124 MUST be represented in a *Dependency* element within *Requirements* that apply to the
1125 *InstallableUnit*.

4.3.2 ConfigurationUnitType
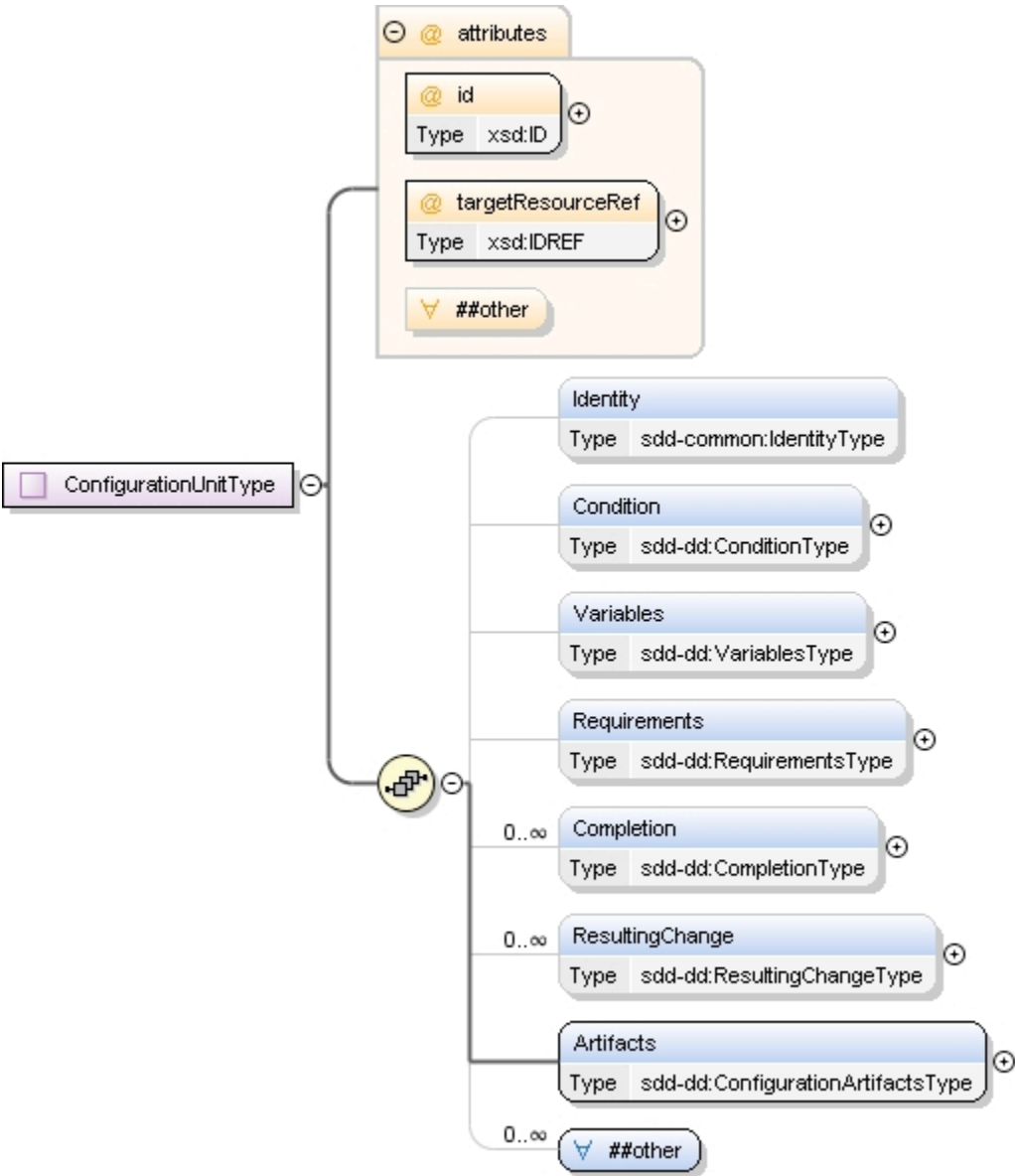
Figure 20: ConfigurationUnitType structure.

The *ConfigurationUnit* element defines artifacts that configure one or more existing resources. It also defines the requirements for applying those artifacts. It MUST NOT install, update, or uninstall resources.

4.3.2.1 ConfigurationUnitType Property Summary

Name	Type	*	Description
Identity	IdentityType	0..1	Human-understandable identity information about the ConfigurationUnit.
Condition	ConditionType	0..1	A condition that determines if the content element is relevant to a particular deployment.
Variables	VariablesType	0..1	Variables for use within the ConfigurationUnit's requirement and artifact definitions.

Requirements	RequirementsType	0..1	Requirements that must be met prior to successful processing of the ConfigurationUnit's artifacts.
Completion	CompletionType	0..*	Describes completion actions such as restart and the conditions under which the action is applied.
ResultingChange	ResultingChangeType	0..*	A definition of changes made to a resource that is configured by processing the ConfigurationUnit's ConfigArtifact.
Artifacts	ConfigurationArtifactsType	1	The artifact associated with the ConfigurationUnit.
	xsd:any	0..*	
id	xsd:ID	1	An identifier for the ConfigurationUnit scoped to the deployment descriptor.
targetResourceRef	xsd:IDREF	1	Reference to the resource that can process the ConfigurationUnit's artifacts.
	xsd:anyAttribute	0..*	

4.3.2.2 ConfigurationUnitType Property Usage Notes

- **Identity**: The *ConfigurationUnit's Identity* element defines human-understandable information that reflects the identity of the provided configuration as understood by the end user of the solution. *Identity* has elements that are common with elements in the corresponding *PackageDescriptor's PackageIdentity* element, for example, *Name* and *Version*. The values of these common elements SHOULD be the same as the corresponding *PackageIdentity* element values.

  See the *IdentityType* section for structure and additional usage details [3.4].

- **Condition**: A *Condition* is used when the deployment of configuration content is dependent on the existence of certain conditions in the deployment environment.

    For example, a package that has one configuration artifact that creates a database table for one database product and a different artifact that creates a table for a different database product would have two configuration units, each with a condition on the associated database product.

  See the *ConditionType* section for structure and additional usage details [4.5.1].

- **Variables**: A *ConfigurationUnit's Variables* element defines variables that are used in the definition of requirements and artifact parameters.

  When the deployment descriptor defines a single *ConfigurationUnit* at the top level, that is, not inside a *CompositeInstallable*, the variables it defines MAY be referred to by any element under *Topology*.

  See the *VariablesType* section for structure and additional usage details [4.6.5].

- **Requirements**: *Requirements* specified in a *ConfigurationUnit* identify requirements that MUST be met prior to successful processing of the *ConfigurationUnit's* artifacts.

  See the *RequirementsType* section for structure and additional usage details [4.7.1].

- **Completion**: A *Completion* element MUST be included if the artifact being processed requires a system operation such as a reboot or logoff to occur to function successfully after deployment or if the artifact executes a system operation to complete deployment of the contents of the artifact.

  There MUST be an artifact associated with the operation defined by a *Completion* element.

    For example, if there is a *Completion* element for the *configure* operation, the *ConfigurationUnit* must define a *ConfigArtifact*.

  See the *CompletionType* section for the structure and additional usage details [4.3.14].

- **ResultingChange**: Configuration changes made when the configuration artifact is processed SHOULD be declared here. This information may be necessary when the SDD is aggregated into another SDD and the resulting change satisfies a constraint in the aggregation. The information

1163	declared here can be compared with resource constraints to determine if application of the
1164	*ConfigurationUnit* will satisfy the constraint.
1165	See the *ResultingChangeType* section for structure and additional usage details [4.8.2].
1166	▪ **Artifacts**: When the *ConfigurationUnit* is a singleton defined outside of a *CompositeInstallable*, it
1167	MUST define at least one artifact element. The inclusion of an artifact element in a singleton
1168	*ConfigurationUnit* implies support for the associated operation.
1169	When the *ConfigurationUnit* is defined within a *CompositeInstallable*, it MUST define exactly one
1170	artifact. The artifact defined MUST be a *ConfigArtifact* and it MUST support the single top level
1171	operation defined by the *CompositeInstallable*.
1172	See the *ConfigurationArtifactsType* section for structure and additional usage details [4.3.5].
1173	▪ **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
1174	log and trace messages*.*
1175	▪ **targetResourceRef**: The *targetResourceRef* attribute identifies the resource in *Topology* that will
1176	process the *ConfigurationUnit's* artifacts to configure the resources identified by the
1177	*ConfigurationUnit's ResultingChange* definition.
1178	This value MUST match an *id* of a resource element in *Topology*.

4.3.3 ArtifactType

1179

1180
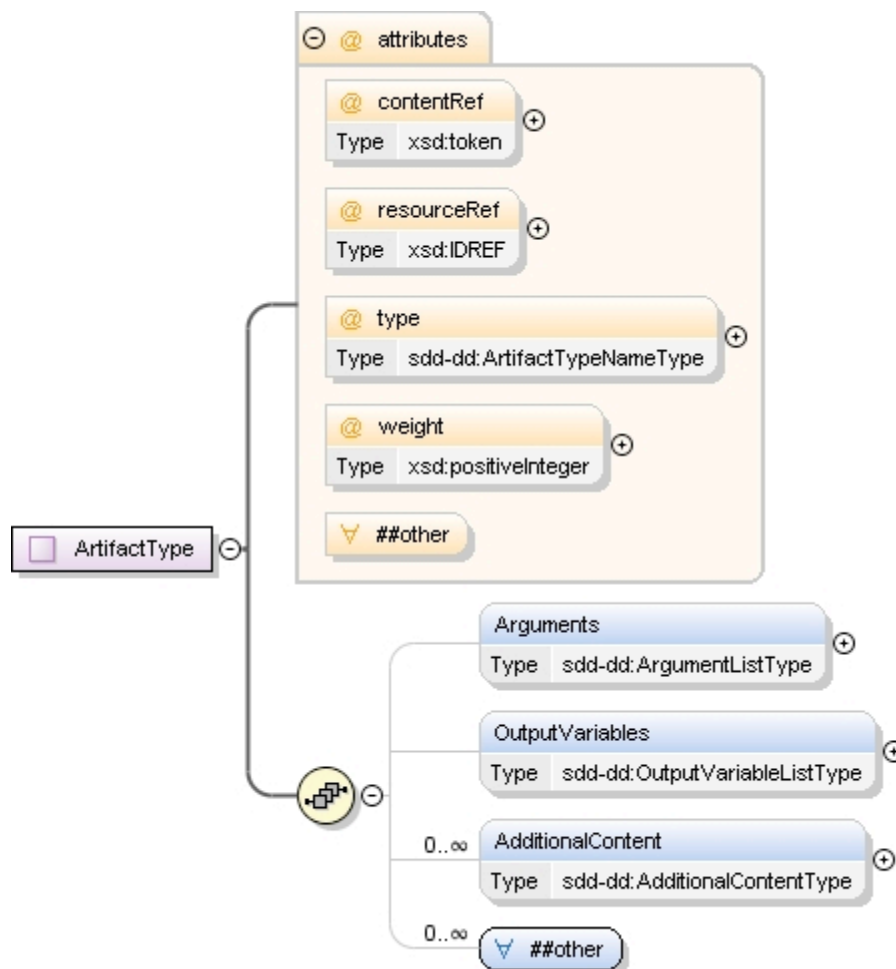
Figure 21: ArtifactType structure.

1181

1182 *ArtifactType* elements define the files, arguments and other information required to perform a particular
1183 deployment operation. Every artifact that can be defined in a content element is an instance of

1184   *ArtifactType*. These are *InstallArtifact, UpdateArtifact, UndoArtifact, UninstallArtifact, RepairArtifact, and*
1185   *ConfigArtifact*.

4.3.3.1 ArtifactType Property Summary
1186

Name	Type	*	Description
Arguments	ArgumentListType	0..1	Arguments used during processing of the artifact.
OutputVariables	OutputVariableListType	0..1	Variables whose values are set during processing of the artifact.
AdditionalContent	AdditionalContentType	0..*	Additional content files that are part of the artifact.
	xsd:any	0..*	
contentRef	xsd:token	0..1	The primary artifact file. Not used if resourceRef is used.
resourceRef	xsd:IDREF	0..1	The resulting resource representing the artifact file. Not used if contentRef is used.
type	ArtifactTypeNameType	0..1	Type of the primary artifact file.
weight	xsd:positiveInteger	0..1	The time required to process this artifact relative to all other artifacts in the SDD.
	xsd:anyAttribute	0..*	

4.3.3.2 ArtifactType Property Usage Notes
1187

1188   ■ **Arguments**: Inputs to the processing of the artifact MUST be specified by defining an *Arguments*
1189   element. All required inputs MUST be included in the arguments list. There are no implied arguments.

1190   For example, there is no implication that the selected required resource instances will be passed
1191   with an *InstallArtifact* on the install operation. If knowledge of those selections is required,
1192   instance identifiers should be passed as arguments.

1193   When one *Argument* refers to the *OutputVariable* of another artifact, the output value must be
1194   available at the time of processing the dependent artifact.

1195   For example, an artifact in a content element that is conditioned on the operating system being
1196   Linux should not refer to the output of an artifact in a content element conditioned on the
1197   operating system being Windows™[3].

1198   A *Dependency* requirement MUST be defined between the content elements to indicate that the
1199   artifact that defines the output variable is a pre-requisite of the content element with the dependent
1200   artifact.

1201   See the *ArgumentListType* section for structure and additional usage details [4.3.8].

1202   ■ **OutputVariables**: *OutputVariables* are variables whose values are set by artifact processing.

1203   *OutputVariables* can also be useful in log and trace messages.

---

[3] Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

1204   See the *OutputVariableListType* section for structure and additional usage details [4.3.10].

1205 ▪ **AdditionalContent**: *AdditionalContent* elements MUST be defined when supporting files are needed
1206   by the artifact for this operation. The content file reference is specified via the *contentRef* attribute of
1207   *AdditionalContent*.

1208   See the *AdditionalContentType* section for structure and additional usage details [4.3.12].

1209 ▪ **contentRef**: The value MUST be a reference to the *id* of the primary artifact file defined in a *Content*
1210   element in the package descriptor.

1211   Note that it is valid to have no artifact file and drive the operation from arguments alone.

1212   When more than one artifact file is needed, *contentRef* points to the primary artifact file and
1213   *AdditionalContent.contentRef* points to any other files used during application of the content element.

1214   When *resourceRef* is defined, *contentRef* MUST NOT be defined.

1215 ▪ **resourceRef**: Sometimes, artifact files are created during a deployment rather than being contained
1216   in the package.

1217   For example, some install programs create an uninstall program when the software is deployed.
1218   The uninstall program is the artifact file that is needed by the *UninstallArtifact*, but is created by,
1219   but not contained in, the package. In this case, the created artifact file is represented as a
1220   *ResultingResource*.

1221   An *Artifact* element that defines *resourceRef* identifies the resulting resource as its artifact file.

1222   When *contentRef* is defined, *resourceRef* MUST NOT be defined.

1223   The value MUST reference the *id* of a resource element in *Topology*.

1224 ▪ **type**: The *type* attribute identifies the format of the artifact file or files. When there is no artifact file
1225   identified, *type* MAY be left undefined. If there is an artifact file or additional files defined, *type* MUST
1226   be defined.

1227   Values for this attribute are not defined by this specification. *ArtifactTypeNameType* restricts *type* to
1228   valid `xsd:QNames`.

1229 ▪ **weight**: Defining weights for all artifacts and referenced packages in an SDD provides useful
1230   information to software that manages deployment. The weight of the artifact refers to the relative time
1231   taken to deploy the artifact with respect to other artifacts and referenced packages in this SDD.

1232   For example, if the artifact takes three times as long to deploy as another artifact whose weight is
1233   "2", then the weight would be "6". The weight numbers have no meaning in isolation and do not
1234   describe actual time elapsed. They simply provide an estimate of relative time.

4.3.4 InstallationArtifactsType
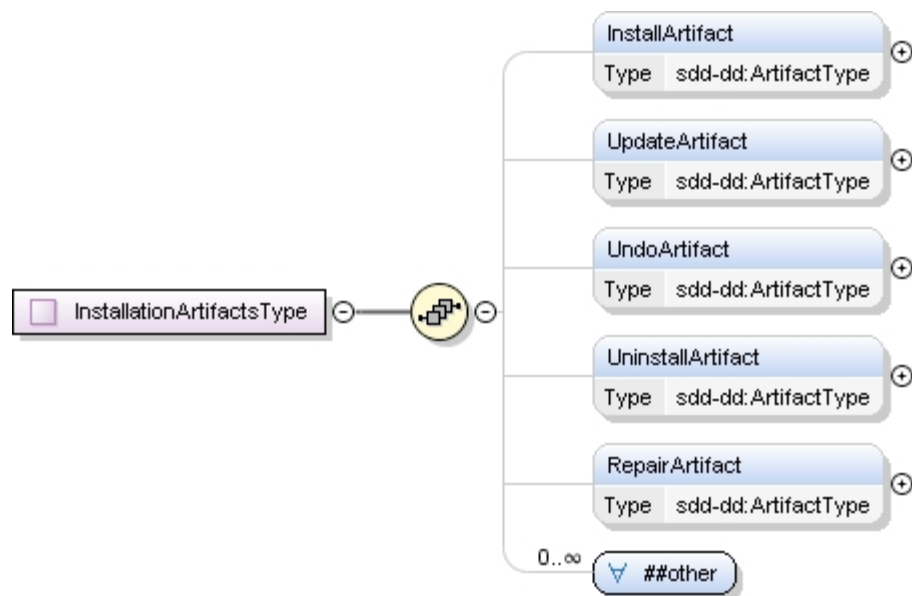
Figure 22: InstallationArtifactsType structure.

InstallationArtifactsType provides the type definition for the *Artifacts* element of *InstallableUnit* and *LocalizationUnit*. At least one *Artifact* element MUST be defined. Within a *CompositeInstallable* definition, exactly one *Artifact* element MUST be defined.

4.3.4.1 InstallationArtifactsType Property Summary

Name	Type	*	Description
InstallArtifact	ArtifactType	0..1	Artifact for install operation.
UpdateArtifact	ArtifactType	0..1	Artifact for update operation.
UndoArtifact	ArtifactType	0..1	Artifact for undo operation.
UninstallArtifact	ArtifactType	0..1	Artifact for uninstall operation.
RepairArtifact	ArtifactType	0..1	Artifact for repair operation.
	xsd:any	0..*	

4.3.4.2 InstallationArtifactsType Property Usage Notes

▪ **InstallArtifact**: The *InstallArtifact* element declares deployment information sufficient to enable the target resource to perform an install using the named artifact files. The *ResultingResource* and *ResultingChange* elements describe the characteristics of the new or modified resource(s).

See the *ArtifactType* section for structure and additional usage details [4.3.3].

▪ **UpdateArtifact**: The *UpdateArtifact* element declares deployment information sufficient to enable the target resource to perform an update using the named artifact files. The *RequiredBase* element defines the resource(s) that can be updated. The *ResultingResource* and *ResultingChange* elements describe the updated characteristics of the resource(s).

See the *ArtifactType* section for structure and additional usage details [4.3.3].

▪ **UndoArtifact**: The *UndoArtifact* element declares deployment information sufficient to enable the target resource to undo an update. This undo will put the resource back to a previous level.

1254 The update that can be undone is described in the *RequiredBase* element. The *ResultingResource*
1255 definition can be used to describe the state of the resource(s) after the undo completes.

1256 See the *ArtifactType* section for structure and additional usage details [4.3.3].

1257 ▪ **UninstallArtifact**: The *UninstallArtifact* element declares deployment information sufficient to enable
1258 the target resource to perform an uninstall.

1259 If an *InstallArtifact* is defined in the same *InstallableUnit*, the *ResultingResource* element defines the
1260 resource(s) that will be uninstalled.

1261 When an *UninstallArtifact* is the only artifact defined for an *InstallableUnit*, the *RequiredBase* MUST
1262 be defined to declare the resource(s) that will be uninstalled. The *ResultingResource* element MUST
1263 be left blank because the result of the uninstall is that the resource(s) are removed.

1264 See the *ArtifactType* section for structure and additional usage details [4.3.3].

1265 ▪ **RepairArtifact**: The *RepairArtifact* element declares deployment information sufficient to enable the
1266 target resource to repair an installation.

1267 If an *InstallArtifact* is defined in the same *InstallableUnit*, the *ResultingResource* element defines the
1268 resource(s) that will be repaired.

1269 When a *RepairArtifact* is the only artifact defined for an *InstallableUnit*, the *RequiredBase* MUST be
1270 defined to declare the resource(s) that will be repaired.

1271 See the *ArtifactType* section for structure and additional usage details [4.3.3].

1272 4.3.5 ConfigurationArtifactsType

1273
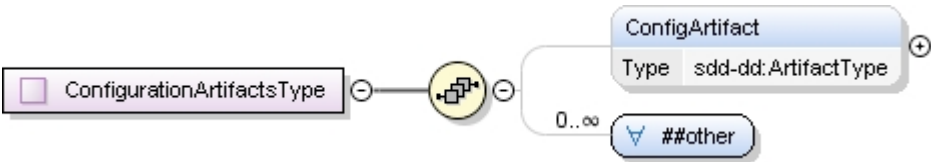
1274 **Figure 23: ConfigurationArtifactsType structure.**

1275 *ConfigurationArtifactsType* provides the type definition for the *Artifacts* element of *ConfigurationUnit.*

1276 4.3.5.1 ConfigurationArtifactsType Property Summary

Name	Type	*	Description
ConfigArtifact	ArtifactType	0..1	Artifact for configure operation.
	xsd:any	0..*	

1277 4.3.5.2 ConfigurationArtifactsType Property Usage Notes

1278 ▪ **ConfigArtifact**: The *ConfigArtifact* element declares deployment information sufficient to allow the
1279 target resource to configure the resources identified in the content element's *ResultingChange*
1280 elements.

1281 See the *ArtifactType* section for structure and additional usage details [4.3.3].

1282 4.3.6 OperationListType

1283 This simple type extends the `xsd:list` type as defined in **[XSD]**, and adds the restriction that each
1284 value in the list must be one of the operations from the enumeration defined by *OperationType* [4.3.7].

1285 4.3.7 OperationType

1286 Operations are used in the SDD to associate requirements and completion actions with particular
1287 artifacts.

1288 For example, when a requirement defines an *operation* attribute with value *undo*, it is a statement that
1289 the requirement must be met prior to processing of the undo artifact.

1290 *OperationType* enumerates the basic resource lifecycle operations that use the content and information
1291 defined in the SDD to change the state of the resources being installed, updated, or configured.

4.3.7.1 OperationType Property Usage Notes

1293 Attributes of *OperationType* MUST be set to one of the following values:

1294 ▪ **configure**: Uses the *ConfigArtifact* to perform configuration actions on a resource.

1295 ▪ **install**: Uses the *InstallArtifact* to install resources.

1296 ▪ **repair**: Uses the *RepairArtifact* to repair an installation.

1297 ▪ **undo**: Uses the *UndoArtifact* to restore a resource to the state before the most recent update was
1298 applied.

1299 ▪ **update**: Uses the *UpdateArtifact* to update an existing instance of a resource, as specified by the
1300 required base.

1301 ▪ **use**: Associates a requirement or completion action with use of the deployed software resources.
1302 Setting the operation attribute to *use* indicates that the requirement or completion action is not
1303 associated with an artifact.

1304 ▪ **uninstall**: Uses the *UninstallArtifact* to uninstall a resource.

4.3.8 ArgumentListType

1306
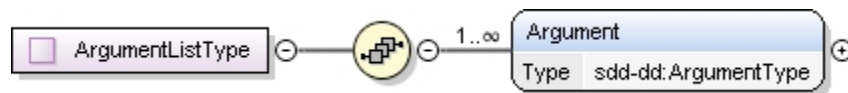
Figure 24: ArgumentListType structure.

1308 Each artifact MAY optionally include an *Arguments* element whose type is provided by *ArgumentListType*.
1309 This simply defines a list of *Argument* elements.

4.3.8.1 ArgumentListType Property Summary

Name	Type	*	Description
Argument	ArgumentType	1..*	An input to artifact processing.

4.3.8.2 ArgumentListType Property Usage Notes

1312 ▪ **Argument**: An argument value is a variable expression used to define a fixed value for the argument
1313 or to define a value in terms of one of the variables visible to the artifact.

1314 See the *ArgumentType* section for structure and additional usage details [4.3.9].

4.3.9 ArgumentType

1316
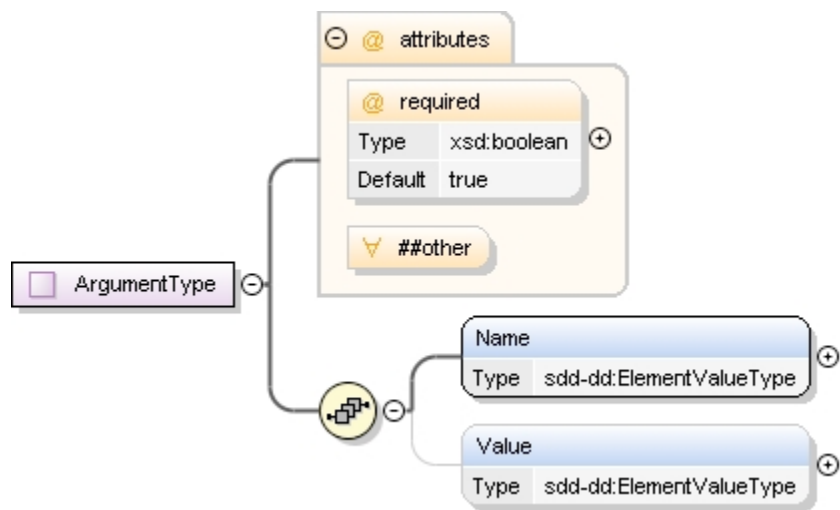
Figure 25: ArgumentType structure.

ArgumentType provides the type definition for *Argument* elements in artifacts [4.3.3]. This complex type is used to declare the argument name and optionally include a value for that argument.

4.3.9.1 ArgumentType Property Summary

Name	Type	*	Description
Name	ElementValueType	1	The argument name.
Value	ElementValueType	0..1	The argument value.
required	xsd:boolean	0..1	Indicates that the argument value must result in a valid expression for each particular deployment. **default value="true"
	xsd:anyAttribute	0..*	

4.3.9.2 ArgumentType Property Usage Notes

- **Name**: Evaluation of the *Name* expression produces the name of the argument. This can be useful for arguments with only a name, for example, those that are not name-value pairs.

  When the argument name alone is sufficient to communicate its meaning, the argument value SHOULD be omitted.

  If *Name* and *Value* elements are specified, the *Name* expression MUST be a literal string. Additionally, a *pattern* of wildcard is not supported and MUST NOT be used with the *Name* element.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

- **Value**: Evaluation of the *Value* expression provides the value of the argument.

  The variable expression MAY be used to define a fixed value for the argument or to define a value in terms of one of the variables visible to the artifact.

  A *pattern* of wildcard is not supported and MUST NOT be used with the *Value* element.See the *ElementValueType* section for structure and additional usage details [4.6.2].

- **required**: In cases where the argument should be ignored when the value expression is not valid for a particular deployment, set *required* to "false".

4.3.10 OutputVariableListType
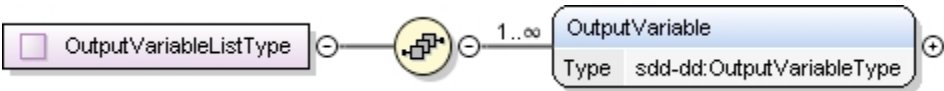
Figure 26: OutputVariableListType structure.

An artifact can set variables. The variables set by the artifact are defined in the artifact's *OutputVariables*.

4.3.10.1 OutputVariableListType Property Summary

Name	Type	*	Description
OutputVariable	OutputVariableType	1..*	An output from artifact processing.

4.3.10.2 OutputVariableListType Property Usage Notes

- **OutputVariable**: This is the definition of the variable, not a reference to a variable defined elsewhere. See the *OutputVariableType* section for structure and additional usage details [4.3.11].

4.3.11 OutputVariableType
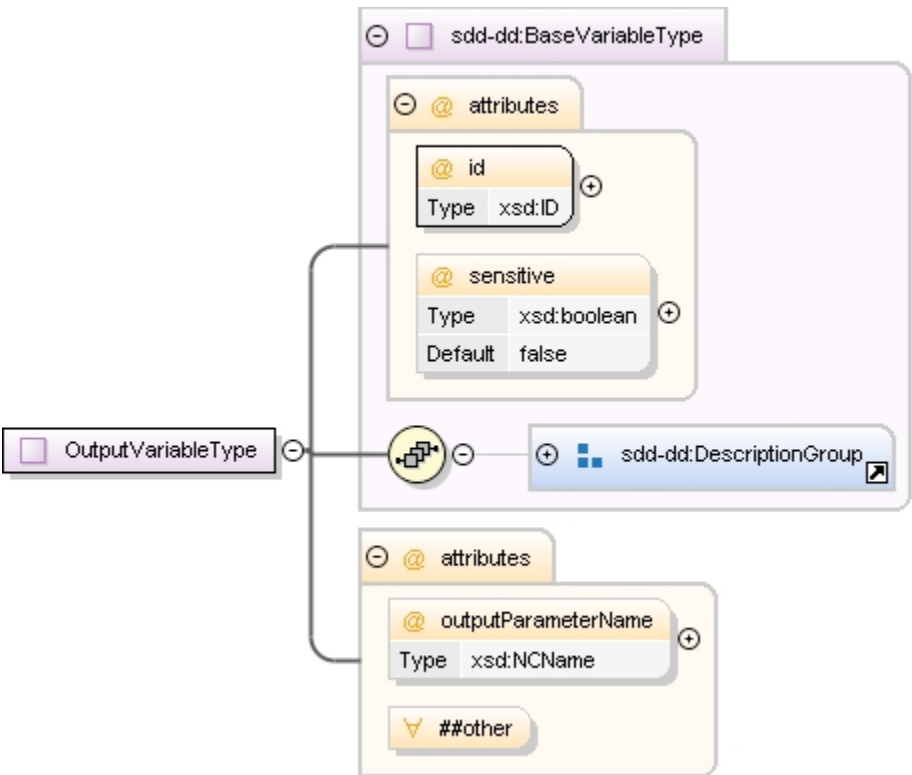
Figure 27: OutputVariableType structure.

Output variables are variables whose value is set by artifact processing. *OutputVariableType* extends *BaseVariableType* and so has all of the attributes defined there, including an *id* attribute that is used to refer to the output variable within the SDD. Output variables can be useful in log and trace messages.

4.3.11.1 OutputVariableType Property Summary

Name	Type	*	Description

	[extends] BaseVariableType		See the BaseVariableType section for additional properties [4.6.4].
outputParameterName	xsd:NCName	0..1	An output from artifact processing.
	xsd:anyAttribute	0..*	

4.3.11.2 OutputVariableType Property Usage Notes

See the *BaseVariableType* section for details about the inherited attributes and elements [4.6.4].

- **outputParameterName**: This is the name of the output variable as understood within the artifact processing environment. The output value is associated with the output variable's *id*. The SDD author uses this *id* within the SDD to refer to this output value.

4.3.12 AdditionalContentType
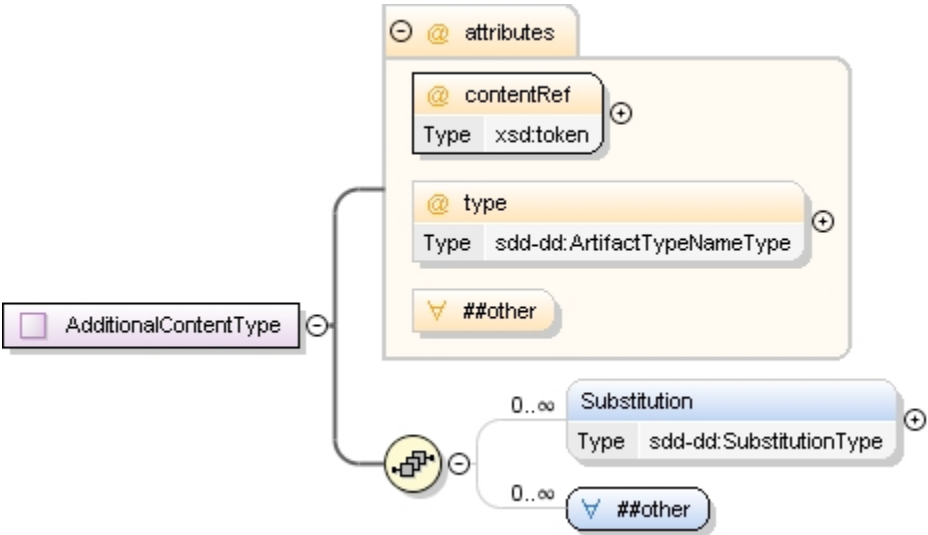
Figure 28: AdditionalContentType structure.

When artifact processing requires more than a single file, the artifact declaration includes information about the additional files needed. *AdditionalContentType* provides the type definition. Additional content MAY include input files that need to be edited to include values received as input to a particular solution deployment. In this case, the additional file can include a *Substitution* element.

4.3.12.1 AdditionalContentType Property Summary

Name	Type	*	Description
Substitution	SubstitutionType	0..*	A value to substitute into the file.
	xsd:any	0..*	
contentRef	xsd:token	1	A reference to the content element's id defined in the package descriptor.
type	ArtifactTypeNameType	0..1	Type of the additional artifact file.
	xsd:anyAttribute	0..*	

4.3.12.2 AdditionalContentType Property Usage Notes

- **Substitution**: The *Substitution* element supports the use of files that require some editing before they can be used in artifact processing. The definitions in this element support placement of values determined during a particular deployment into the file identified by the *contentRef* attribute.

  See the *SubstitutionType* section for structure and additional usage details [4.3.13].

- **contentRef**: The *contentRef* attribute points back to the package descriptor for information about the physical file. This value MUST match an *id* of a content element in the package descriptor.

- **type**: The *type* attribute identifies the format of the additional file. Values for this attribute are not defined by this specification. *ArtifactTypeNameType* restricts values of *type* to valid `xsd:QNames`.

4.3.13 SubstitutionType
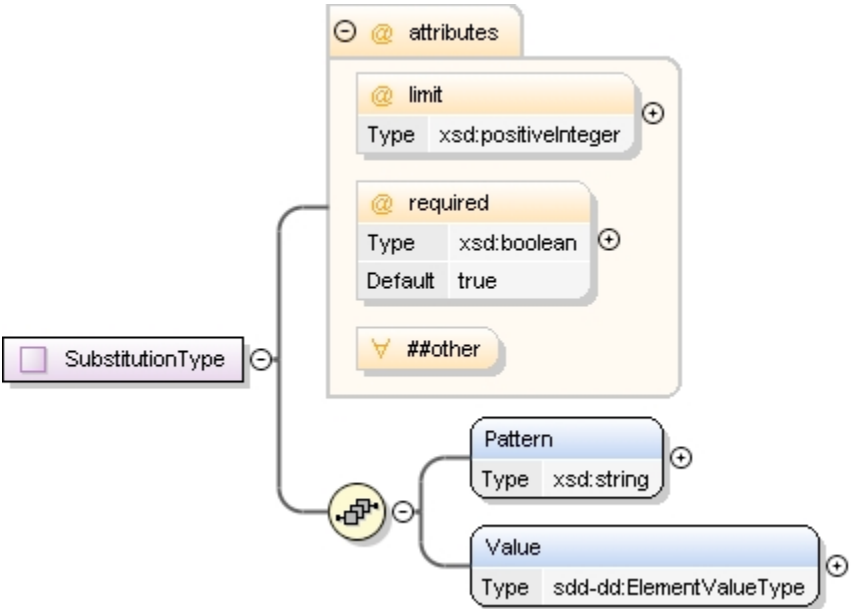
Figure 29: SubstitutionType structure.

SubstitutionType provides the type definition for the *Substitution* element in *AdditionalContent* declarations. It enables declaration of patterns in the file and the values that should replace the patterns before the file is used in artifact processing.

4.3.13.1 SubstitutionType Property Summary

Name	Type	*	Description
Pattern	xsd:string	1	The search pattern in the file that needs to be substituted.
Value	ElementValueType	1	The value to be substituted in the file.
limit	xsd:positiveInteger	0..1	The number of substitutions that should be made.
required	xsd:boolean	0..1	Indicates that substitution's value must result in a valid expression for each particular deployment. **default value="true"
	xsd:anyAttribute	0..*	

4.3.13.2 SubstitutionType Property Usage Notes

- **Pattern**: This is the substitution string that will be replaced with the *Value* when found in the file referenced by the *contentRef* attribute of the *AdditionalContent* element.

- **Value**: Evaluation of the *Value* expression results in the value that will be substituted for the pattern.

  A *pattern* of wildcard is not supported and MUST NOT be used with the *Value* element.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

- **limit**: If *limit* is not defined, there is no limit and all instances of the pattern found in the file will be replaced.

- **required**: In cases where the substitution should be ignored when the value expression is not valid for a particular deployment, set *required* to "false".

4.3.14 CompletionType
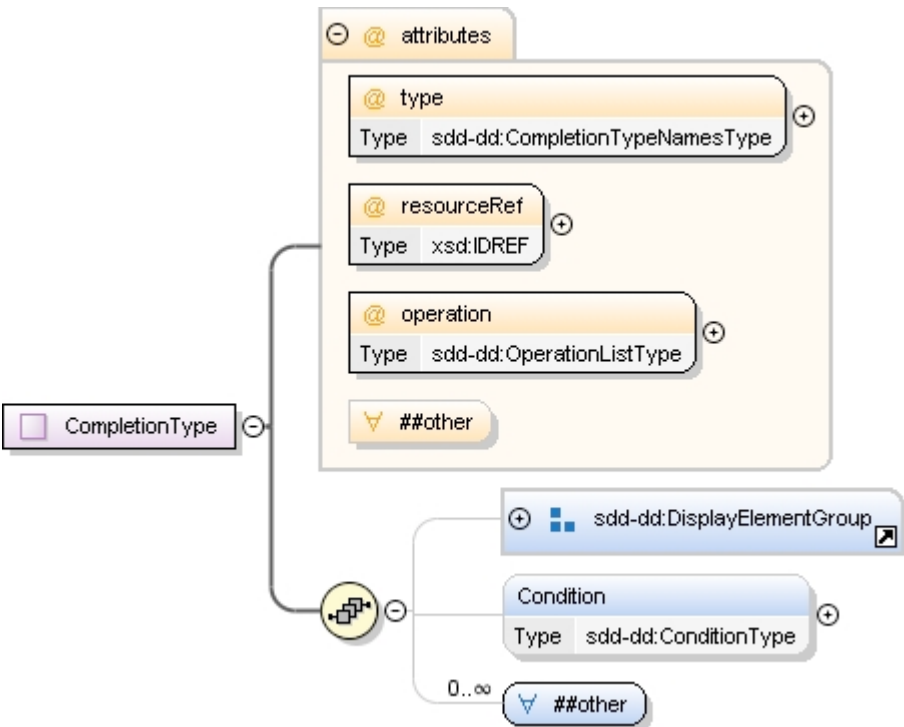
**Figure 30: CompletionType structure.**

For some deployments certain completion actions such as restart and logoff are required before a deployment operation using a particular content element can be considered complete. The *CompletionType* elements enable the SDD author to indicate either that one of these actions is required or that one of these actions will be performed by the associated artifact.

4.3.14.1 CompletionType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the completion action.
Description	DisplayTextType	0..1	Description of the completion action.
ShortDescription	DisplayTextType	0..1	Short description of the completion action.
Condition	ConditionType	0..1	Conditions that determine when the completion action will be used.

	xsd:any	0..*	
type	CompletionTypeNamesType	1	The type of the completion action.
resourceRef	xsd:IDREF	1	The resource where the completion action will be executed.
operation	OperationListType	1	Associates a completion action with the processing of a particular artifact.
	xsd:anyAttribute	0..*	

4.3.14.2 CompletionType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the *Completion* element.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the *Completion* element.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Condition**: *Conditions* specified on resource characteristics determine if the completion action applies. If the conditions are met, the action applies. If not met, then the action is not needed. Unmet conditions are not considered a failure. When no conditions are defined, the action always applies.

  See the *ConditionType* section for structure and additional usage details [4.5.1].

- **type**: This is the completion action that applies when conditions defined in *ResourceConstraint* are met. Allowed values defined in *CompletionTypeNameType* are:

  - **restartRequiredImmediately**: A system restart is required before the deployment operation is considered complete and the artifact associated with the operation does not perform the restart. The restart MUST happen before further deployment actions are taken.

  - **restartRequiredBeforeUse**: A system restart is required before the deployment operation is considered complete and the artifact associated with the operation does not perform this action. The restart MUST happen before the associated resources are used.

  - **restartOccurs**: The artifact associated with the lifecycle operation will initiate a system restart.

  - **logoffRequired**: A logoff and logon to the user account is required before the deployment operation is considered complete and the artifact associated with the operation does not perform this action. The logoff and logon MUST happen before the operation can be considered complete.

- **resourceRef**: This will often be the resource named as the target resource for the defining content element.

  The value MUST reference the *id* of a resource element in *Topology*.

- **operation**: A completion action is associated with the processing of one artifact by setting *operation* to the operation associated with that artifact. The element that defines the *Completion* MUST also define an artifact associated with the operation defined for the *Completion* element.

  See the *OperationListType* section for *operation* enumerations and their meaning [4.3.6].

4.4 Constraints

The SDD author needs to communicate constraints on resources for a variety of purposes.

- Some constraints must be met for the requirements of a content element to be met. See the *RequirementsType* section [4.7.1].

1435 ▪ Other constraints must be met for a resource to serve as the required base for an update. See the
1436 *RequiredBaseType* section [4.7.8].

1437 ▪ Still others must be met for to satisfy a condition that determines the applicability of a content element
1438 or completion action. See the *ConditionType* section [4.5.1] and the *CompletionType* section [4.3.14].

1439 The *Constraint* types described in this section support identification of resource constraints in these
1440 various contexts. These types are:

1441 ▪ *CapacityConstraint*

1442 ▪ *ConsumptionConstraint*

1443 ▪ *PropertyConstraint*

1444 ▪ *VersionConstraint*

1445 ▪ *UniquenessConstraint*

1446 ▪ *RelationshipConstraint*

1447 ▪ *AuthorizationConstraint*

1448 All of these constraint types are constraints on a resource. There are different constraint types because
1449 there are distinct semantics for each type of constraint. Examples are:

1450 • values within a certain range;

1451 • one of a set of values;

1452 • all of a set of values;

1453 • equal to a certain value;

1454 • no more than or no less than a certain value;

1455 • no more than or no less than a certain value when all constraints of that type are added
1456 together.

1457 In all cases, deployment software must be able to discover the property's value to honor the SDD author's
1458 intent.

1459 4.4.1 CapacityConstraintType

1460
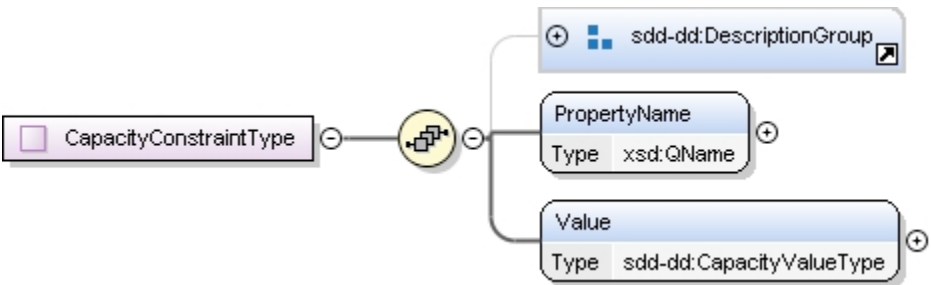
1461 **Figure 31: CapacityConstraintType structure.**

1462 *CapacityConstraintType* provides the type definition of the *Capacity* elements of
1463 *RequirementResourceConstraintType* [4.7.5]. These elements are used to express a requirement on the
1464 capacity of a particular resource property such as memory available from an operating system. Capacity
1465 is shared: multiple capacity constraints expressed on the same property are evaluated individually without
1466 assuming any change to the available quantity of the property.

1467 4.4.1.1 CapacityConstraintType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	A description of the capacity constraint. Required if ShortDescription is defined.

ShortDescription	DisplayTextType	0..1	A short description of the capacity constraint.
PropertyName	xsd:QName	1	Name of the constrained property.
Value	CapacityValueType	1	Bounds on the value of the constrained property.

4.4.1.2 CapacityConstraintType Property Usage Notes

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the capacity constraint on the resource.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

- **PropertyName**: This name corresponds to the name of the constrained resource property in the environment. This name may be specified in profiles [5.3].

- **Value**: *Value* specifies the bound and optional recommended bound on the resource property identified in the *PropertyName* element.

  See the *CapacityValueType* section for structure and additional usage details [4.4.2].

4.4.2 CapacityValueType
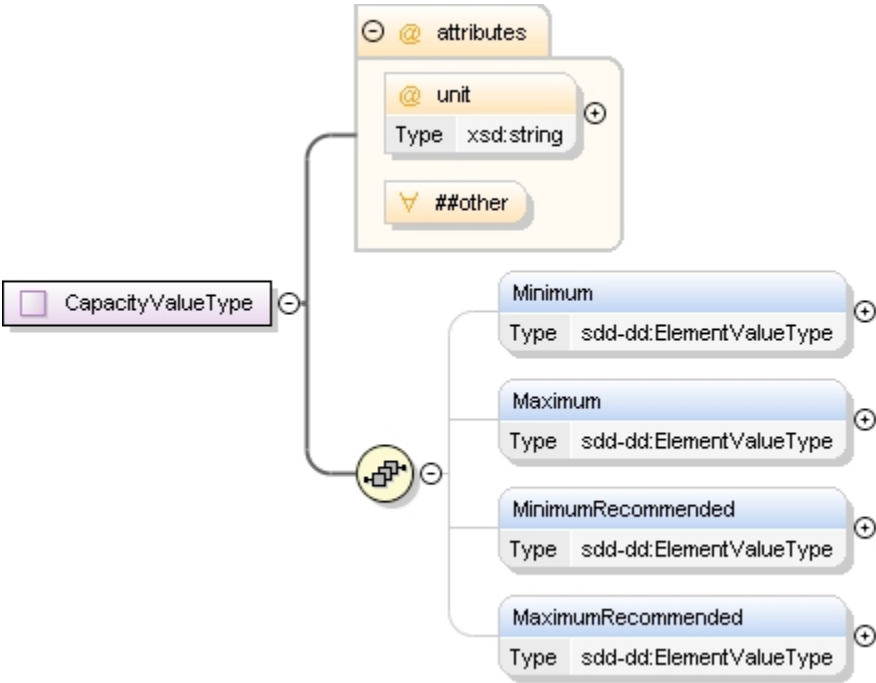
Figure 32: CapacityValueType structure.

Capacity value is expressed in terms of a minimum or maximum capacity. *CapacityValueType* provides the elements that support this expression. It also supports expression of a recommended minimum or maximum capacity.

4.4.2.1 CapacityValueType Property Summary

Name	Type	*	Description
Minimum	ElementValueType	0..1	Minimum capacity.
Maximum	ElementValueType	0..1	Maximum capacity.

MinimumRecommended	ElementValueType	0..1	Minimum recommended capacity.
MaximumRecommended	ElementValueType	0..1	Maximum recommended capacity.
unit	xsd:string	0..1	Unit of measure used to interpret the capacity value.
	xsd:anyAttribute	0..*	

4.4.2.2 CapacityValueType Property Usage Notes

- **Minimum**: There will usually be either a minimum value or a maximum value defined, but not both. When minimum is specified, the actual value of the capacity property MUST be equal to or greater than the minimum value.

  A *pattern* of wildcard is not supported and MUST NOT be used with the *Minimum* element.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

- **Maximum**: When specified, the actual value of the capacity property MUST be less than or equal to the defined maximum.

  If *Minimum* and *Maximum* are both defined, *Minimum* MUST be less than or equal to *Maximum*.

  A *pattern* of wildcard is not supported and MUST NOT be used with the *Maximum* element.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

  MinimumRecommended: The SDD author can indicate a preferred, but not required, minimum by defining a value for this element.A *pattern* of wildcard is not supported and MUST NOT be used with the *MinimumRecommended* element.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

- **MaximumRecommended**: The SDD author can indicate a preferred, but not required, maximum by defining a value for this element.

  If *MinimumRecommended* and *MaximumRecommended* are both defined, *MinimumRecommended* MUST be less than or equal to *MaximumRecommended*.

  A *pattern* of wildcard is not supported and MUST NOT be used with the *MaximumRecommended* element.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

- **unit**: Values for *unit* SHOULD be well-known units of measure from the International System of Units [UNIT]. A unit of measure SHOULD be specified for all properties that are measured in any kind of unit.

4.4.3 ConsumptionConstraintType

Figure 33: ConsumptionConstraintType structure.

ConsumptionConstraintType provides the type definition of the *Consumption* elements of *RequirementResourceConstraintType* [4.7.5]. These elements are used to express a requirement on the available quantity of a particular resource property such as disk space on a file system. *ConsumptionConstraints* represent exclusive use of the defined quantity of the resource property. In other words, consumption constraints are additive, with each consumption constraint specified in the SDD

1518 adding to the total requirement for the specified resource(s). A consumption constraint is assumed to alter
1519 the available quantity such that the portion of the property used to satisfy one constraint is not available to
1520 satisfy another consumption constraint on the same property.

1521 For example, suppose that the target file system has 80 megabytes available. The application of a
1522 content element's *InstallArtifact* results in installation of files that use 5 megabytes of file space. The
1523 application of a second *InstallArtifact* results in installation of files that use 2 megabytes of file space.
1524 Consumption constraints are additive, so the total space used for this content element is 7
1525 megabytes, leaving 73 (80–7) megabytes available on the target file system.

1526 4.4.3.1 ConsumptionConstraintType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	A description of the consumption constraint. Required if ShortDescription is defined.
ShortDescription	DisplayTextType	0..1	A short description of the consumption constraint.
PropertyName	xsd:QName	1	Names the resource property to test.
Value	ConsumptionConstraintValueType	1	A variable expression defining the minimum available quantity.

1527 4.4.3.2 ConsumptionConstraintType Property Usage Notes

1528 ▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable
1529 information. If used, they MUST provide a description of the consumption constraint on the resource.

1530 The *Description* element MUST be defined if the *ShortDescription* element is defined.

1531 See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

1532 ▪ **PropertyName**: The property name can be used to find the property value in the deployment
1533 environment. This name may be specified in profiles [5.3].

1534 ▪ **Value**: The result of evaluating this variable expression represents the minimum quantity of the
1535 named resource property that MUST be available for successful deployment of the defining content
1536 element's artifacts. This quantity will be consumed by application of the associated artifact.

1537 See the *ConsumptionConstraintValueType* section for structure and additional usage details [4.4.4].

4.4.4 ConsumptionConstraintValueType

Figure 34: ConsumptionConstraintValueType structure.

A consumption value is defined using a variable expression. *ConsumptionConstraintValueType* provides the variable expression by extending *ElementValueType*.

4.4.4.1 ConsumptionConstraintValueType Property Summary

Name	Type	*	Description
	[extends] ElementValueType		See the ElementValueType section for additional properties [4.6.2].
unit	xsd:string	0..1	Unit of measure used to interpret the consumption value.
	xsd:anyAttribute	0..*	

4.4.4.2 ConsumptionConstraintValueType Property Usage Notes

See the *ElementValueType* section for details about the inherited attributes and elements [4.6.2].

- **unit**: Values for *unit* SHOULD be well-known units of measure from International System of Units **[UNIT]**. A unit of measure SHOULD be specified for all properties which are measured in any kind of unit.
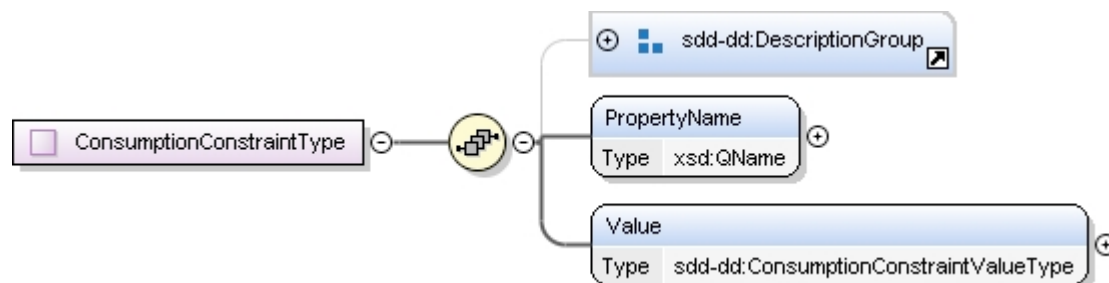
4.4.5 PropertyConstraintType
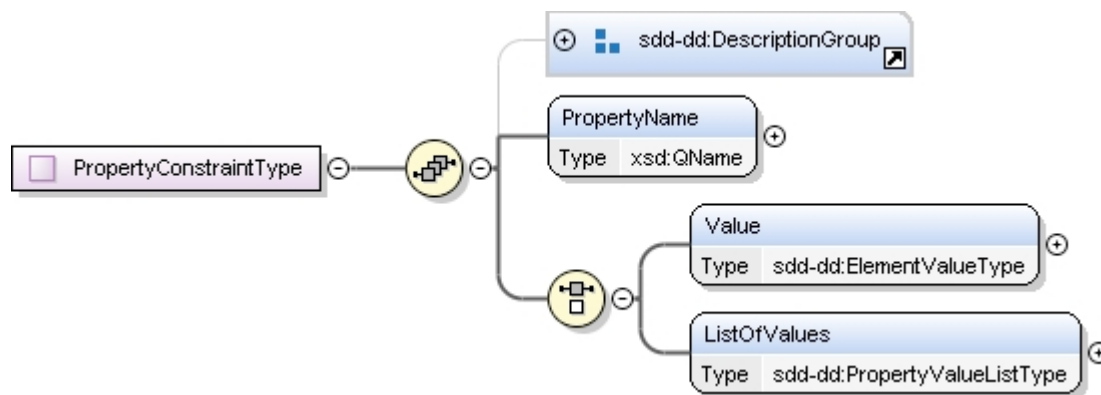
Figure 35: PropertyConstraintType structure.

PropertyConstraintType provides the type definition of the *Property* elements of *RequirementResourceConstraintType* [4.7.5]. It supports definition of a required value or set of acceptable values for a particular resource property.

4.4.5.1 PropertyConstraintType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	A description of the property constraint. Required if ShortDescription is defined.
ShortDescription	DisplayTextType	0..1	A short description of the property constraint.
PropertyName	xsd:QName	1	Name of the constrained property.
Value	ElementValueType	0..1	Required property value.
ListOfValues	PropertyValueListType	0..1	List of required property values.

4.4.5.2 PropertyConstraintType Property Usage Notes

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the property constraint on the resource.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

- **PropertyName**: The property name can be used to find the property value in the deployment environment. This name may be specified in profiles [5.3].

- **Value**: The result of evaluating the *Value* expression represents the required value of the named resource property.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

- **ListOfValues**: A list of required values can be defined in place of a single required value.

  See the *PropertyValueListType* section for structure and additional usage details [4.4.6].

4.4.6 PropertyValueListType

1569
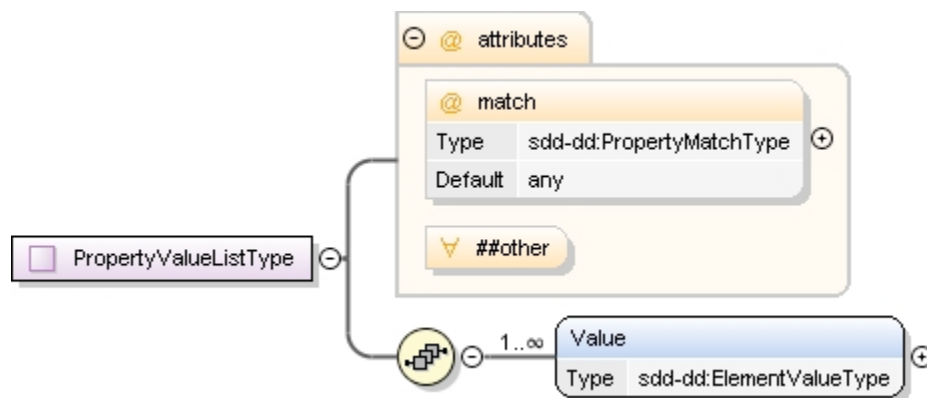
Figure 36: PropertyValueListType structure.

A property value list is expressed as one or more strings representing valid values for the property.

4.4.6.1 PropertyValueListType Property Summary

Name	Type	*	Description
Value	ElementValueType	1..*	A property value.
match	PropertyMatchType	0..1	Determines whether the actual property value must match any or all of the listed values. **default value="any"
	xsd:anyAttribute	0..*	

4.4.6.2 PropertyValueListType Property Usage Notes

- **Value**: The result of the *Value* expression represents one possible required value of the named resource property.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

- **match**: The value or values of the property found in the deployment environment are compared to the value or values listed in the property constraint. *PropertyMatchType* defines two enumerated values: *any* and *all*. When *match* is set to *any*, the property constraint is considered met when any one of the found property values matches any one of the declared property values. When *match* is set to *all*, the constraint is considered met when all of the declared property values match values found for the property.

4.4.7 VersionConstraintType

1584
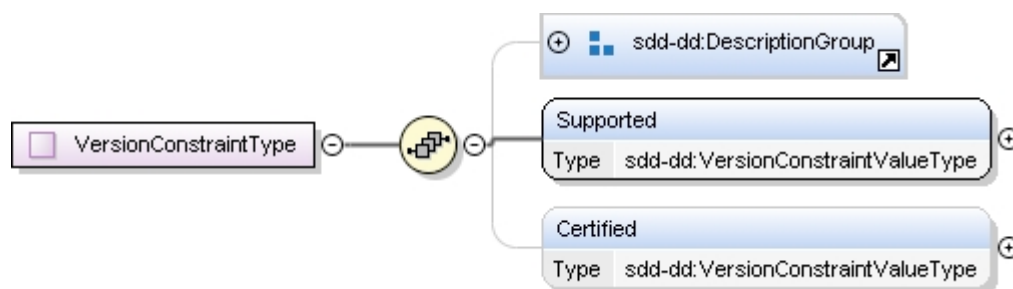
Figure 37: VersionConstraintType structure.

1586 *VersionConstraintType* provides the type definition of the *VersionConstraint* elements of
1587 *RequirementResourceConstraintType* [4.7.5]. A *VersionConstraint* can define a set of individual versions
1588 or ranges of versions that are supported and a similar set that are certified. During deployment, the set of
1589 certified versions, if provided, will be resolved and if successful, prioritized first. If the version constraint is
1590 still unmet after processing the certified versions, then the supported versions will be resolved.

4.4.7.1 VersionConstraintType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	A description of the version constraint. Required if ShortDescription is defined.
ShortDescription	DisplayTextType	0..1	A short description of the version constraint.
Supported	VersionConstraintValueType	1	A supported version or set of versions.
Certified	VersionConstraintValueType	0..1	A subset of the supported versions that are certified as tested.

4.4.7.2 VersionConstraintType Property Usage Notes

1593 ▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable
1594 information. If used, they MUST provide a description of the version constraint on the resource.

1595 The *Description* element MUST be defined if the *ShortDescription* element is defined.

1596 See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

1597 ▪ **Supported**: If the version of a resource is in the *Supported* set, it meets the requirements.

1598 See the *VersionConstraintValueType* section for structure and additional usage details [4.4.8].

1599 ▪ **Certified**: In some cases the set of required versions may be different from the set of versions that
1600 are certified by the manufacturer as thoroughly tested. Version(s) declared as *Certified* MUST be a
1601 subset of the version(s) declared as *Supported*.

1602 See the *VersionConstraintValueType* section for structure and additional usage details [4.4.8].
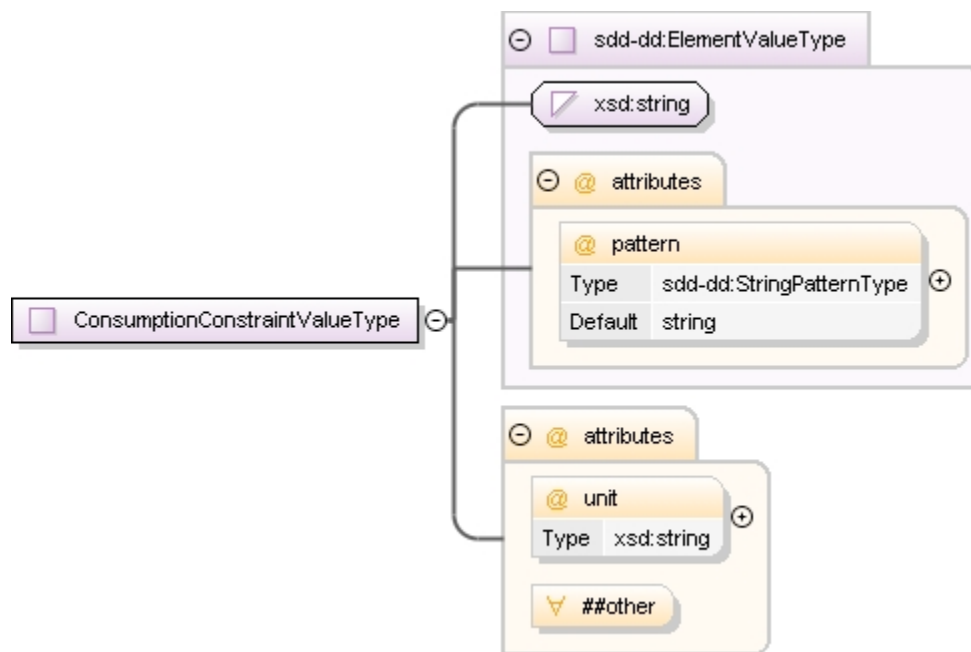
4.4.8 VersionConstraintValueType
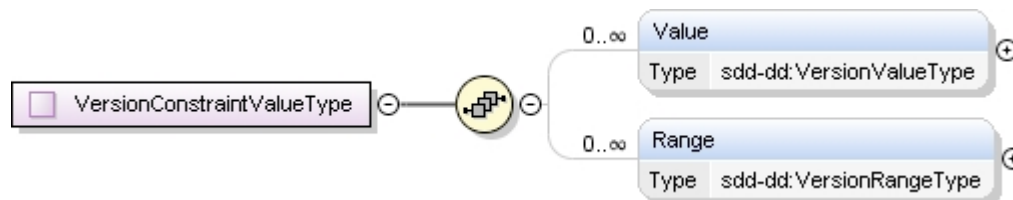
1605 **Figure 38: VersionConstraintValueType structure.**

1606 A version constraint can be specified using any number of individual version values in combination with
1607 any number of version ranges.

4.4.8.1 VersionConstraintValueType Property Summary

Name	Type	*	Description
Value	VersionValueType	0..*	A version value with associated fixes specified.
Range	VersionRangeType	0..*	A range of version values with associated fixes specified for each range.

4.4.8.2 VersionConstraintValueType Property Usage Notes

- **Value**: Discrete version values can be defined when the set of required versions includes versions that do not fall within a range. There is no assumption by this specification that version values are numerically comparable. The method of comparing version values may be resource-specific.

  See the *VersionValueType* section for structure and additional usage details [4.4.9].

- **Range**: See the *VersionRangeType* section for structure and additional usage details [4.4.10].

4.4.9 VersionValueType
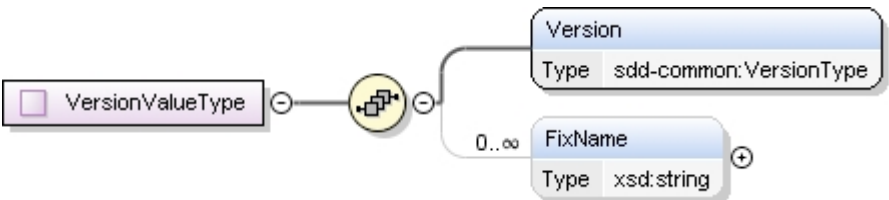
Figure 39: VersionValueType structure.

A version value includes a version and a list of required fixes associated with that version.

4.4.9.1 VersionValueType Property Summary

Name	Type	*	Description
Version	VersionType	1	An allowable version value.
FixName	xsd:string	0..*	The name of a fix.

4.4.9.2 VersionValueType Property Usage Notes

- **Version**: A string containing a single, exact version value. This is compared with the version value of specific resource instances. Only equal values satisfy this part of the constraint.

  See the *VersionType* section for structure and additional usage details [3.10].

- **FixName**: Any number of *FixName* elements can be defined; identifying fixes that must be discovered to be applied for the version constraint to be considered met.

4.4.10 VersionRangeType
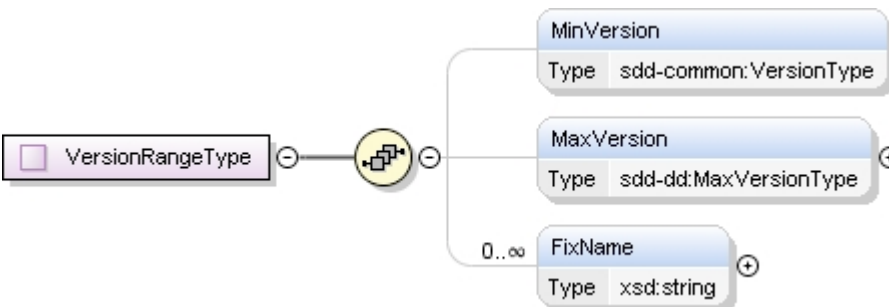
Figure 40: VersionRangeType structure.

A *VersionRange* is specified with a minimum and maximum version value and a list of required fixes associated with that range. The method of comparing version strings in a version range is resource-specific.

4.4.10.1 VersionRangeType Property Summary

Name	Type	*	Description
MinVersion	VersionType	0..1	The least allowable version value.
MaxVersion	MaxVersionType	0..1	The greatest allowable version value.
FixName	xsd:string	0..*	The name of a fix.

4.4.10.2 VersionRangeType Property Usage Notes

- **MinVersion**: This is the lower bound of a version range. If *MinVersion* is defined but *MaxVersion* is not, there is no upper bound. A version that is equal to *MinVersion* is within the defined range.

  See the *VersionType* section for structure and additional usage details [3.10].

- **MaxVersion**: This is the upper bound of a version range. If *MaxVersion* is defined but *MinVersion* is not, there is no lower bound. A version that is equal to *MaxVersion* may be within the defined range depending on the value specified for the *inclusive* attribute.

  See the *MaxVersionType* section for structure and additional usage details [4.4.11].

- **FixName**: Any number of *FixNames* can be defined identifying fixes that must be found to be applied for the version constraint is to be considered satisfied. This is true for all versions within the defined range.

  When *FixName* is defined, either a *MinVersion* or a *MaxVersion* element MUST also be defined.

4.4.11 MaxVersionType
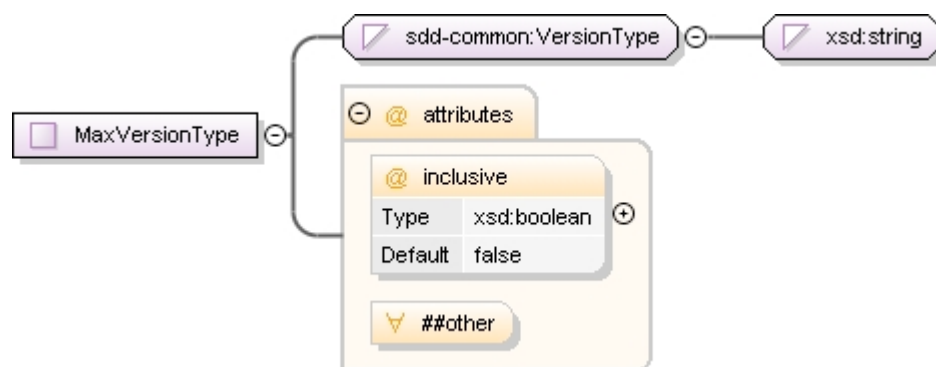
Figure 41: MaxVersionType structure.

A maximum version can be inclusive or exclusive.

4.4.11.1 MaxVersionType Property Summary

Name	Type	*	Description
	[extends] VersionType		See the VersionType section for additional properties [3.10].
inclusive	xsd:boolean	0..1	Indicates whether the max version value is included in the supported range of versions. **default value="false"
	xsd:any	0..*	

4.4.11.2 MaxVersionType Property Usage Notes

See the *VersionType* section for details about the inherited attributes and elements [3.10].

1652   ■   **inclusive**: The *inclusive* attribute allows the SDD author to choose the semantics of maximum
1653   version. Supported ranges are often everything equal to or greater than the minimum version and up
1654   to, but not including, the maximum version. Sometimes it is more convenient for the range to include
1655   the maximum version.

4.4.12 UniquenessConstraintType

1656
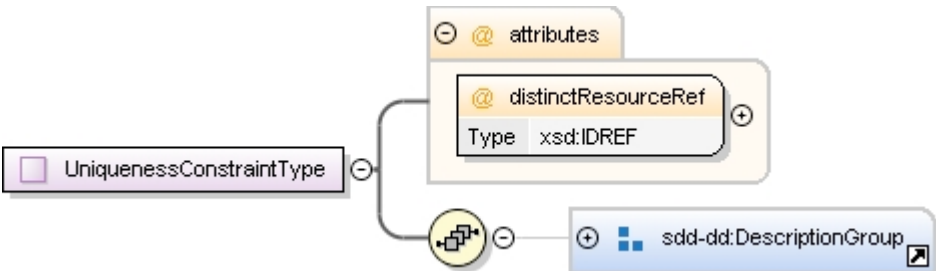
1657
1658   **Figure 42: UniquenessConstraintType structure.**

1659   A *UniquenessConstraint* is used to indicate when two resources defined in topology MUST or MUST NOT
1660   resolve to the same resource instance during a particular deployment. A *UniquenessConstraint* indicates
1661   that the two resources MUST NOT be the same when it is defined in a *ResourceConstraint* element with
1662   testValue="true". A *UniquenessConstraint* indicates that the two resources MUST be the same when
1663   defined in a *ResourceConstraint* with testValue="false".

1664   When no *UniquenessConstraint* is in scope for a particular pair of resources, the two resources MAY
1665   resolve to the same resource when their identifying characteristics are the same and when all in-scope
1666   constraints on both resources are satisfied.

1667   The first of the pair of resources is identified in the *resourceRef* attribute of the *ResourceConstraint*
1668   element that defines the *UniquenessConstraint*. The second of the pair is identified in the
1669   *distinctResourceRef* attribute of the *UniquenessConstraint*.

4.4.12.1 UniquenessConstraintType Property Summary

1670

Name	Type	*	Description
Description	DisplayTextType	0..1	A description of the uniqueness constraint, for example what must or must not be unique and why.
ShortDescription	DisplayTextType	0..1	A short description of the uniqueness constraint.
distinctResourceRef	xsd:IDREF	1	One of the pair of resources referred to by the constraint.

4.4.12.2 UniquenessConstraintType Property Usage Notes

1671

1672   ■   **Description, ShortDescription**: These elements MAY be used to provide human-understandable
1673   information. If used, they MUST provide a description of the uniqueness constraint on the resource.

1674   The *Description* element MUST be defined if the *ShortDescription* element is defined.

1675   See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

1676   ■   **distinctResourceRef**: The second resource in the pair of resources.

1677   The value MUST reference the *id* of a resource element in *Topology*.

4.4.13 RelationshipConstraintType
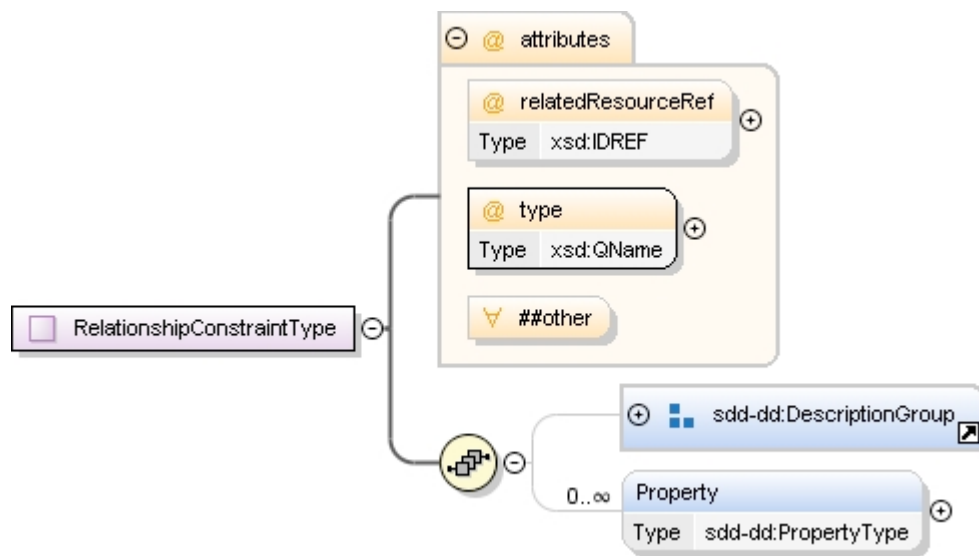
Figure 43: RelationshipConstraintType structure.

A *RelationshipConstraint* identifies a particular relationship between two resources that is constrained in some way by the SDD. The value of the *testValue* attribute of the *ResourceConstraint* that contains the *RelationshipConstraint* determines whether the constraint MUST be satisfied or MUST NOT be satisfied.

The first resource of the pair is defined by the *resourceRef* attribute of the *ResourceConstraint* containing the *RelationshipConstraint*.

4.4.13.1 RelationshipConstraintType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	A description of the relationship and its purpose in the overall solution.
ShortDescription	DisplayTextType	0..1	A short description of the relationship.
Property	PropertyType	0..*	A property constraint that further constrains the relationship.
relatedResourceRef	xsd:IDREF	0..1	The second resource in the relationship.
type	xsd:QName	1	The type of the relationship.
	xsd:anyAttribute	0..*	

4.4.13.2 RelationshipConstraintType Property Usage Notes

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the relationship constraint on the resource.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

- **Property**: This element MAY be used to provide additional constraints on the relationship.

  For example, a connectivity relationship might specify additional information such as the specific protocol used (for instance, TCP/IP) and/or particular characteristics of a protocol (for instance, port number).

  See the *PropertyType* section for structure and additional usage details [4.2.3].

1697	▪ **relatedResourceRef**: Naming the second resource is optional. When it is not named, the relationship
1698	constraint is satisfied if the first resource has the defined relationship with any other resource.
1699	When it is named, the value MUST reference the *id* of a resource element in *Topology*.
1700	▪ **type**: Values for relationship type are not defined by the SDD specification.

4.4.14 AuthorizationConstraintType
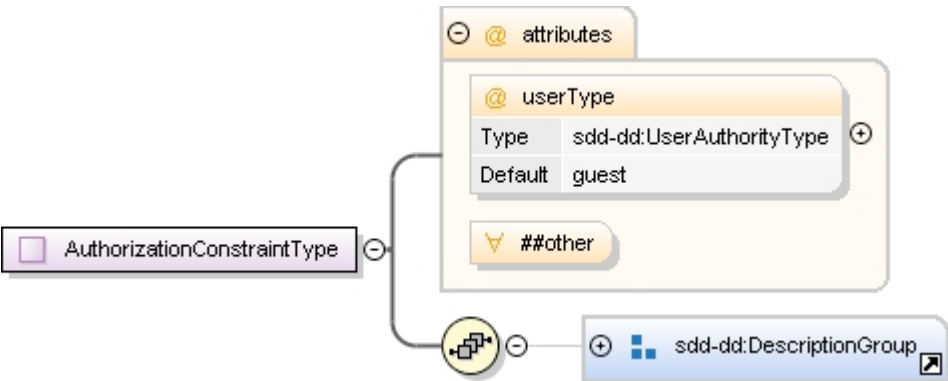
Figure 44: AuthorizationConstraintType structure.

An *AuthorizationConstraint* enables the SDD author to declare the level of authority that is required by the target resource in order to deploy the content that is described by the SDD.

> For example, if the referenced resource for which this contraint applies is an operating system–the SDD author would use this constraint to declare that root authority is required. This pattern similarly applies to databases, application servers, etc.

4.4.14.1 AuthorizationConstraintType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	A description of the authorization level and its purpose in the overall solution.
ShortDescription	DisplayTextType	0..1	A short description of the authorization level.
userType	UserAuthorityType	0..1	The type of the authority, for example, "guest" or "root". **default value="guest".
	xsd:anyAttribute	0..*	

4.4.14.2 AuthorizationConstraintType Property Usage Notes

▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the authorization constraint on the resource.

The *Description* element MUST be defined if the *ShortDescription* element is defined.

See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

▪ **userType**: If specified, the *userType* attribute designates the level of authority required by the referenced resource.

> For example, setting the *userType* to "guest" implies that the resource does not need a level of authoritization higher than a common user. However, if *userType* is set to "root", the resource requires an elevated level of administrative authority in order to successfully function.

4.5 Conditions

Conditions are expressed on characteristics of resources in the deployment environment. Conditions are used to indicate when particular elements of the SDD are applicable, or when they should be ignored. Conditions are not requirements. Failure to satisfy a condition does not indicate a failure; it simply means the conditioned element should be ignored. Conditions are used to:

- determine if a content element is applicable
- choose from among values for a variable
- determine when a feature is applicable
- determine when a particular result is applicable
- determine if a particular completion action is necessary.

Because conditions are always based on the characteristics of resources, they are expressed using resource constraints.

4.5.1 ConditionType
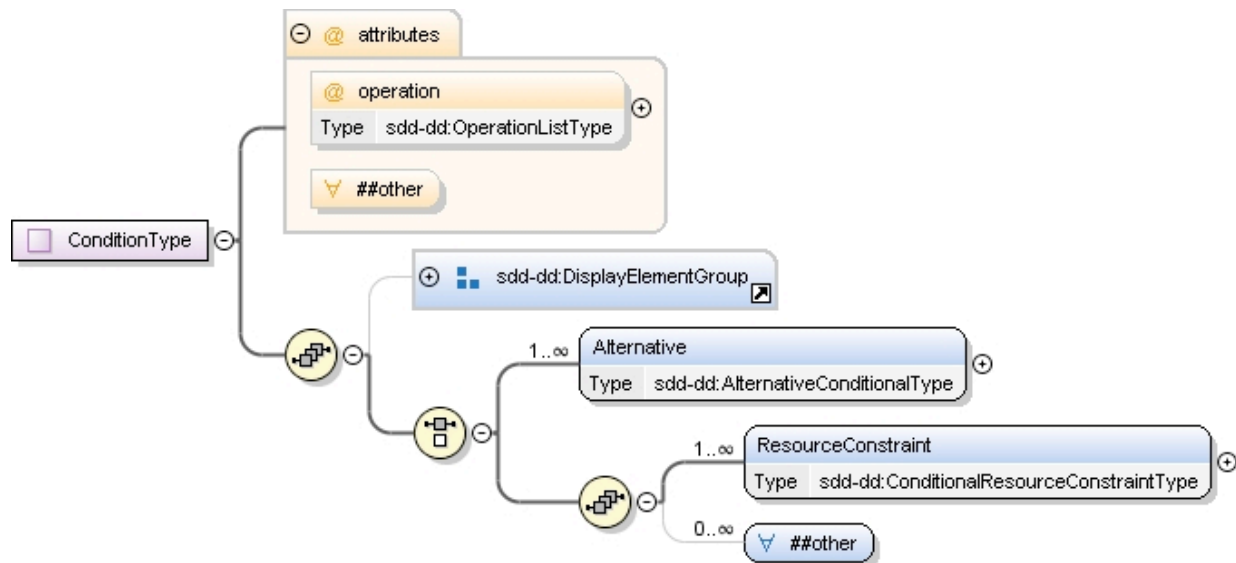
Figure 45: ConditionType structure.

ConditionType allows expression of the particular resource characteristics that must be true for the condition to be considered met. These are resource characteristics that may vary from one particular deployment to another.

For example, one deployment using the SDD might use one version of an application server and a different deployment might use a different version. The differences in the version might be great enough to:

- select among content elements.

  For example, one content element has an artifact for a Web application that works in a particular version and a different content element has an artifact for a later version of the same Web application.

- select among variable values.

  For example, the default installation path on one operating system may be different from the default install path on another operating system.

- select among completion actions.

  For example, a reboot may be required when deploying on one operating system but not another.

4.5.1.1 ConditionType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the condition.
Description	DisplayTextType	0..1	Description of the condition.
ShortDescription	DisplayTextType	0..1	Short description of the condition.
Alternative	AlternativeConditionalType	0..*	An alternative set of resource constraints.
ResourceConstraint	ConditionalResourceConstraintType	0..*	A set of constraints on one resource.
	xsd:any	0..*	
operation	OperationListType	0..1	The condition applies only when processing the artifact associated with this operation.
	xsd:anyAttribute	0..*	

4.5.1.2 ConditionType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the condition.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the condition.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Alternative**: When a condition can be satisfied in multiple ways, two or more *Alternative* elements are defined.

  As a convenience for tooling that produces SDDs, it is also possible to define a single *Alternative*. This is semantically identical to directly defining *ResourceConstraints*.

  To meet a condition, at least one of the specified *Alternatives* must be satisfied.

  See the *AlternativeConditionalType* section for structure and additional usage details [4.5.2].

- **ResourceConstraint**: When a condition can be satisfied in only one way, constraints MAY be defined directly under *Condition* or in a single *Alternative* element.

  Constraints are defined using a sequence of *ResourceConstraints*. Every constraint in the sequence must be met for the condition to be met.

  See the *ConditionalResourceConstraintType* section for structure and additional usage details [4.5.3].

- **operation**: In a singleton atomic content element, a condition MAY be associated with application of one or more artifacts. The association is made by setting the *operation* attribute to the operations associated with those artifacts.

  Conditions defined for *CompositeInstallable* and for atomic content elements defined within a *CompositeInstallable* SHOULD NOT define *operation*. If the *operation* is defined for a *CompositeInstallable Condition*, it MUST be set to the operation defined in the *CompositeInstallable's operation* attribute. If *operation* is defined for an atomic content element's *Condition*, it MUST be set to the operation associated with the single artifact defined by the atomic content element.

  When *operation* is not specified, the condition applies to the processing of all artifacts.

  See the *OperationListType* section for *operation* enumerations and their meaning [4.3.6].

4.5.2 AlternativeConditionalType
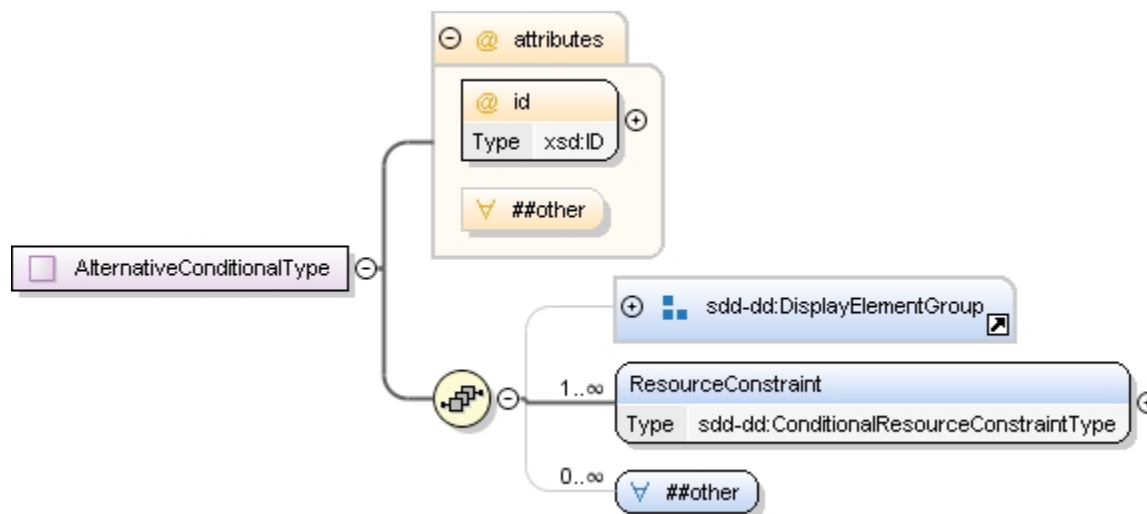
Figure 46: AlternativeConditionalType structure.

When a condition can be met in more than one way, alternative sets of conditional resource constraints can be defined. *AlternativeConditionalType* provides the type definition for these elements.

4.5.2.1 AlternativeConditionalType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the alternative.
Description	DisplayTextType	0..1	Description for the alternative.
ShortDescription	DisplayTextType	0..1	Short description of the alternative.
ResourceConstraint	ConditionalResourceConstraintType	1..*	A set of constraints on one resource.
	xsd:any	0..*	
id	xsd:IDREF	1	Identifier for the alternative that is unique within the deployment descriptor.
	xsd:anyAttribute	0..*	

4.5.2.2 AlternativeConditionalType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the alternative condition.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the alternative condition.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **ResourceConstraint**: All constraints defined in the individual *Alternative* MUST be met for the *Alternative* condition to evaluate to true.

  See the *ConditionalResourceConstraintType* section for structure and additional usage details [4.5.3].

- **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating log and trace messages.

4.5.3 ConditionalResourceConstraintType

1801
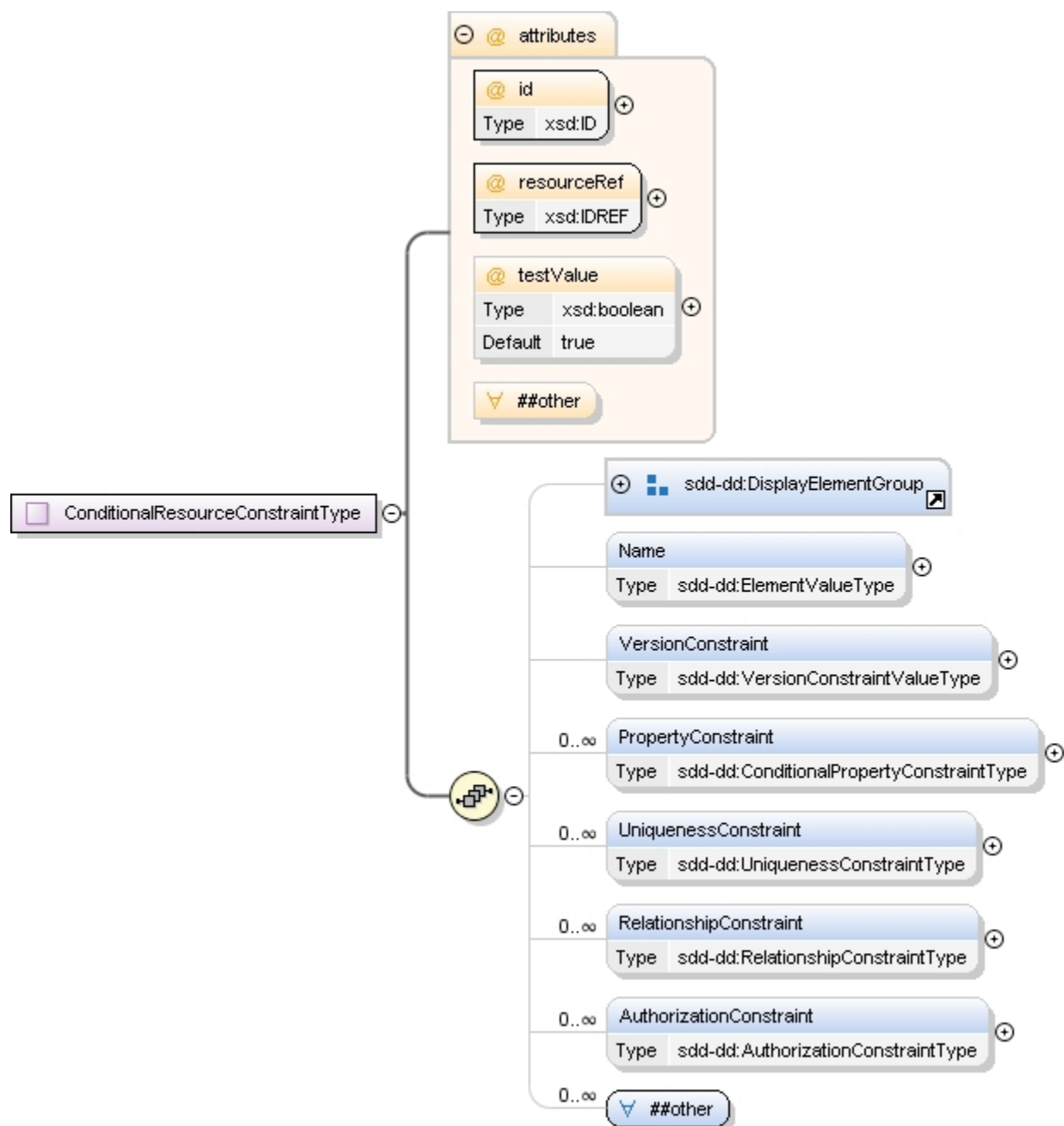
Figure 47: ConditionalResourceConstraintType structure.

1803 *ConditionalResourceConstraintType* provides the type definitions for *ResourceConstraint* elements used
1804 in conditions. These constraints do not represent requirements for deployment. Instead, they identify the
1805 resource characteristics associated with a condition. Version, property and the existence or absence of
1806 the resource can be specified with a resource constraint used in a condition.

1807   For example, an SDD author might want to deploy a specific artifact to Windows XP operating
1808   systems but not to other Windows operating systems. To accomplish this, a *ResourceConstraint* that
1809   includes a *VersionConstraint* that identifies the OS version that matches Windows XP is included with
1810   the *InstallableUnit* for that specific artifact. At runtime, if the OS version matches, the condition is met
1811   and the artifact is deployed; otherwise, it is skipped.

4.5.3.1 ConditionalResourceConstraintType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the resource constraint.
Description	DisplayTextType	0..1	Description for the resource constraint.
ShortDescription	DisplayTextType	0..1	Short description of the resource constraint.
~~Name~~	~~VariableExpressionType~~	~~0..1~~	~~Name of the resource constraint.~~ [DEPRECATED in SDD v2.0]
VersionConstraint	VersionConstraintValueType	0..1	A resource version set.
PropertyConstraint	ConditionalPropertyConstraintType	0..*	A resource property name and required value.
UniquenessConstraint	UniquenessConstraintType	0..*	A required mapping of two resources in the topology to unique instances in the deployment environment.
RelationshipConstraint	RelationshipConstraintType	0..*	A required relationship between the resource identified in the resourceRef and another resource in the topology.
AuthorizationConstraint	AuthorizationConstraintType	0..*	A required authorization level for a resource.
	xsd:any	0..*	
id	xsd:ID	1	Identifier for the resource constraint that is unique within the deployment descriptor.
resourceRef	xsd:IDREF	1	The resource to which the conditions apply.
testValue	xsd:boolean	0..1	The result of evaluating the contained constraints, which will result in the ResourceConstraint being met. **default value="true"
	xsd:anyAttribute	0..*	

4.5.3.2 ConditionalResourceConstraintType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the resource constraint.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the resource constraint.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- ~~**Name**: The name of the resource identified by *resourceRef*. If the resource name is defined in topology it SHOULD NOT be defined here. If it is defined in both places, the one defined in the condition is used when evaluating the condition.~~

  ~~See the *VariableExpressionType* section for structure and additional usage details [4.6.1].~~

  [Starting with SDD v2.0, *Name* has been deprecated.]

- **VersionConstraint**: The actual version of the resource MUST be one of the set of versions defined here for the version condition to be considered met.

  See the *VersionConstraintValueType* section for structure and additional usage details [4.4.8].

- **PropertyConstraint**: The actual value of the property MUST match the value defined here for the condition to be considered met.

  See the *ConditionalPropertyConstraintType* section for structure and additional usage details [4.5.4].

- **UniquenessConstraint**: *UniquenessConstraint* elements are used in *ResourceConstraints* to indicate when two resources defined in topology MUST or MUST NOT resolve to the same resource instance during a particular deployment.

  See the *UniquenessConstraintType* section for structure and additional usage details [4.4.12].

- **RelationshipConstraint**: *RelationshipConstraint* elements are used in *ResourceConstraints* to indicate a constraint on a particular relationship between resources.

  See the *RelationshipConstraintType* section for structure and additional usage details [4.4.13].

- **AuthorizationConstraint**: *AuthorizationConstraint* elements are used in *ResourceConstraints* to indicate a required level of authorization required by a resource in order to deploy the content that is described by the SDD.

  See the *AuthorizationConstraintType* section for structure and additional usage details [4.4.14].

- **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating log and trace messages.

- **resourceRef**: The version and property constraints defined here all apply to the one resource specification in topology identified by this attribute.

  The value MUST reference the *id* of that resource element in *Topology*.

- **testValue**: When the result of evaluating all of the constraints defined in the *ResourceConstraint* matches the value of *testValue*, the *ResourceConstraint* is considered met.

  When no version or property constraints are defined, and *testValue* is "true", the constraint is met if the resource exists as defined in topology.

  When no version or property constraints are defined, and *testValue* is "false", the constraint is met if the resource, as defined in topology, does not exist.

4.5.4 ConditionalPropertyConstraintType
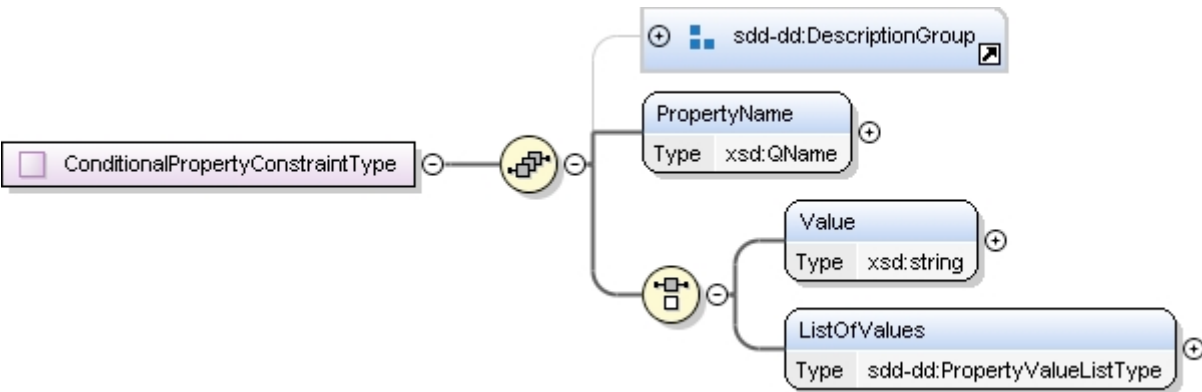
Figure 48: ConditionalPropertyConstraintType structure.

ConditionalPropertyConstraintType provides the type definition for a *PropertyConstraint* included within *Alternatives* specified in *Condition* elements. The *ConditionalPropertyConstraintType* is very similar to the *PropertyConstraintType*; the only difference is that the *Value* element defined in the *ConditionalPropertyConstraintType* is of type `xsd:string` which is less restrictive than the *Value* element defined in the *PropertyConstraintType* which is of *ElementValueType*.

4.5.4.1 ConditionalPropertyConstraintType Property Summary

Name	Type	*	Description

Description	DisplayTextType	0..1	A description of the property constraint. Required if ShortDescription is defined.
ShortDescription	DisplayTextType	0..1	A short description of the property constraint.
PropertyName	xsd:QName	1	Name of the constrained property.
Value	xsd:string	0..1	Required property value.
ListOfValues	PropertyValueListType	0..1	List of required property values.

4.5.4.2 ConditionalPropertyConstraintType Property Usage Notes

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the *PropertyConstraint* element.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DescriptionGroup* section for structure and additional usage details [4.14.1].
- **PropertyName**: The property name can be used to find the property value in the deployment environment. The name may be defined in a profile [5.3].
- **Value**: In a condition, the value used in a property constraint is a string rather than a variable expression.
- **ListOfValues**: A list of required values can be defined in place of a single required value.

  See the *PropertyValueListType* section for structure and additional usage details [4.4.6].

4.6 Variables

Variables provide a means to associate user inputs, resource property values, fixed strings and values derived from these with input arguments for artifacts and with constraints on resources. Three types of variables can be defined in an SDD. *Parameter* is a variable whose value is expected to be received as input to the deployment process. *ResourceProperty* is a variable whose value is set from the property of a specific instance of a resource during a particular solution deployment. *DerivedVariable* is exactly what its name indicates; it is a variable that is derived from values defined elsewhere in the descriptor or in the environment.

A variable is considered defined if it has a value provided, even if that value is the empty string. A variable expression is considered valid if it contains no variable references, or if all contained variable references are defined.

Specifically, a *ResourceProperty* variable is undefined when the resource does not participate in the particular deployment or when the specified property has no value. A *Parameter* variable is undefined when it has no default value and has no value provided by the deployer. A *DerivedVariable* that uses *ConditionalExpression* elements is undefined when none of its conditions evaluates to true, or the selected condition's value expression is not valid. A *DerivedVariable* that uses an unconditioned *Expression* is undefined when its value expression is undefined.

To avoid an undefined *Parameter* variable, default parameter values may be used. To avoid an undefined *ResourceProperty* variable, replace references to the *ResourceProperty* variable with references to a *DerivedVariable* defined to provide a default value in cases where the *ResourceProperty* is undefined. This *DerivedVariable* would define one expression, conditioned on the resource, that refers to the *ResourceProperty* variable and another, low priority, catch-all expression that defines the desired "default" value. Note that the default value in either of these cases MAY be an empty string, for example, "". An empty string acts just like any other defined variable value. When the provided value of a variable is an empty string, the variable reference in a variable expression is replaced by an empty string.

4.6.1 VariableExpressionType

Variable expressions allow the value of a variable to be used as all, or part of, the value of some other SDD element. A variable expression is a string that can include a reference to a variable. The string is

1902 evaluated by replacing all references to variables with the value of the variable. A variable reference is a
1903 variable id placed inside parentheses preceded by a dollar sign.

1904   For example, the variable expression "C:\Program Files\$(InstallDirectory)" resolves to "C:\Program
1905   Files\Acme Software Product" if the value of the variable with the id "InstallDirectory" has the value
1906   "Acme Software Product".

1907 The value of a variable that is replaced into a variable expression can itself have a variable reference.
1908 This reference is resolved before using the value. This nesting of variable expressions is unlimited. Any
1909 number of variable references can be used in a variable expression. If a variable expression string does
1910 not contain a variable reference, it is used as is.

4.6.2 ElementValueType

1911
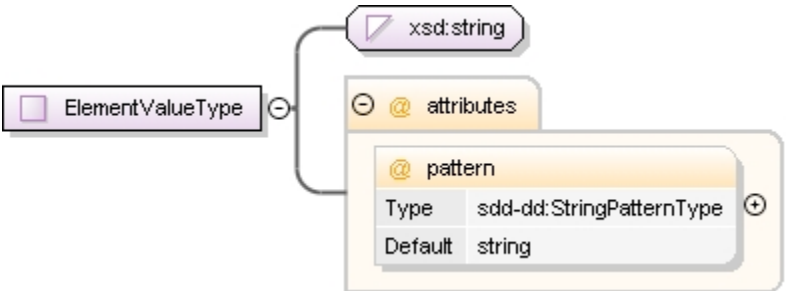
1912
Figure 49: ElementValueType structure.
1913

1914 *ElementValueType* provides a string value and a processing hint to the runtime necessary for proper
1915 handling the string. This provides the author with flexibility when specifying string values, so that literals
1916 and expressions are interpreted as the author intended. See **[SDDP]** and **[SDDEX]** for examples using
1917 the *ElementValueType*.

4.6.2.1 ElementValueType Property Summary

1918

Name	Type	*	Description
	[extends] xsd:string		See the xsd:string definition in **[XSD]**.
pattern	StringPatternType	0..1	The format of the specified value indicating how the string should be interpreted at runtime. **default value="string"

4.6.2.2 ElementValueType Property Usage Notes

1919

1920 See the `xsd:string` definition in **[XSD]** for inherited attributes and elements.

1921 ▪ **pattern**: The value of *pattern* MUST be one of the enumerations provided by *StringPatternType*.

1922   In some expressions of *ElementValueType*, wildcards logically do not make sense. A usage note is
1923   provided if an element does not support a *pattern* of wildcard.

1924   See the *StringPatternType* section for structure and additional usage details [4.6.3].

4.6.3 StringPatternType

1925

1926 *StringPatternType* provides an enumeration of the possible patterns for runtime processing of string
1927 values specified in elements of *ElementValueType*.

4.6.3.1 StringPatternType Property Usage Notes

1928

1929 Attributes of *StringPatternType* MUST be set to one of the following values:

1930 ▪ **string**: The value specified will be treated as a case insensitive string.

1931 ▪ **literal**: The value specified will be treated as a case sensitive string.

1932 ▪ **variableReference**: The value specified will be treated as a variable expression and string resolution
1933   will be performed.

1934 ▪ **wildcard**: The value specified will be processed as a type of regular expression, where "*" is the
1935   preferred wildcard character.

1936   For example, a *PropertyType* element might have a *Value* that can either match "Oracle 9iAS" or
1937   "Oracle 10". In order to indicate that either is an acceptible match, specify the expression value as
1938   "Oracle*", and set pattern="wildcard".

1939   In some expressions of *ElementValueType*, wildcards logically do not make sense. A usage note is
1940   provided if an element does not support a *pattern* of wildcard.

4.6.4 BaseVariableType

1942
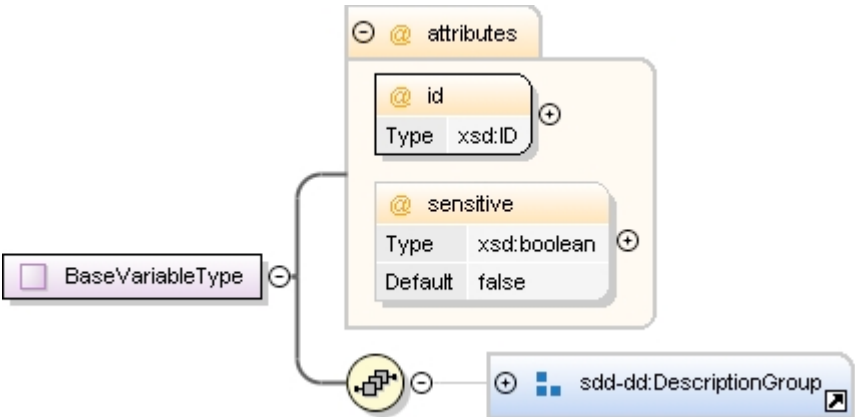
Figure 50: BaseVariableType structure.

1944 *BaseVariableType* is the base type of the *DerivedVariable* and *ResourceProperty* elements defined by
1945 *VariablesType* [4.6.5]. It provides the *id* attribute, which is used to reference the variable in a variable
1946 expression.

4.6.4.1 BaseVariableType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	Description of the variable.
ShortDescription	DisplayTextType	0..1	Short description of the variable.
id	xsd:ID	1	Identifier used for referencing the variable within the descriptor.
sensitive	xsd:boolean	0..1	A "true" value indicates the variable contains sensitive data. **default value="false"

4.6.4.2 BaseVariableType Property Usage Notes

1949 ▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable
1950   information. If used, they MUST provide a description of the variable.

1951   The *Description* element MUST be defined if the *ShortDescription* element is defined.

1952   See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

1953 ▪ **id**: Variables may be referenced in deployment descriptor elements of type *VariableExpression* within
1954   the scope of the variable. The scope of the variable includes the content element where defined and
1955   all nested content elements. *Variables* defined in the top level content element are also visible in

| 1956 | *Topology*. The *Variable* is referenced by placing the variable *id* within parentheses preceded by a |
| 1957 | dollar sign. |

| 1958 | For example, a variable with *id* value "InstallLocation" is referenced with the string |
| 1959 | "$(InstallLocation)". |

| 1960 | The *id* attribute may be useful to software that processes the SDD, for example, for use in creating |
| 1961 | log and trace messages. |

1962 ▪ **sensitive**: The *sensitive* attribute provides an indication of whether the data within a variable is likely
1963 to be considered sensitive. User name and password are examples of data that may be considered
1964 sensitive.

1965 For example, *sensitive* data typically would not be displayed in a user interface, written to a log
1966 file, stored without protection, or in any way made visible except to authorized users.

1967 The default value is "false".

4.6.5 VariablesType

1969

Figure 51: VariablesType structure.

1971 There are three types of variables that can be defined in a content element: input parameter variables,
1972 variables that take the value of a resource property, and variables whose value is derived from a variable
1973 expression.

1974 A variable is in scope for a particular deployment when the content element that defines the variable is in
1975 scope for that deployment.

4.6.5.1 VariablesType Property Summary

Name	Type	*	Description
Parameters	ParametersType	0..*	A list of variables whose values can be supplied as input to the deployment process.
ResourceProperty	ResourcePropertyType	0..*	A variable whose value is set from the value of a resource property.
DerivedVariable	DerivedVariableType	0..*	A set of expressions with optional associated conditions. The DerivedVariable's value is determined by evaluating the conditions and then setting the variable value to the result of the top priority expression from the set of expressions whose conditions evaluate to true.
	xsd:any	0..*	

4.6.5.2 VariablesType Property Usage Notes

1978 ▪ **Parameters**: See the *ParametersType* section for structure and additional usage details [4.6.6].

1979 ▪ **ResourceProperty**: See the *ResourcePropertyType* section for structure and additional usage details
1980 [4.6.18].

1981 ▪ **DerivedVariable**: See the *DerivedVariableType* section for structure and additional usage details
1982 [4.6.19].

1983 ## 4.6.6 ParametersType
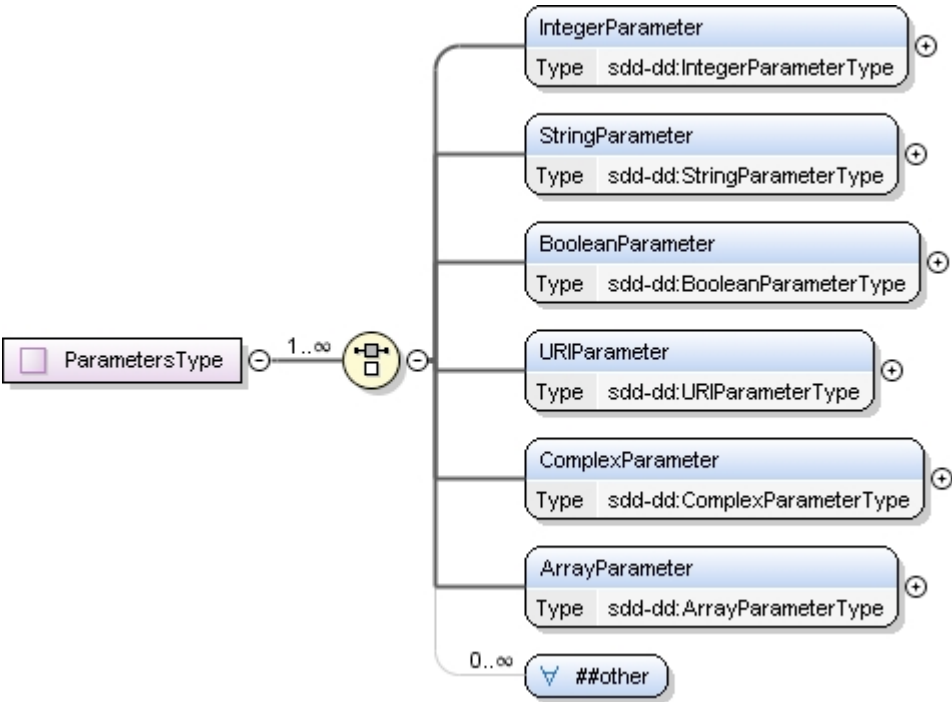
1984

1985 **Figure 52: ParametersType structure.**

1986 Parameters are variables whose value is expected to be received as input to the deployment process.
1987 The SDD author can specify multiple specific types of parameters, including validation rules for the values
1988 of the parameters.

1989 ### 4.6.6.1 ParametersType Property Summary

Name	Type	*	Description
IntegerParameter	IntegerParameterType	0..*	An integer input parameter.
StringParameter	StringParameterType	0..*	A string input parameter.
BooleanParameter	BooleanParameterType	0..*	A boolean input parameter.
URIParameter	URIParameterType	0..*	A Universal Resource Identifier input parameter.
ComplexParameter	ComplexParameterType	0..*	A set of input parameters of different base types.
ArrayParameter	ArrayParameterType	0..*	An array of input parameters of the same type.
	xsd:any	0..*	

1990 ### 4.6.6.2 ParametersType Property Usage Notes

1991 ▪ **IntegerParameter**: See the *IntegerParameterType* section for structure and additional usage details
1992 [4.6.8].

| 1993 | ▪ | **StringParameter**: See the *StringParameterType* section for structure and additional usage details [4.6.10]. |
| 1994 | | |

1993 ▪ **StringParameter**: See the *StringParameterType* section for structure and additional usage details
1994   [4.6.10].

1995 ▪ **BooleanParameter**: See the *BooleanParameterType* section for structure and additional usage
1996   details [4.6.12].

1997 ▪ **URIParameter**: See the *URIParameterType* section for structure and additional usage details
1998   [4.6.12.2].

1999 ▪ **ComplexParameter**: See the *ComplexParameterType* section for structure and additional usage
2000   details [4.6.13.2].

2001 ▪ **ArrayParameter**: See the *ArrayParameterType* section for structure and additional usage details
2002   [4.6.15].

2003 4.6.7 BaseParameterType

2004
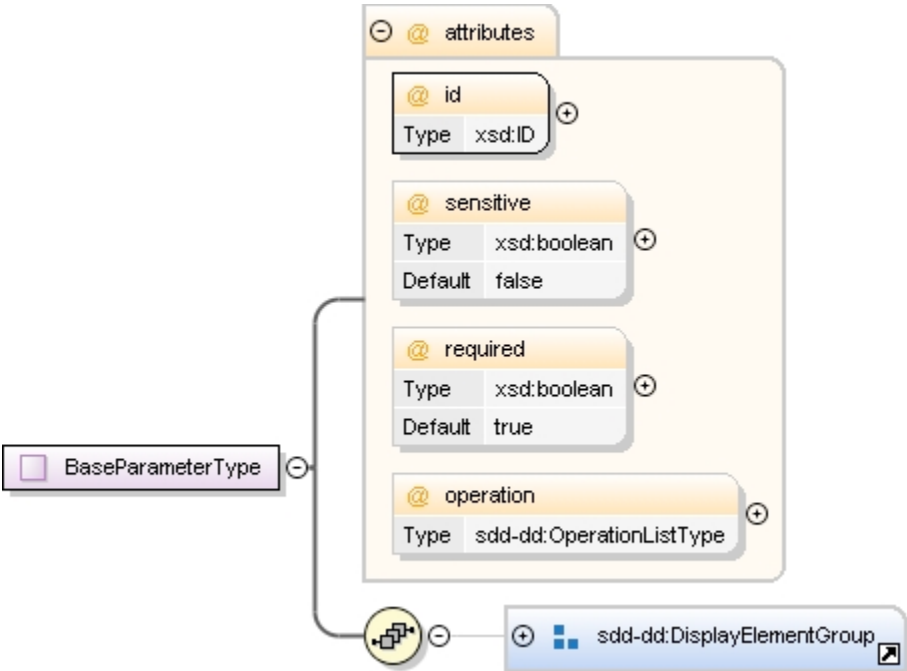
2005 **Figure 53: BaseParameterType structure.**

2006 *BaseParameterType* provides a default value, along with other attributes used by all parameter types. It
2007 also provides the *id* attribute, which is used to reference the parameter in variable expressions.

2008 4.6.7.1 BaseParameterType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the parameter.
Description	DisplayTextType	0..1	Description of the parameter.
ShortDescription	DisplayTextType	0..1	Short description of the parameter.
id	xsd:ID	1	Identifier used for referencing the variable within the descriptor.
sensitive	xsd:boolean	0..1	A "true" value indicates the variable contains sensitive data. **default value="false"
required	xsd:boolean	0..1	A "true" value indicates that a value for the parameter must be provided. **default value="true"

operation	OperationListType	0..1	The parameter is used when the specified operation(s) is (are) performed.

4.6.7.2 BaseParameterType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the parameter.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the parameter.

  These elements may be used to assist the deployer in understanding the purpose and expected values for the parameters.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **id**: Parameters may be referenced in *DeploymentDescriptor* elements of type *VariableExpression* or *ElementValueType* within the scope of the parameter variable. The scope of the variable includes the content element where the variable is defined and all nested content elements. Variables defined in the top level content element are also visible in *Topology*. The *Variable* is referenced by placing the variable *id* within parentheses preceded by a dollar sign.

  For example, a variable with *id* value "InstallLocation" is referenced with the string "$(InstallLocation)".

  The *id* attribute may be useful to software that processes the SDD, for example, for use in creating log and trace messages.

- **sensitive**: The *sensitive* attribute provides an indication of whether the data within a variable is likely to be considered sensitive. User name and password are examples of data that may be considered sensitive.

  For example, *sensitive* data typically would not be displayed in a user interface, written to a log file, stored without protection, or in any way made visible except to authorized users.

- **required**: A "true" value for *required* indicates that a value for the parameter must be provided when the parameter is in scope for a particular deployment.

  In cases where the parameter should be ignored when the value expression is not valid for a particular deployment, set required to "false".

  A "false" value for the *required* attribute has no effect when a *DefaultValue* is provided.

- **operation**: This attribute enables unique parameters to be defined per operation. Note that the use of a parameter for a particular operation is determined by a reference to the parameter in a variable expression or artifact argument used when performing that operation. The operation(s) associated with a parameter's use can be determined by examining its use in the SDD. The *operation* attribute provides a quick way to know which operation(s) will use the parameter without having to examine the use of the parameter.

  All parameters defined within a *CompositeInstallable* are associated with the single operation supported by the *CompositeInstallable*. The *operation* attribute SHOULD NOT be set in this situation.

  See the *OperationListType* section for *operation* enumerations and their meaning [4.3.6].

4.6.8 IntegerParameterType
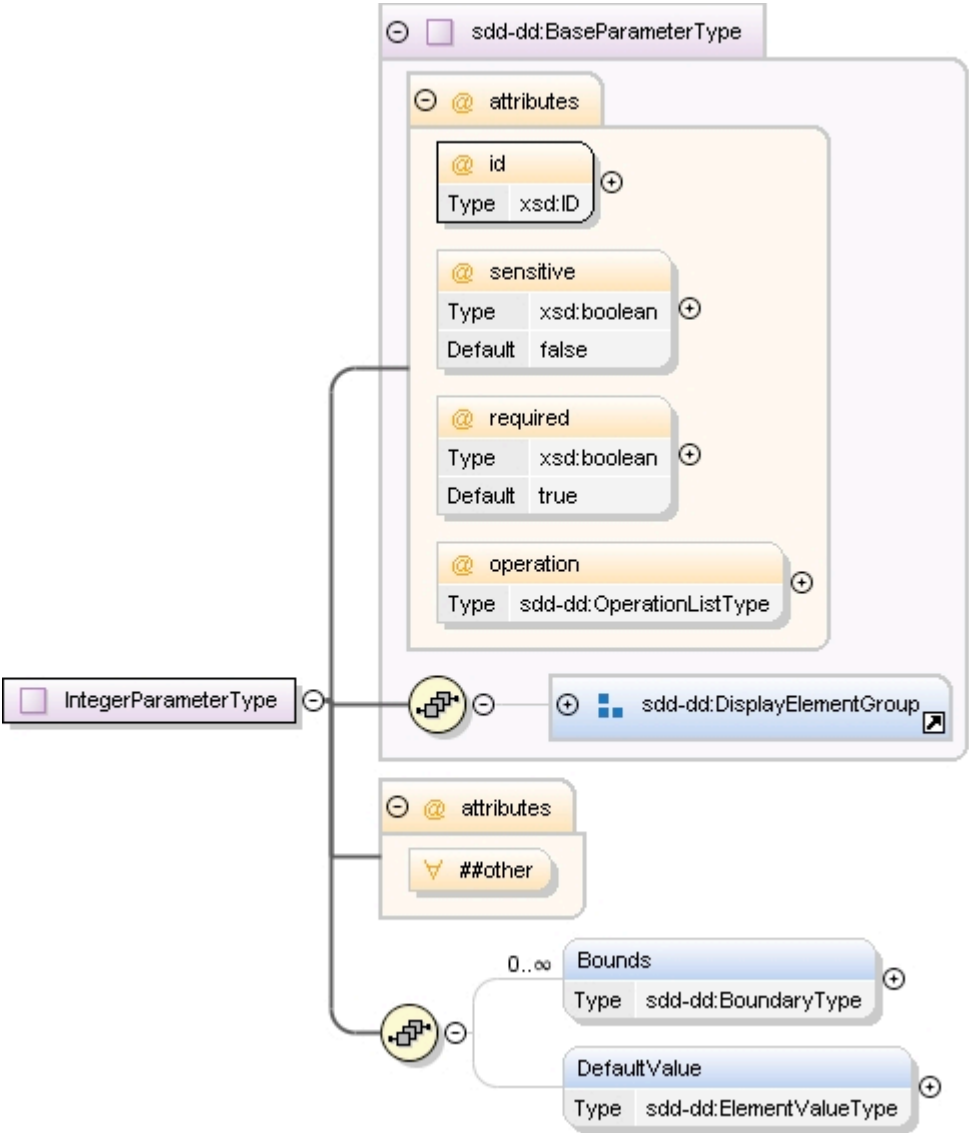
Figure 54: IntegerParameterType structure.

IntegerParameterType defines upper and lower bounds that can be used to validate the input received for that parameter.

4.6.8.1 IntegerParameterType Property Summary

Name	Type	*	Description
	[extends] BaseParameterType		See the BaseParameterType section for additional properties [4.6.7].
Bounds	BoundaryType	0..*	Specifies the boundaries for the value of the parameter.
DefaultValue	ElementValueType	0..1	Default value for the parameter.
	xsd:anyAttribute	0..*	

4.6.8.2 IntegerParameterType Property Usage Notes

2053

2054 See the *BaseParameterType* section for details about the inherited attributes and elements [4.6.7].

2055 ▪ **Bounds**: If there are restrictions on the range of values that are valid for a parameter, those
2056 restrictions MUST be specified in *Bounds*.

2057 See the *BoundaryType* section for structure and additional usage details [4.6.9].

2058 ▪ **DefaultValue**: The *DefaultValue* is used if no other value is provided as input to the deployment
2059 process.

2060 The value is interpreted based on the type of the defining parameter.

2061 For example, the *DefaultValue* for a *BooleanParameter* must be either "true" or "false"; the
2062 D*efaultValue* for a *StringParameter* must be a string; etc.

2063 A *pattern* of wildcard is not supported and MUST NOT be used with the *DefaultValue* element.

2064 See the *ElementValueType* section for structure and additional usage details [4.6.2].

4.6.9 BoundaryType

2065

2066
2067 **Figure 55: BoundaryType structure.**

2068 *BoundaryType* defines upper and lower bounds that can be used to validate the input received for that
2069 parameter.

4.6.9.1 BoundaryType Property Summary

2070

Name	Type	*	Description
LowerBound	ElementValueType	0..1	Lowest valid value for the parameter.
UpperBound	ElementValueType	0..1	Highest valid value for the parameter.

4.6.9.2 BoundaryType Property Usage Notes

2071

2072 ▪ **LowerBound**: This variable expression MUST resolve to an integer.

2073 If no *LowerBound* is specified, no integer value is too low.

2074 A *LowerBound* of "0" restricts the integer parameter to positive integer values.

2075 A *pattern* of wildcard is not supported and MUST NOT be used with the *LowerBound* element.

2076 See the *ElementValueType* section for structure and additional usage details [4.6.2].

2077 ▪ **UpperBound**: This variable expression MUST resolve to an integer.

2078 If no *UpperBound* is specified, no integer value is too high.

2079 A *pattern* of wildcard is not supported and MUST NOT be used with the *UpperBound* element.

2080 See the *ElementValueType* section for structure and additional usage details [4.6.2].

4.6.10 StringParameterType

2082
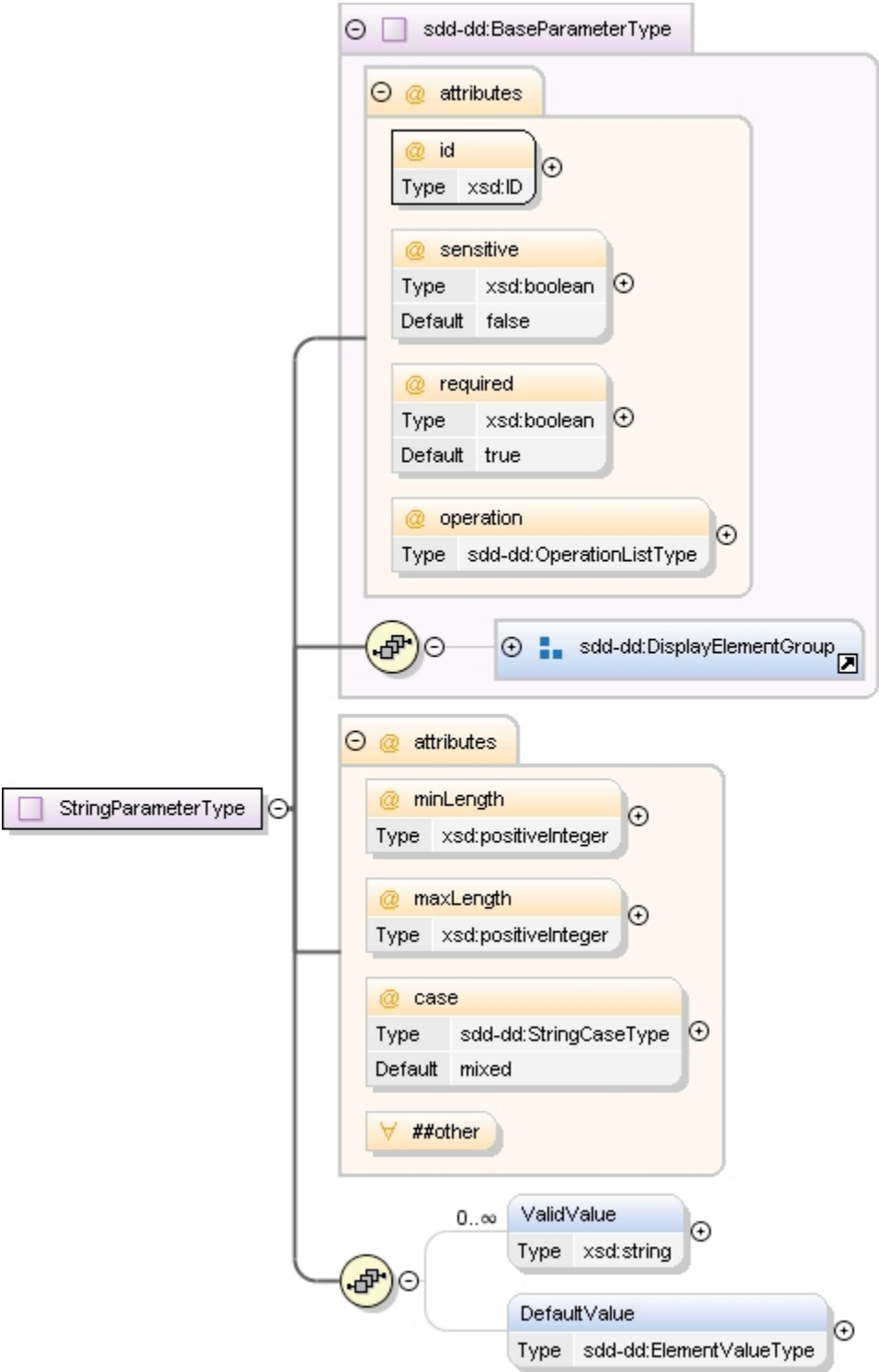
Figure 56: StringParameterType structure.

2084 *StringParameterType* supports definition of minimum and maximum lengths that can be used to validate
2085 the input received for the string parameter. It also supports definition of a list of valid input values.

4.6.10.1 StringParameterType Property Summary

Name	Type	*	Description
	[extends] BaseParameterType		See the BaseParameterType section for additional properties [4.6.7].
ValidValue	xsd:string	0..*	A string representing one valid value for the parameter.
DefaultValue	ElementValueType	0..1	Default value for the parameter.
minLength	xsd:positiveInteger	0..1	Minimum length of the parameter value.
maxLength	xsd:positiveInteger	0..1	Maximum length of the parameter value.
case	StringCaseType	0..1	The case of the string–"upper", "lower" or "mixed". **default value="mixed"
	xsd:anyAttribute	0..*	

4.6.10.2 StringParameterType Property Usage Notes

See the *BaseParameterType* section for details about the inherited attributes and elements [4.6.7].

- **ValidValue**: Any number of valid values for the parameter can be listed using *ValidValue* elements.

  When both *DefaultValue* and one or more *ValidValues* are specified, *DefaultValue* MUST match one of the *ValidValues*.

  ValidValues should be in the correct case as identified in the *case* attribute.

- **DefaultValue**: The *DefaultValue* is used if no other value is provided as input to the deployment process.

  The value is interpreted based on the type of the defining parameter.

  For example, the *DefaultValue* for a *BooleanParameter* must be either "true" or "false"; the D*efaultValue* for a *StringParameter* must be a string; etc.

  A *pattern* of wildcard is not supported and MUST NOT be used with the *DefaultValue* element.

  See the *ElementValueType* section for structure and additional usage details [4.6.2].

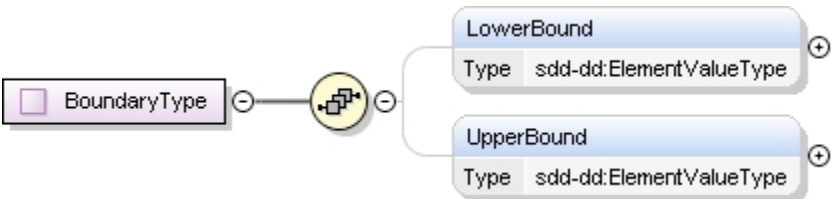- **minLength**: When no minimum length is specified, no string is too short, including an empty string.

- **maxLength**: When no maximum length is specified, no string is too long.

- **case**: Used when the case of the string is restricted. Defaults to *mixed* if not defined.

  See the *StringCaseType* section for enumeration values and their meaning [4.6.11].

4.6.11 StringCaseType

StringCaseType defines the enumeration values for specifying case restrictions on a string parameter.

4.6.11.1 StringCaseType Property Usage Notes

- **lower**: The string MUST be lower case.
- **upper**: The string MUST be upper case.
- **mixed**: The string SHOULD be mixed case.

4.6.12 BooleanParameterType
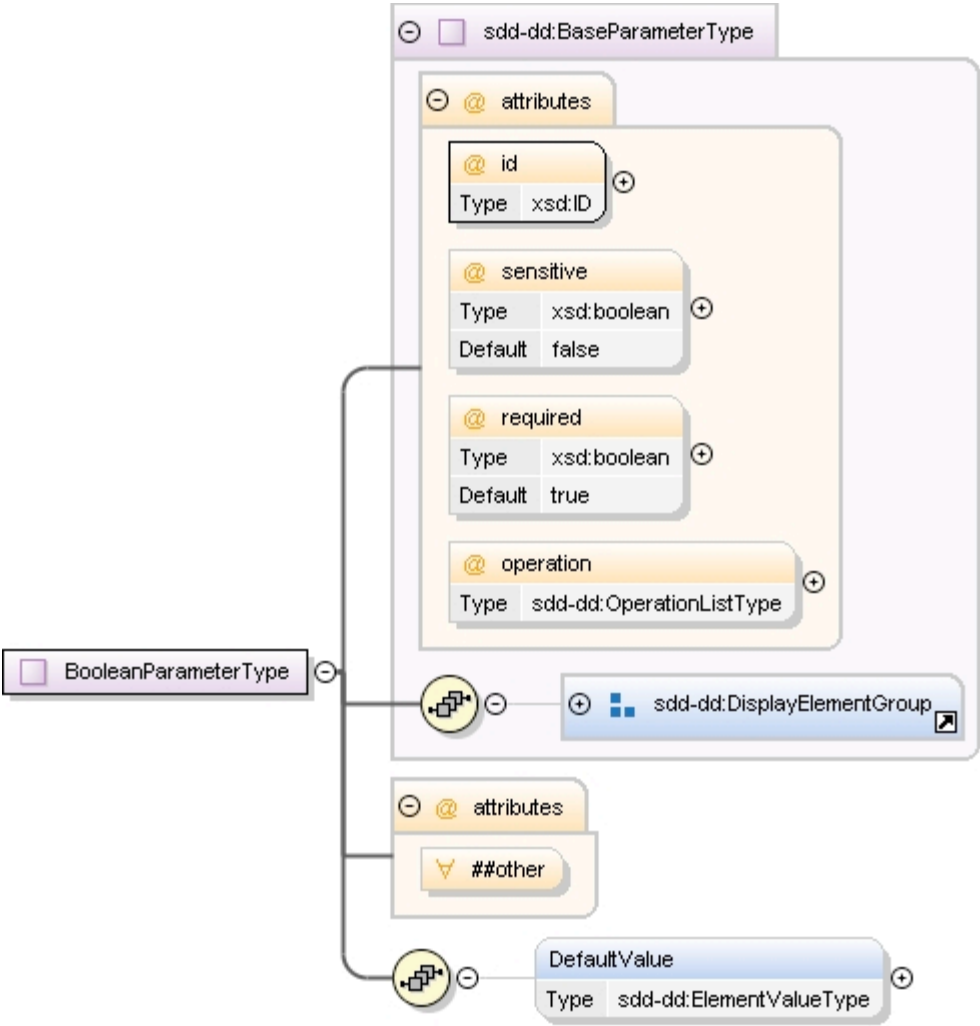
Figure 57: BooleanParameterType structure.

When the *DefaultValue* element is defined for a boolean parameter, its value MUST be either "true" or "false".

4.6.12.1 BooleanParameterType Property Summary

Name	Type	*	Description
	[extends] BaseParameterType		See the BaseParameterType section for additional properties [4.6.7].
DefaultValue	ElementValueType	0..1	Default value for the parameter.
	xsd:anyAttribute	0..*	

4.6.12.2 BooleanParameterType Property Usage Notes

See the *BaseParameterType* section for details about the inherited attributes and elements [4.6.7].

- **DefaultValue**: The *DefaultValue* is used if no other value is provided as input to the deployment process.

   The value is interpreted based on the type of the defining parameter.

2121		For example, the *DefaultValue* for a *BooleanParameter* must be either "true" or "false"; the *DefaultValue* for a *StringParameter* must be a string; etc.
2123		A *pattern* of wildcard is not supported and MUST NOT be used with the *DefaultValue* element.
2124		See the *ElementValueType* section for structure and additional usage details [4.6.2].
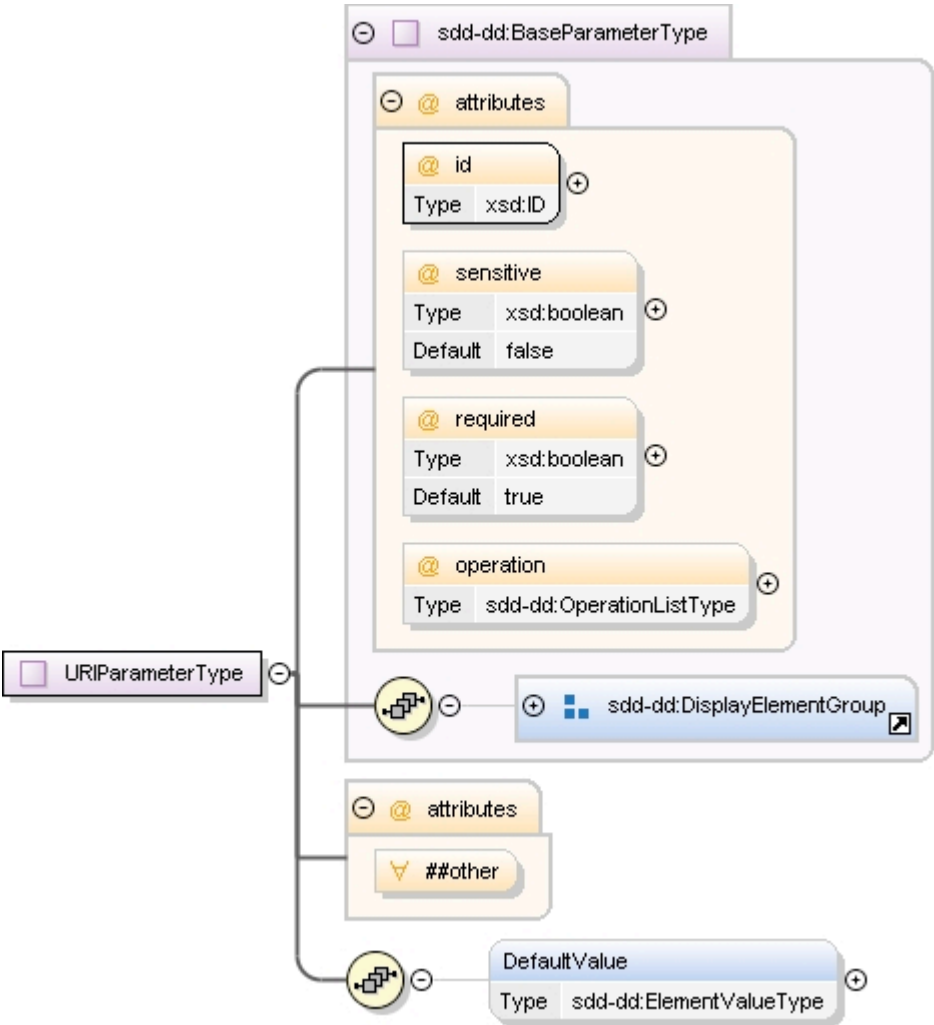
4.6.13 URIParameterType

Figure 58: URIParameterType structure.

When the *DefaultValue* element is specified for a URI parameter, its value MUST be a valid Uniform Resource Identifier.

4.6.13.1 URIParameterType Property Summary

Name	Type	*	Description
	[extends] BaseParameterType		See the BaseParameterType section for additional properties [4.6.7].
DefaultValue	ElementValueType	0..1	Default value for the parameter.
	xsd:anyAttribute	0..*	

4.6.13.2 URIParameterType Property Usage Notes

2131

2132 See the *BaseParameterType* section for details about the inherited attributes and elements [4.6.7].

2133 ▪ **DefaultValue**: The *DefaultValue* is used if no other value is provided as input to the deployment
2134 process.

2135 The value is interpreted based on the type of the defining parameter.

2136 For example, the *DefaultValue* for a *BooleanParameter* must be either "true" or "false"; the
2137 D*efaultValue* for a *StringParameter* must be a string; etc.

2138 A *pattern* of wildcard is not supported and MUST NOT be used with the *DefaultValue* element.

2139 See the *ElementValueType* section for structure and additional usage details [4.6.2].

4.6.14 ComplexParameterType

2140

2141
2142 **Figure 59: ComplexParameterType structure.**

2143 *ComplexParameterType* is useful for specifying variables that contain a set of values of differing base
2144 types, specifically for inputs that are logically grouped together, such as a host name and port number.
2145 See **[SDDEX]** for an example that demonstrates how to use the *ComplexParameterType* element.

2146 4.6.14.1 ComplexParameterType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the complex parameter.
Description	DisplayTextType	0..1	Description of the complex parameter.
ShortDescription	DisplayTextType	0..1	Short description of the complex parameter.
IntegerParameter	IntegerParameterType	0..*	An input array element of integer type.
StringParameter	StringParameterType	0..*	An input array element of string type.
BooleanParameter	BooleanParameterType	0..*	An input array element of boolean type.
URIParameter	URIParameterType	0..*	An input array element that is a Universal Resource Identifier.
	xsd:any	0..*	
id	xsd:ID	1	Identifier used for referencing the complex parameter within the descriptor.
required	xsd:boolean	0..1	A "true" value indicates that a value for the complex parameter must be provided. **default value="true"
operation	OperationListType	0..1	The complex parameter is used when the specified operation(s) is (are) performed.
length	xsd:positiveInteger	1	The number of elements in the input array.
	xsd:anyAttribute	0..*	

2147 4.6.14.2 ComplexParameterType Property Usage Notes

2148 ▪ **DisplayName**: This element MAY be used to provide human-understandable information. If used, it
2149 MUST provide a label for the parameter.

2150 See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2151 ▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable
2152 information. If used, they MUST provide a description of the parameter.

2153 These elements may be used to assist the deployer in understanding the purpose and expected
2154 values for the parameters.

2155 The *Description* element MUST be defined if the *ShortDescription* element is defined.

2156 See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2157 ▪ **IntegerParameter, StringParameter, BooleanParameter, URIParameter**: These define the type of
2158 the elements in the input array as Integer, String, Boolean, or URI respectively.

2159 ▪ **id**: Parameters may be referenced in *DeploymentDescriptor* elements of type *VariableExpression* or
2160 *ElementValueType* within the scope of the parameter variable. The scope of the variable includes the
2161 content element where the variable is defined and all nested content elements. Variables defined in
2162 the top level content element are also visible in *Topology*. The *Variable* is referenced by placing the
2163 variable *id* within parentheses preceded by a dollar sign.

| 2164 | For example, a variable with *id* value "InstallLocation" is referenced with the string |
| 2165 | "$(InstallLocation)". |

| 2166 | The *id* attribute may be useful to software that processes the SDD, for example, for use in creating |
| 2167 | log and trace messages. |

2168 ▪ **required**: A "true" value for *required* indicates that a value for the parameter must be provided when
2169 the parameter is in scope for a particular deployment.

2170 In cases where the parameter should be ignored when the value expression is not valid for a
2171 particular deployment, set required to "false".

2172 ▪ **operation**: This attribute enables unique parameters to be defined per operation. Note that the use of
2173 a parameter for a particular operation is determined by a reference to the parameter in a variable
2174 expression or artifact argument used when performing that operation. The operation(s) associated
2175 with a parameter's use can be determined by examining its use in the SDD. The *operation* attribute
2176 provides a quick way to know which operation(s) will use the parameter without having to examine
2177 the use of the parameter.

2178 All parameters defined within a *CompositeInstallable* are associated with the single operation
2179 supported by the *CompositeInstallable*. The *operation* attribute SHOULD NOT be set in this situation.

2180 See the *OperationListType* section for *operation* enumerations and their meaning [4.3.6].

2181 ▪ **length**: The *length* attribute MUST be set to the number of elements in the input array.

2182   4.6.15 ArrayParameterType

2183
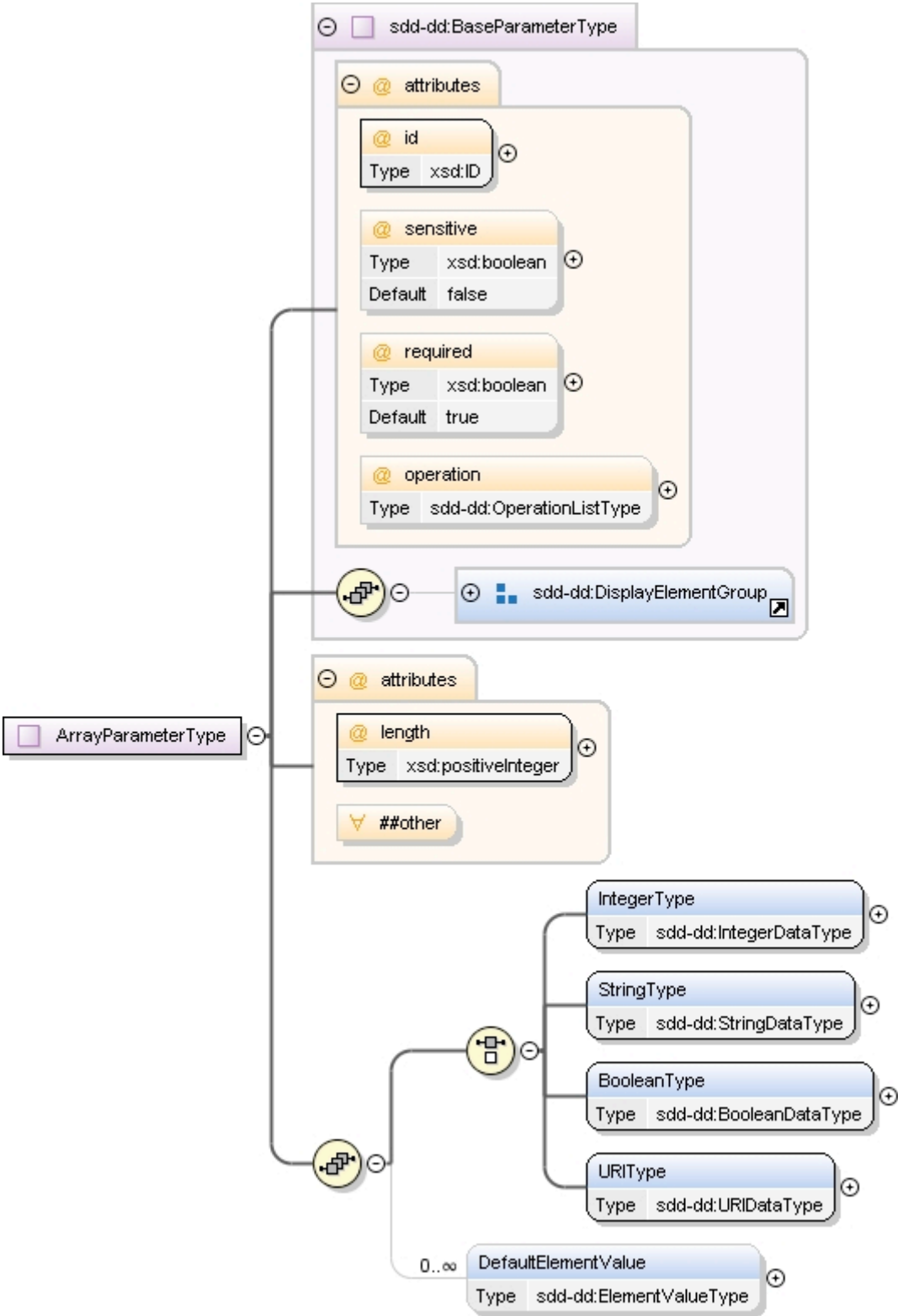
Figure 60: ArrayParameterType structure.

ArrayParameterType should be used for variables containing multiple inputs of the same base type.

2186   For example, a parameter might specify a set of files and their locations that need to have a string
2187   substitution applied, in this case, the *ArrayParameter* would have *URIType* specified and the
2188   *DefaultElementValue* elements would specify a URI for each file needing the substitution.

2189   See **[SDDEX]** for an example that demonstrates how to use the *ArrayParameterType* element.

4.6.15.1 ArrayParameterType Property Summary

Name	Type	*	Description
	[extends] BaseParameterType		See the BaseParameterType section for additional properties [4.6.7].
IntegerType	IntegerDataType	0..1	An array parameter of integer type.
StringType	StringDataType	0..1	An array parameter of string type.
BooleanType	BooleanDataType	0..1	An array parameter of boolean type.
URIType	URIDataType	0..1	An array parameter that is a Universal Resource Identifier.
DefaultElementValue	ElementValueType	0..*	The default value given to each item in the array.
length	xsd:positiveInteger	1	The number of elements in the input array.
	xsd:anyAttribute	0..*	

4.6.15.2 ArrayParameterType Property Usage Notes

2192   See the *BaseParameterType* section for details about the inherited attributes and elements [4.6.7].

2193   ▪   **IntegerType**: Used to specify a variable that contains an integer array.

2194   See the *IntegerDataType* section for structure and additional usage details [4.6.16].

2195   ▪   **StringType**: Used to specify a variable that contains a string array.

2196   See the *StringDataType* section for structure and additional usage details [4.6.17].

2197   ▪   **BooleanType**: Used to specify a variable that contains a boolean array.

2198   The *BooleanType* XML tag MUST be included for type identification of a boolean array parameter
2199   defined in the *DeploymentDescriptor*; however, the element does not include any attributes or
2200   elements.

2201   ▪   **URIType**: Used to specify a variable that contains an array of Universal Resource Identifier values.

2202   The *URIType* XML tag MUST be included for type identification of an array parameter containing
2203   URIs defined in the *DeploymentDescriptor*; however, the element does not include any attributes or
2204   elements.

2205   ▪   **DefaultElementValue**: The default value for each element of the array parameter.

2206   Separate *DefaultElementValue* entries MUST be defined even if the value of each array element is
2207   identical.

2208   See the *ElementValueType* section for enumeration values and their meaning [4.6.2].

2209   ▪   **length**: The *length* attribute MUST be set to the number of elements in the input array. The value of
2210   the *length* attribute MUST match the number of *DefaultElementValue* elements defined in the
2211   *ArrayParameter*.

2212   For example, if the parameter being defined is an integer array representing a list of default ports,
2213   and the length is set to "3", three separate *DefaultElementValue* elements MUST be defined, one
2214   for each port.

4.6.16 IntegerDataType
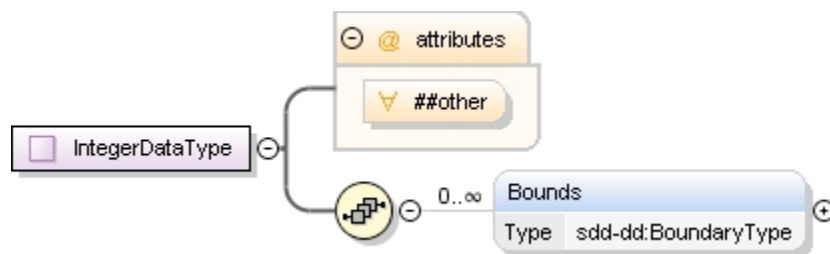
Figure 61: IntegerDataType structure.

IntegerDataType defines upper and lower bounds that can be used to validate the input received for the integer array parameter.

4.6.16.1 IntegerDataType Property Summary

Name	Type	*	Description
Bounds	BoundaryType	0..*	Specifies the boundaries for the values of the array parameter's entries.
	xsd:anyAttribute	0..*	

4.6.16.2 IntegerDataType Property Usage Notes

- **Bounds**: If there are restrictions on the range of values that are valid for the entries in an array parameter, those restrictions MUST be specified in *Bounds*.

  See the *BoundaryType* section for structure and additional usage details [4.6.9].

4.6.17 StringDataType
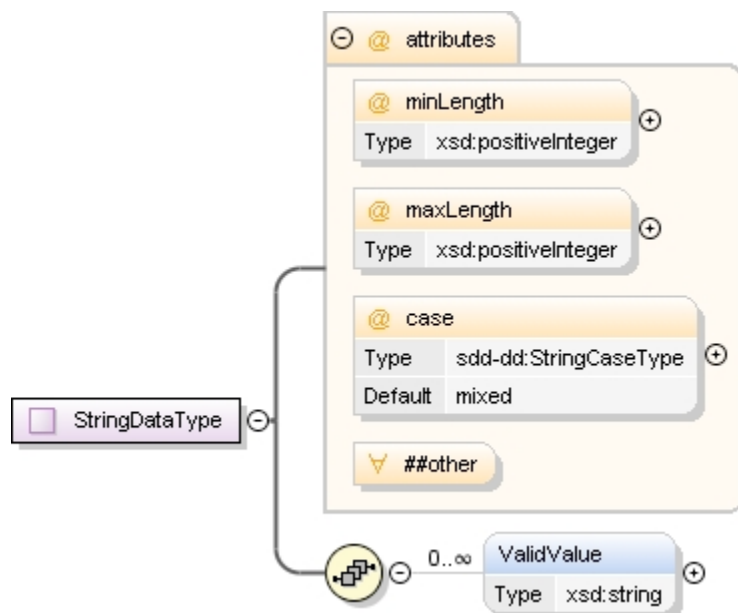
Figure 62: StringDataType structure.

StringDataType supports definition of minimum and maximum lengths that can be used to validate the input received for the string array parameter. It also supports definition of a list of valid input values.

4.6.17.1 StringDataType Property Summary

Name	Type	*	Description
ValidValue	xsd:string	0..*	A string representing one valid value for the parameter.
minLength	xsd:positiveInteger	0..1	Minimum length of the parameter value.
maxLength	xsd:positiveInteger	0..1	Maximum length of the parameter value.
case	StringCaseType	0..1	The case of the string–"upper", "lower" or "mixed". **default value="mixed"
	xsd:anyAttribute	0..*	

4.6.17.2 StringDataType Property Usage Notes

- **ValidValue**: Any number of valid values for the parameter can be listed using *ValidValue* elements.

  When both *DefaultElementValue* and one or more *ValidValues* are specified, *DefaultElementValue* MUST match one of the *ValidValues*.

  ValidValues should be in the correct case as identified in the *case* attribute.

- **minLength**: When no minimum length is specified, no string is too short, including an empty string.

- **maxLength**: When no maximum length is specified, no string is too long.

- **case**: Used when the case of the string is restricted. Defaults to *mixed* if not defined.

  See the *StringCaseType* section for enumeration values and their meaning [4.6.11].

4.6.18 ResourcePropertyType
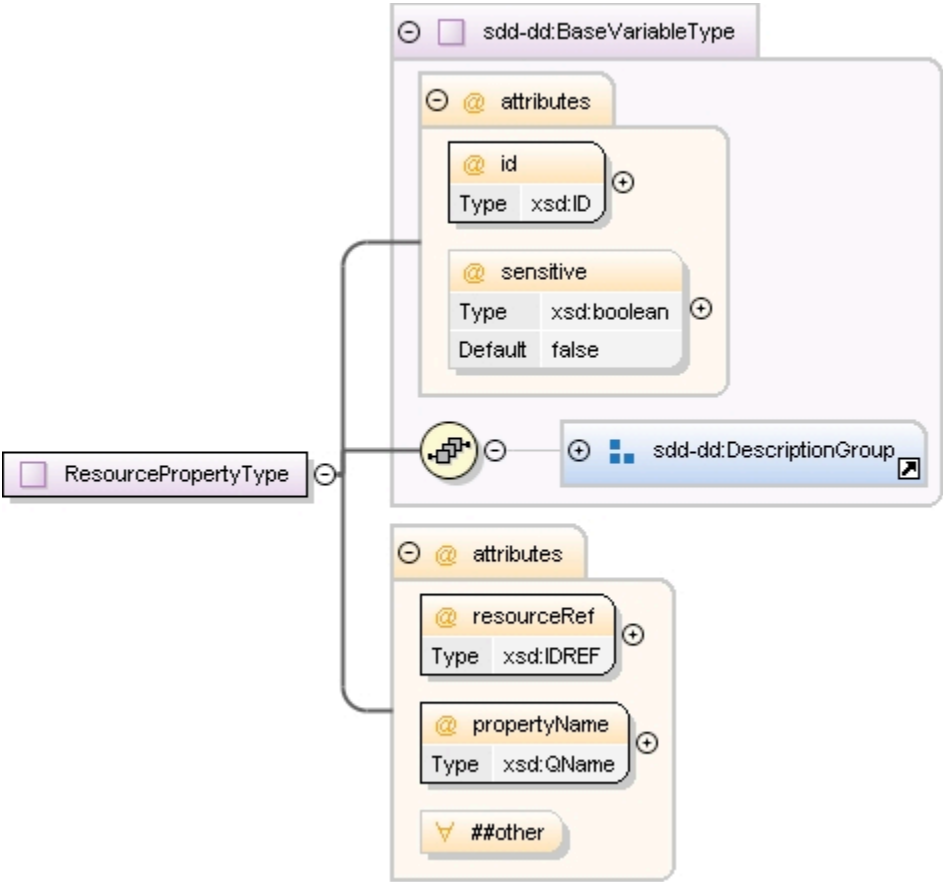
2241

Figure 63: ResourcePropertyType structure.

ResourcePropertyType provides the type definition for the *ResourceProperty* element of *VariablesType* [4.6.5]. *ResourceProperty* is a variable whose value is set from the property of a specific instance of a resource during a particular solution deployment. A SDD author MUST NOT include a *ResourceProperty* if the value of the property alone is not sufficient to resolve to a specific physical resource instance. All content elements can define *ResourceProperty* elements.

4.6.18.1 ResourcePropertyType Property Summary

Name	Type	*	Description
	[extends] BaseVariableType		See the BaseVariableType section for additional properties [4.6.4].
resourceRef	xsd:IDREF	1	The resource in Topology that owns the property.
propertyName	xsd:QName	1	Name of the property whose value provides the variable's values.
	xsd:anyAttribute	0..*	

4.6.18.2 ResourcePropertyType Property Usage Notes

See the *BaseVariableType* section for details about the inherited attributes and elements [4.6.4].

- **resourceRef**: The *resourceRef* attribute MUST identify the resource in *Topology* that owns the property and will provide the value for *ResourceProperty*.

- **propertyName**: The *propertyName* attribute identifies the name of the resource property whose value is to be used as the value of *ResourceProperty*.

4.6.19 DerivedVariableType
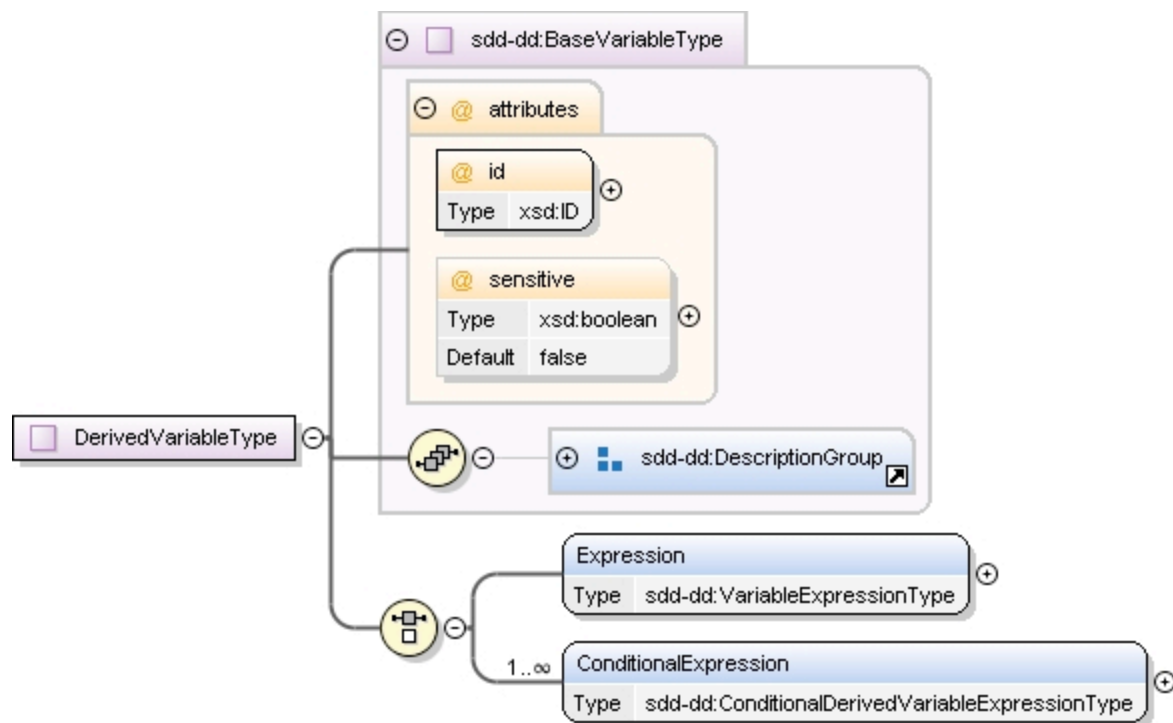
Figure 64: DerivedVariableType structure.

A *DerivedVariable* defines a series of expressions with optional conditions. The value of the variable is determined by evaluating the boolean conditions and then setting the variable to the result of the top priority expression from the set of expressions whose conditions evaluate to true. This restriction does not apply to variables of the same name in different descriptors. The SDD author MUST create *DerivedVariables* in a way that makes the selection of the expression unambiguous.

4.6.19.1 DerivedVariableType Property Summary

Name	Type	*	Description
	[extends] BaseVariableType		See the BaseVariableType section for additional properties [4.6.4].
Expression	VariableExpressionType	1	An expression whose results become the value of the variable.
ConditionalExpression	ConditionalDerivedVariableExpressionType	1..*	An expression and an associated condition.

4.6.19.2 DerivedVariableType Property Usage Notes

See the *BaseVariableType* section for details about the inherited attributes and elements [4.6.4].

▪ **Expression**: When the *DerivedVariable* is used to define one variable whose value is not conditional, the SDD author can include one variable expression defined in one *Expression* element.

See the *VariableExpressionType* section for structure and additional usage details [4.6.1].

▪ **ConditionalExpression**: When the variable will take one of a number of possible values depending on the characteristics of the resources that participate in the particular deployment, then one *ConditionalExpression* element is defined for each value-condition pair.

See the *ConditionalDerivedVariableExpressionType* section for structure and additional usage details [4.6.20].

4.6.20 ConditionalDerivedVariableExpressionType

2275
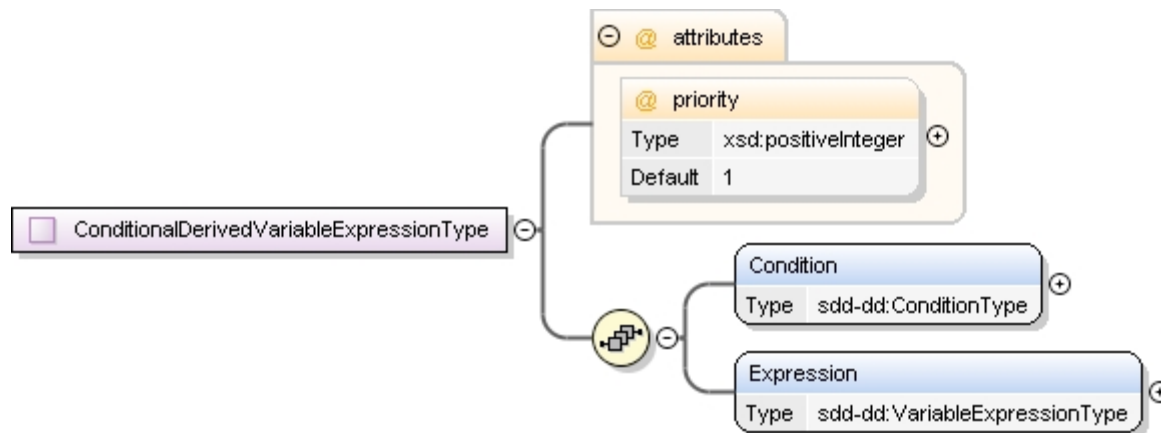
Figure 65: ConditionalDerivedVariableExpressionType structure.

ConditionalDerivedVariableExpressionType is the type of the *ConditionalExpression* elements in derived variables. These elements associate a condition with a variable expression.

4.6.20.1 ConditionalDerivedVariableExpressionType Property Summary

Name	Type	*	Description
Condition	ConditionType	1	A set of resource characteristics that are evaluated to determine if the associated expression is a candidate for determining the value of the derived variable.
Expression	VariableExpressionType	1	Evaluation of this expression produces a candidate value for the derived variable.
priority	xsd:positiveInteger	0..1	A priority used as a tie-breaker when multiple expressions are available to determine the value of the variable. **default value="1"

4.6.20.2 ConditionalDerivedVariableExpressionType Property Usage Notes

- **Condition**: Selection of conditioned expressions is based on the characteristics of one or more resources that participate in a particular solution deployment. These characteristics are defined in the *Condition* element.

  See the *ConditionType* section for structure and additional usage details [4.5.1].

- **Expression**: The *Expression* element contains the expressions that evaluate to a potential value of the *DerivedVariable*. Only one expression will be selected for use in a particular solution deployment.

  See the *VariableExpressionType* section for structure and additional usage details [4.6.1].

- **priority**: When multiple conditions evaluate to true for a particular deployment, the expression chosen is determined by the *priority* value. A higher priority is indicated by a lower value. "1" is the highest priority.

4.7 Requirements

A Requirement is an environmental necessity that a resource must have fulfilled in order for an artifact to be deployed successfully into that environment. *Requirements* are defined by content elements. A *Requirement* consists of resource constraints that the SDD author states MUST be met prior to successful deployment or use of the software described by the SDD package. Each *Requirement* definition lists one or more deployment lifecycle operations to which the *Requirement* applies. When the

2297 *Requirement* is specified in an atomic content element, the operation associates the *Requirement* with
2298 artifacts within the atomic content element. (See the *OperationType* section for the mapping between
2299 operations and artifacts [4.3.7]. Note that the *use* operation indicates that the *Requirement* is associated
2300 with running of the software after deployment and not with content element artifacts.) When the
2301 *Requirement* is specified in a *CompositeUnit* or *CompositeInstallable*, the *operation* value MUST either be
2302 *use* or be the same top level *operation* as defined in the *CompositeInstallable* element. When the
2303 Requirement is specified for a *ReferencedPackage*, the *operation* associates the *Requirement* with a top
2304 level *operation* within the referenced SDD.

2305 All *Requirements* specified for content elements that are in scope for a particular deployment MUST be
2306 met.

2307 When a *Requirement* can be satisfied in more than one way, *Alternatives* can be defined within a
2308 *Requirement*. A *Requirement* is considered met when any one of the *Alternatives* is satisfied.

4.7.1 RequirementsType
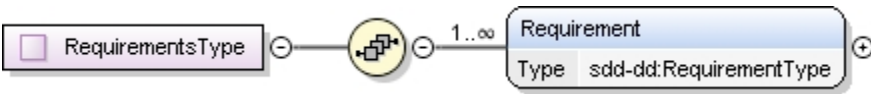
Figure 66: RequirementsType structure.

RequirementsType provides the type definition for *Requirements* in *InstallableUnit* and *LocalizationUnit*
elements. It defines a list of *Requirement* elements.

4.7.1.1 RequirementsType Property Summary

Name	Type	*	Description
Requirement	RequirementType	1..*	A requirement that must be met prior to processing the defining content element's artifacts.

4.7.1.2 RequirementsType Property Usage Notes

- **Requirement**: The *Requirements* element contains a sequence of *Requirement* elements. The *Requirement* elements define requirements that MUST be met prior to successful processing of the content element's artifacts.

  See the *RequirementType* section for structure and additional usage details [4.7.2].

4.7.2 RequirementType
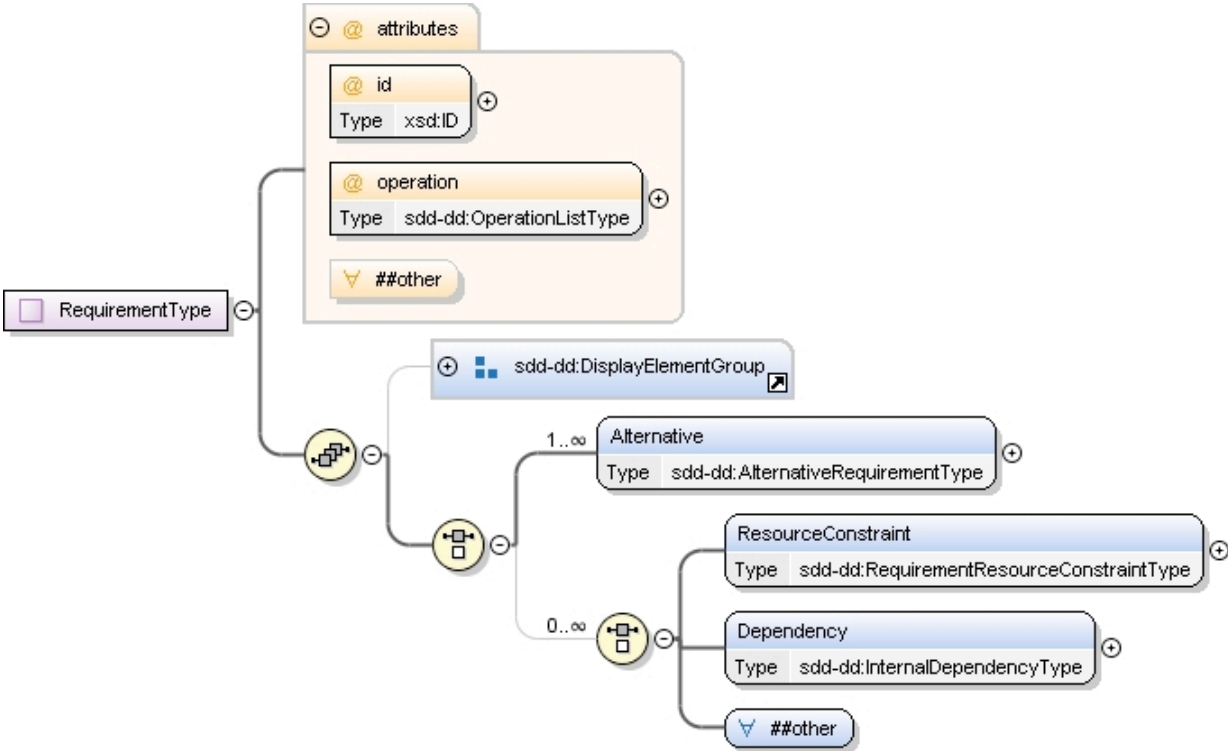
2321

Figure 67: RequirementType structure.

A *Requirement* either directly defines a single set of resource constraints that MUST be met or defines one or more alternative sets of resource constraints, only one of which MUST be met.

When multiple *Requirement* elements are declared for the same operation, all MUST be met prior to processing the associated artifact.

The association is made between a requirement and an artifact via the *operation* attribute.

4.7.2.1 RequirementType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the requirement.
Description	DisplayTextType	0..1	Description of the requirement.
ShortDescription	DisplayTextType	0..1	Short description of the requirement.
Alternative	AlternativeRequirementType	0..*	An alternative that can satisfy the requirement.
ResourceConstraint	RequirementResourceConstraintType	0..*	A set of constraints on one resource.
Dependency	InternalDependencyType	0..*	A dependency on another content element.
	xsd:any	0..*	
id	xsd:ID	1	Identifier for requirement scoped to the deployment descriptor.
operation	OperationListType	1	Requirement must be met before this operation is performed.

	xsd:anyAttribute	0..*	

4.7.2.2 RequirementType Property Usage Notes

2329

2330 ▪ **DisplayName**: This element MAY be used to provide human-understandable information. If used, it
2331 MUST provide a label for the requirement.

2332 See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2333 ▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable
2334 information. If used, they MUST provide a description of the requirement.

2335 The *Description* element MUST be defined if the *ShortDescription* element is defined.

2336 See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2337 ▪ **Alternative**: Alternative elements are used when a requirement can be satisfied in multiple ways.

2338 As a convenience for tooling that produces SDDs, it is also possible to define a single *Alternative*.
2339 This is semantically identical to directly defining *ResourceConstraints* under *Requirements*.

2340 To satisfy a requirement, at least one of the specified alternatives MUST be satisfied.

2341 See the *AlternativeRequirementType* section for structure and additional usage details [4.7.3].

2342 ▪ **ResourceConstraint**: When a requirement can be satisfied in only one way, constraints MAY be
2343 defined directly under *Requirement* or in a single *Alternative* element.

2344 Constraints are defined using a sequence of *ResourceConstraints*. Every constraint in the sequence
2345 MUST be met for the requirement to be met.

2346 See the *RequirementResourceConstraintType* section for structure and additional usage details
2347 [4.7.5].

2348 ▪ **Dependency**: When one content element must be processed before another for any reason, a *pre-*
2349 *req* type *Dependency* MUST be defined. Reasons for a pre-requisite dependency include the use of
2350 an output variable from one artifact as an argument to another; the deployment of a resource before it
2351 is configured; and the configuration of a resource before deployment of another resource that
2352 depends on it.

2353 See the *InternalDependencyType* section for structure and additional usage details [4.7.6].

2354 ▪ **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
2355 log and trace messages.

2356 The *id* attribute for a *Requirement* MUST be unique within a *DeploymentDescriptor* and MUST be
2357 unique across an aggregation of Deployment Descriptors.

2358 ▪ **operation**: A *Requirement* is associated with application of one or more operations by setting its
2359 *operation* attribute value to one of the enumerated values defined in *OperationListType* [4.3.6].

2360 If the *Requirement* is not a pre-requisite for application of an operation, but rather is required before
2361 the resulting resources are considered usable, then the value SHOULD be set to *use*. (Note that a
2362 completion action may also be required before a resulting resource is considered usable. See the
2363 *CompletionType* section [4.3.14].)

2364 The value of *operation* for a *Requirement* defined in an atomic content element MUST be set either to
2365 *use* or to an *operation* that is associated with an artifact element defined in the content element's
2366 *Artifacts*. The *operation* value(s) associate the *Requirement* with one or more artifact(s).

2367 When the *Requirement* is specified in a *CompositeUnit* or *CompositeInstallable*, the *operation* value
2368 MUST be set either to *use* or be the same top level *operation* as defined in the *CompositeInstallable*
2369 element.

2370 There is no default value for *operation*. The SDD author must define it explicitly.

2371 See the *OperationType* section for enumeration values and their meaning [4.3.7].

4.7.3 AlternativeRequirementType
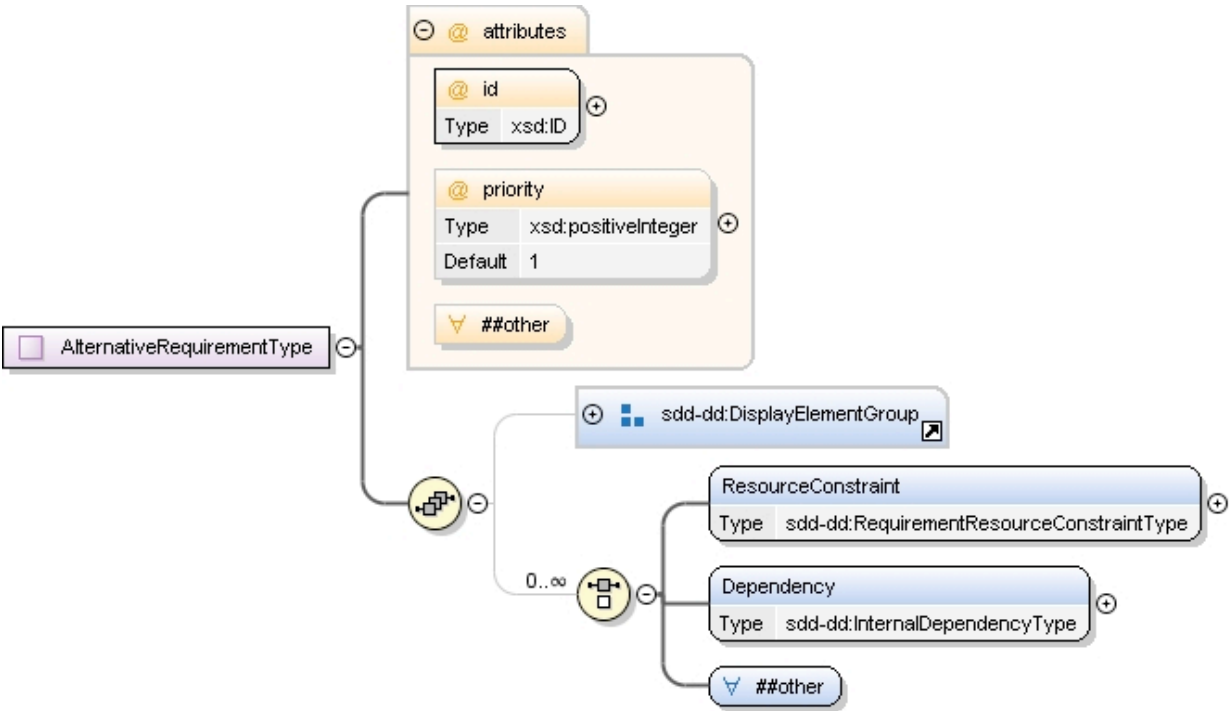
Figure 68: AlternativeRequirementType structure.

AlternativeRequirementType provides the type definition for *Alternative* elements used within requirements to define alternative sets of resource constraints that will satisfy the requirement.

4.7.3.1 AlternativeRequirementType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the alternative.
Description	DisplayTextType	0..1	Description of the alternative.
ShortDescription	DisplayTextType	0..1	Short description of the alternative.
ResourceConstraint	RequirementResourceConstraintType	1..*	A set of requirements on one resource.
Dependency	InternalDependencyType	0..*	A dependency on another content element.
	xsd:any	0..*	
id	xsd:ID	1	Identifier for the alternative scoped to the deployment descriptor.
priority	xsd:positiveInteger	0..1	Assists in determining alternative selected when multiple alternatives evaluate to true. **default value="1"
	xsd:anyAttribute	0..*	

4.7.3.2 AlternativeRequirementType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the alternative requirement.

2381   See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2382   ▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable
2383   information. If used, they MUST provide a description of the alternative requirement.

2384   The *Description* element MUST be defined if the *ShortDescription* element is defined.

2385   See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2386   ▪ **ResourceConstraint**: Every *ResourceConstraint* defined in a single *Alternative* MUST be met for the
2387   alternative requirement to be considered satisfied.

2388   See the *RequirementResourceConstraintType* section for structure and additional usage details
2389   [4.7.5].

2390   ▪ **Dependency**: When one content element must be processed before another for any reason, a *pre-*
2391   *req* type *Dependency* MUST be defined. Reasons for a pre-requisite dependency include the use of
2392   an output variable from one artifact as an argument to another; the deployment of a resource before it
2393   is configured; and the configuration of a resource before deployment of another resource that
2394   depends on it.

2395   See the *InternalDependencyType* section for structure and additional usage details [4.7.6].

2396   ▪ **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
2397   log and trace messages.

2398   ▪ **priority**: If there are multiple satisfied alternatives during a particular solution deployment, one of the
2399   alternatives must be selected. The *priority* attribute communicates the SDD author's prioritization of
2400   the alternatives. A lower number represents a higher priority with "1" representing the highest priority.
2401   Other inputs may also be used to select an alternative. The criteria for making this selection are
2402   outside of the scope of the SDD.

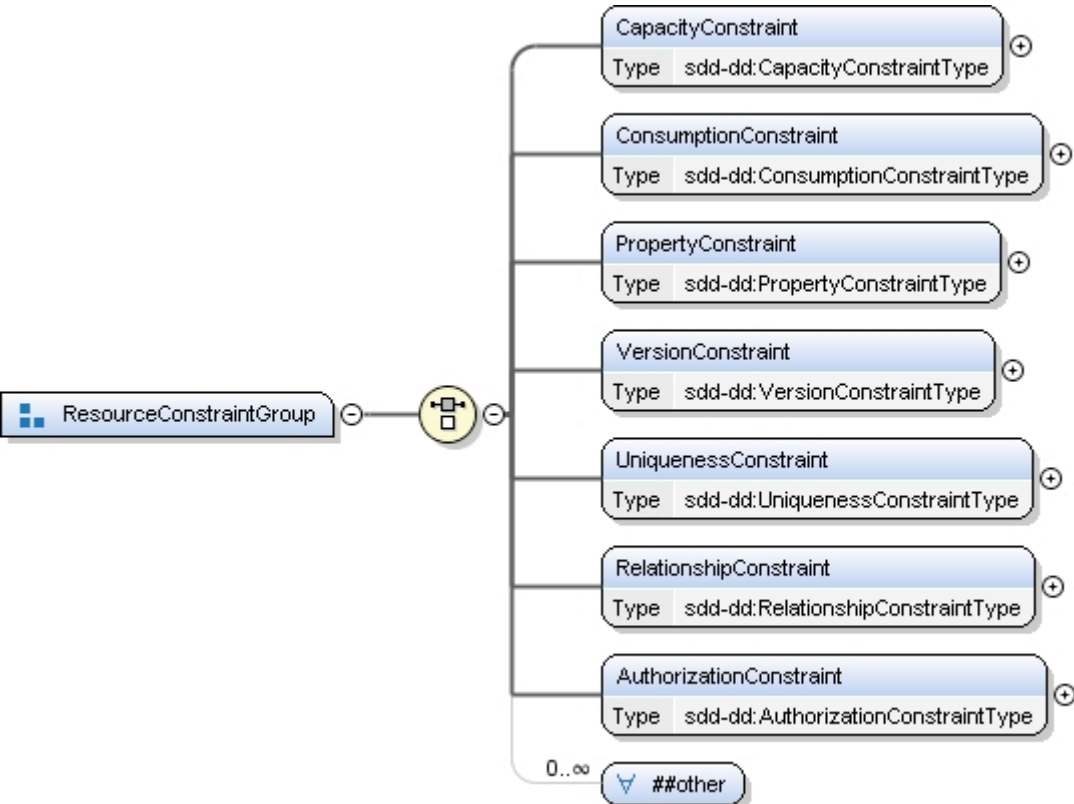## 2403   4.7.4 ResourceConstraintGroup

2404

2405 **Figure 69: ResourceConstraintGroup structure.**

2406 The elements of *ResourceConstraintGroup* are used when defining content element requirements on
2407 resources. The *ResourceConstraint* element is used to group one or more constraints on a single
2408 resource.

4.7.4.1 ResourceConstraintGroup Property Summary

2409

Name	Type	*	Description
CapacityConstraint	CapacityConstraintType	0..1	A bound on a quantifiable property of a resource.
ConsumptionConstraint	ConsumptionConstraintType	0..1	A required quantity of a property of a resource in any state.
PropertyConstraint	PropertyConstraintType	0..1	A required value or set of values of a property.
VersionConstraint	VersionConstraintType	0..1	A required value or set of values of a version property.
UniquenessConstraint	UniquenessConstraintType	0..1	A required mapping of two resources in the topology to unique instances in the deployment environment.
RelationshipConstraint	RelationshipConstraintType	0..1	A required relationship between the resource identified in the resourceRef and another resource in the topology.
AuthorizationConstraint	AuthorizationConstraintType	0..1	A required authorization level for a resource.
	xsd:any	0..*	

4.7.4.2 ResourceConstraintGroup Property Usage Notes

2410

2411 ▪ **CapacityConstraint**: *CapacityConstraint* elements are used in *ResourceConstraints* to express
2412 constraints on the available capacity of a particular property of a particular resource.

2413 A *CapacityConstraint* tests a numeric value representing a bound on a quantifiable property of a
2414 resource, such as processor speed. The test may be for a lower (minimum) or upper (maximum)
2415 bound. This constraint differs from a *ConsumptionConstraint* in that it is comparative, not cumulative.

2416 When multiple *CapacityConstraint* elements are defined by content elements participating in a
2417 particular solution deployment apply to the same property of the same resource, the most restrictive
2418 constraint applies.

2419 See the *CapacityConstraintType* section for structure and additional usage details [4.4.1].

2420 ▪ **ConsumptionConstraint**: *ConsumptionConstraint* elements are used in *ResourceConstraints* to
2421 express constraints on the quantity of a particular property of a specific resource that is available for
2422 consumption.

2423 A *ConsumptionConstraint* defines a required quantity of a consumable resource property. The
2424 *ConsumptionConstraint* is cumulative rather than comparative.

2425 An example of a consumable resource property is the disk space property of a file system
2426 resource.

2427 When multiple *ConsumptionConstraint* elements are defined for the same resource by content
2428 elements participating in a particular solution deployment, the sum of all the expressed consumption
2429 constraints must be met by the resource.

2430 See the *ConsumptionConstraintType* section for structure and additional usage details [4.4.3].

2431 ▪ **PropertyConstraint**: *PropertyConstraint* elements are used in *ResourceConstraints* to indicate that
2432 specific resource properties must have a specific value or set of values.

2433 See the *PropertyConstraintType* section for structure and additional usage details [4.4.5].

2434 ▪ **VersionConstraint**: *VersionConstraint* elements are used in *ResourceConstraints* to express a
2435 constraint on the version of a specific resource.

2436   A *VersionConstraint* defines a required resource version or a range of versions. It MAY include a
2437   certified version or range of versions representing a more restrictive set of versions whose use carries
2438   a higher degree of confidence.

2439   Version formats and comparison rules vary greatly. The SDD does not provide information on how to
2440   interpret version strings.

2441   See the *VersionConstraintType* section for structure and additional usage details [4.4.7].

2442   ▪ **UniquenessConstraint**: *UniquenessConstraint* elements are used in *ResourceConstraints* to
2443   indicate when two resources defined in topology MUST or MUST NOT resolve to the same resource
2444   instance during a particular deployment.

2445   See the *UniquenessConstraintType* section for structure and additional usage details [4.4.12].

2446   ▪ **RelationshipConstraint**: *RelationshipConstraint* elements are used in *ResourceConstraints* to
2447   indicate a constraint on a particular relationship between resources.

2448   See the *RelationshipConstraintType* section for structure and additional usage details [4.4.13].

2449   ▪ **AuthorizationConstraint**: *AuthorizationConstraint* elements are used in *ResourceConstraints* to
2450   indicate a required level of authorization required by a resource in order to deploy the content that is
2451   described by the SDD.

2452   See the *AuthorizationConstraintType* section for structure and additional usage details [4.4.14].

2453 4.7.5 RequirementResourceConstraintType

2454

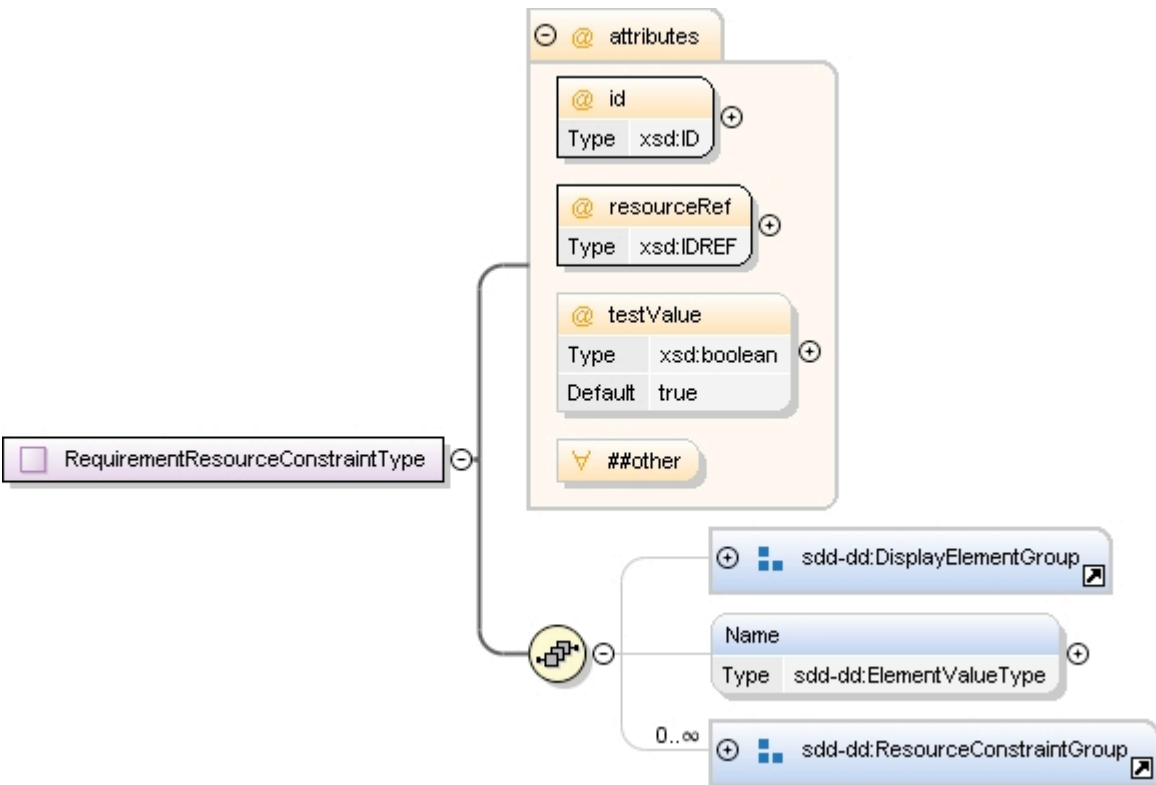2455 **Figure 70: RequirementResourceConstraintType structure.**

2456 *ResourceConstraintType* provides the Type section for the *ResourceConstraint* element in content
2457 element *Requirements*. A *ResourceConstraint* is a set of zero or more constraints on one resource.

2458 4.7.5.1 RequirementResourceConstraintType Property Summary

Name	Type	*	Description

DisplayName	DisplayTextType	0..1	Name for the resource constraint.
Description	DisplayTextType	0..1	Description of the resource constraint.
ShortDescription	DisplayTextType	0..1	Short description of the resource constraint.
~~Name~~	~~VariableExpressionType~~	~~0..1~~	~~The name of the resource.~~ [DEPRECATED in SDD v2.0]
CapacityConstraint	CapacityConstraintType	0..1	A capacity constraint that applies to the resource identified in resourceRef.
ConsumptionConstraint	ConsumptionConstraintType	0..1	A consumption constraint that applies to the resource identified in resourceRef.
PropertyConstraint	PropertyConstraintType	0..1	A property constraint that applies to the resource identified in resourceRef.
VersionConstraint	VersionConstraintType	0..1	A version constraint that applies to the resource identified in resourceRef.
UniquenessConstraint	UniquenessConstraintType	0..1	A required mapping of two resources in the topology to unique instances in the deployment environment.
RelationshipConstraint	RelationshipConstraintType	0..1	A required relationship between the resource identified in the resourceRef and another resource in the topology.
AuthorizationConstraint	AuthorizationConstraintType	0..1	A required authorization level to operate on the resource identified in resourceRef.
	xsd:any	0..*	
id	xsd:ID	1	Identifier for the ResourceConstraint scoped to the deployment descriptor.
resourceRef	xsd:IDREF	1	Reference to a resource specification in topology.
testValue	xsd:boolean	0..1	Indicates whether the ResourceConstraint must evaluate to true or to false. **default value="true".
	xsd:anyAttribute	0..*	

4.7.5.2 RequirementResourceConstraintType Property Usage Notes

2460 ▪ **DisplayName**: This element MAY be used to provide human-understandable information. If used, it
2461   MUST provide a label for the resource constraint.

2462   See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2463 ▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable
2464   information. If used, they MUST provide a description of the resource constraint.

2465   The *Description* element MUST be defined if the *ShortDescription* element is defined.

2466   See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2467 ▪ ~~**Name**: This name is used to identify the resource in the deployment environment. If the resource~~
2468   ~~identified by *resourceRef* does not have the name defined here, then the constraint is not met.~~

2469   ~~See the *VariableExpressionType* section for structure and additional usage details [4.6.1].~~

2470   [Starting with SDD v2.0, *Name* has been deprecated.]

- **CapacityConstraint, ConsumptionConstraint, PropertyConstraint, VersionConstraint, UniquenessConstraint, RelationshipConstraint, AuthorizationConstraint**: See the *ResourceConstraintGroup* section for structure and additional usage of the individual constraints [4.7.4].

- **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating log and trace messages.

- **resourceRef**: This is the resource to which the constraints apply.

  This reference MUST refer to the *id* of a resource in *Topology*.

- **testValue**: When the result of evaluating all of the constraints defined in the *ResourceConstraint* matches the value of *testValue*, the *ResourceConstraint* is considered met.

  When no constraints are defined, and *testValue* is "true", the constraint is met if the resource exists as defined in topology.

  When no constraints are defined, and *testValue* is "false", the constraint is met if the resource, as defined in topology, does not exist.

4.7.6 InternalDependencyType
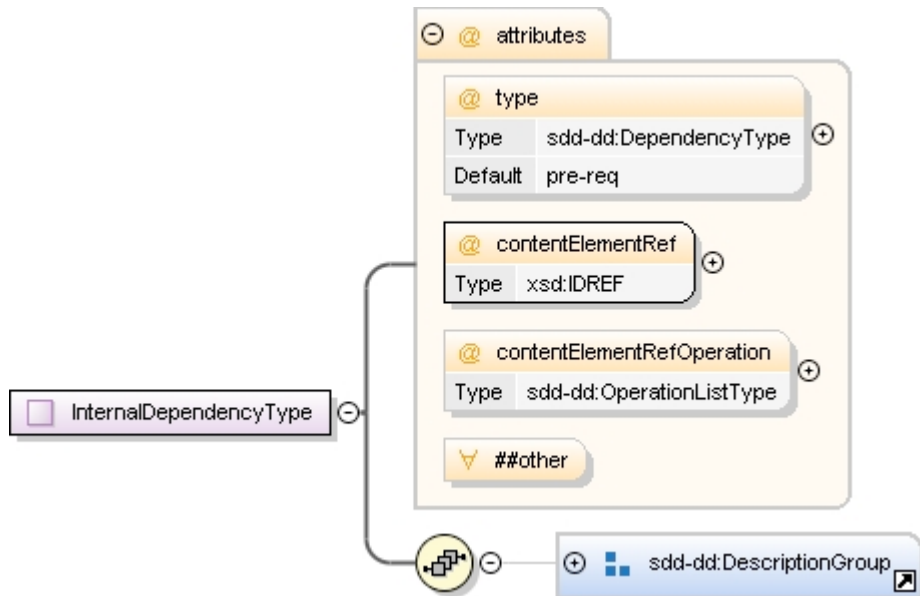
Figure 71: InternalDependencyType structure.

InternalDependencyType provides the type definition for *Dependency* elements defined in all types of content elements. *Dependency* elements allow the expression of dependence on the application of a particular operation to a content element defined in the deployment descriptor before application of a particular operation on the defining content element. The dependency is associated with an operation on the defining content element by the operation attribute in the *Requirement* defining the *Dependency* element. The dependency is associated with an operation on the depended on content element by the *contentRefOperation* attribute in the *Dependency*. There are three types of dependencies: pre-requisites, co-requisites and ex-requisites.

4.7.6.1 InternalDependencyType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	A human-understandable description of the dependency.
ShortDescription	DisplayTextType	0..1	A short human-understandable description of the dependency.

type	DependencyType	0..1	Type can be "pre-req", "co-req", or "ex-req". **default value="pre-req"
contentElementRef	xsd:IDREF	1	A reference to the content element which is depended on.
contentElementRefOperation	OperationListType	0..1	The dependency is on application of this operation to the content element identified in contentRef.
	xsd:anyAttribute	0..*	

4.7.6.2 InternalDependencyType Property Usage Notes

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the dependency.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

- **type**: See the *DependencyType* section for an explanation of the semantics of each of the possible dependency types [4.7.7].

- **contentElementRef**: The *contentElementRef* value is the *id* of the content element that is depended on.

  The value MUST reference the *id* of a content element.

- **contentElementRefOperation**: When the depended-on content element is an atomic content element, the operation defined here effectively identifies the artifact that must be processed for a pre-requisite or co-requisite or not processed for an ex-requisite.

  When the depended-on content element is a *CompositeUnit*, the operation defined in *contentElementRefOperation* MUST be the top level operation defined by the containing *CompositeInstallable*.

  See the *OperationListType* section for structure and additional usage details [4.3.6].

4.7.7 DependencyType

The *DependencyType* enumeration provides the value for the *type* attribute in *Dependency* elements.

4.7.7.1 DependencyType Property Usage Notes

- **pre-req**: A *pre-req* dependency is satisfied if the other content element is in scope for the deployment. The *pre-req* indicates that the other content element MUST be processed before the content element that defines the *pre-req*.

  The dependency is not met if the other content element is not in scope.

- **co-req**: A *co-req* dependency is satisfied if the other content element is in scope for the deployment. There is no dependence on order of processing.

  The dependency is not met if the other content element is not in scope.

- **ex-req**: An *ex-req* dependency indicates that the other content element MUST NOT be in scope.

  The dependency is not met if the other content element is in scope.

4.7.8 RequiredBaseType
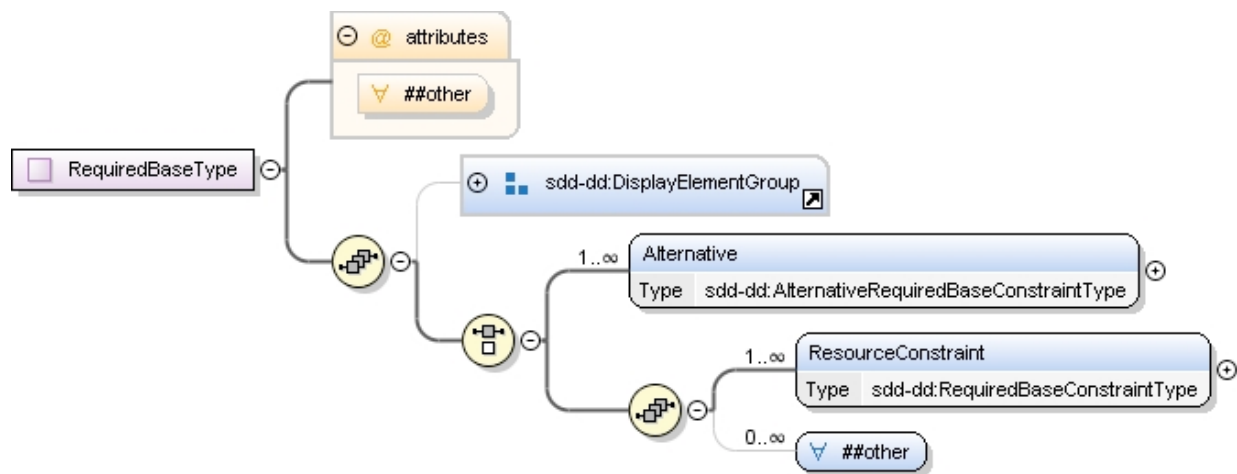
Figure 72: RequiredBaseType structure.

RequiredBaseType provides the type definition for the *RequiredBase* element of *InstallableUnit* and *LocalizationUnit* elements and the *LocalizationBase* element of *LocalizationUnits*. These elements declare the identity characteristics of one or more resources that will be modified or localized by applying of the content element's artifacts. Definition of a *RequiredBase* element represents a requirement that a resource matching the declared characteristic exists. Definition of a *LocalizationBase* element represents a condition on the existence of a resource that matches the declared characteristics.

4.7.8.1 RequiredBaseType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Display name for the requirement on a resource to serve as the base of an update or localization.
Description	DisplayTextType	0..1	Description of the requirement. Required if ShortDescription is defined.
ShortDescription	DisplayTextType	0..1	Short description of the requirement.
Alternative	AlternativeRequiredBaseConstraintType	0..*	Alternative set of constraints on a required base resource.
ResourceConstraint	RequiredBaseConstraintType	1..*	Constraints on the required base resource.
	xsd:any	0..*	
	xsd:anyAttribute	0..*	

4.7.8.2 RequiredBaseType Property Usage Notes

▪ **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the required base element.

See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the required base for this content element.

The *Description* element MUST be defined if the *ShortDescription* element is defined.

See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2544 ▪ **Alternative**: When more than one resource can be used as the update or localization base, two or
2545   more *Alternative* elements are defined to describe the choices. As a convenience for tooling that
2546   produces SDDs, a single *Alternative* can be defined in place of a *ResourceConstraint*.
2547   See the *AlternativeRequiredBaseConstraintType* section for structure and additional usage details
2548   [4.7.10].
2549 ▪ **ResourceConstraint**: *ResourceConstraints* defined here identify one or more particular resources
2550   that can serve as the update or localization base. If *ResourceConstraints* are defined for multiple
2551   resources, they are all updated or localized by application of the content element.
2552   See the *RequiredBaseConstraintType* section for structure and additional usage details [4.7.9].

2553 4.7.9 RequiredBaseConstraintType

2554

2555 Figure 73: RequiredBaseConstraintType structure.

2556 *RequiredBaseConstraintType* provides the type definition for the *ResourceConstraint* elements used in
2557 *RequiredBase* and *LocalizationBase* elements. A required base definition differs from a requirement
2558 definition in the limited nature of the constraints that can be specified. The purpose of constraints within a
2559 required base is to identify resource instances that can be correctly updated or localized by the content
2560 element. Only constraints related to the basic identity characteristics of the resource are allowed.

2561 4.7.9.1 RequiredBaseConstraintType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the constraint.
Description	DisplayTextType	0..1	Description of the constraint.

ShortDescription	DisplayTextType	0..1	Short description of the constraint.
~~Name~~	~~VariableExpressionType~~	~~0..1~~	~~Name of the required base resource as understood in the deployment environment.~~ [DEPRECATED in SDD v2.0]
VersionConstraint	VersionConstraintType	0..1	Allowed versions for the required base resource.
	xsd:any	0..*	
id	xsd:ID	1	Constraint identifier scoped to the deployment descriptor.
resourceRef	xsd:IDREF	1	Reference to the resource representing the required base for an update operation.
testValue	xsd:boolean	0..1	Defines the desired result of the required base constraint. **default value="true"
	xsd:anyAttribute	0..*	

4.7.9.2 RequiredBaseConstraintType Property Usage Notes

2562

2563 ▪ **DisplayName**: This element MAY be used to provide human-understandable information. If used, it
2564 MUST provide a label for the constraint.

2565 See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2566 ▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable
2567 information. If used, they MUST provide a description of the constraint on the required base.

2568 The *Description* element MUST be defined if the *ShortDescription* element is defined.

2569 See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2570 ▪ ~~**Name**: The *Name* element provides the name by which the resource is known in the deployment~~
2571 ~~environment. The value of *Name* is compared to resource names found in the deployment~~
2572 ~~environment as part of constraint evaluation.~~

2573 ~~If the resource name is declared in the referenced resource definition, it SHOULD NOT be declared~~
2574 ~~here. If the resource name is changed by application of the update, the original name SHOULD be~~
2575 ~~declared here and the updated name SHOULD be declared in *ResultingResource*. The name~~
2576 ~~declared here is always the one that represents the required value for the required base.~~

2577 ~~See the *VariableExpressionType* section for structure and additional usage details [4.6.1].~~

2578 [Starting with SDD v2.0, *Name* has been deprecated.]

2579 ▪ **VersionConstraint**: The *VersionConstraint* element defines the set of versions that can serve as a
2580 base for the update.

2581 See the *VersionConstraintType* section for structure and additional usage details [4.4.7].

2582 ▪ **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
2583 log and trace messages.

2584 ▪ **resourceRef**: The *resourceRef* attribute value MUST reference the *id* of the resource element in
2585 *Topology* to which this constraint refers.

2586 ▪ **testValue**: The required base constraint is met when the boolean result of comparing the declared
2587 version to the actual version is equal to the boolean value specified in *testValue*.

2588 Because the purpose of a required base constraint is to positively identify one or more resources that
2589 can serve as the base for an update or localization, there MUST always be one *ResourceConstraint*
2590 that has *testValue* set to "true".

2591 Additional *ResourceConstraints* can be defined with *testValue* set to "false". These constraints
2592 identify characteristics of the same required base resource that must not be true for that resource to
2593 serve as the base.

4.7.10 AlternativeRequiredBaseConstraintType
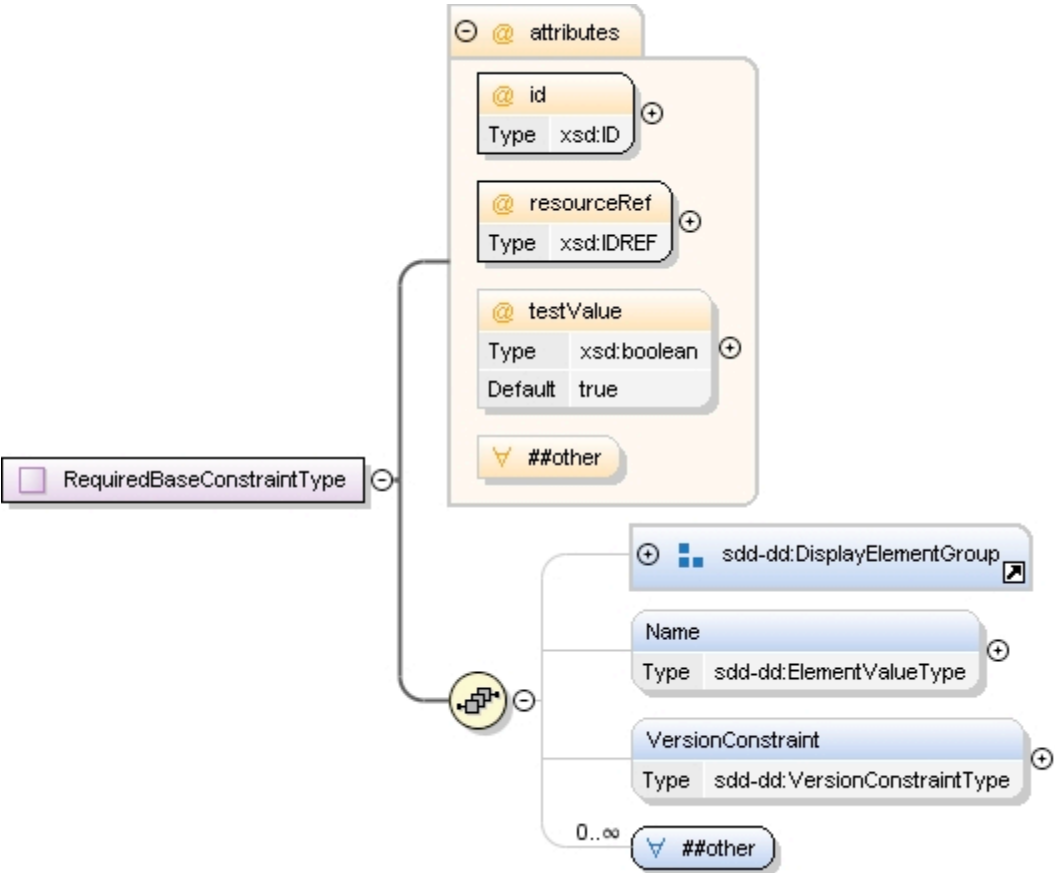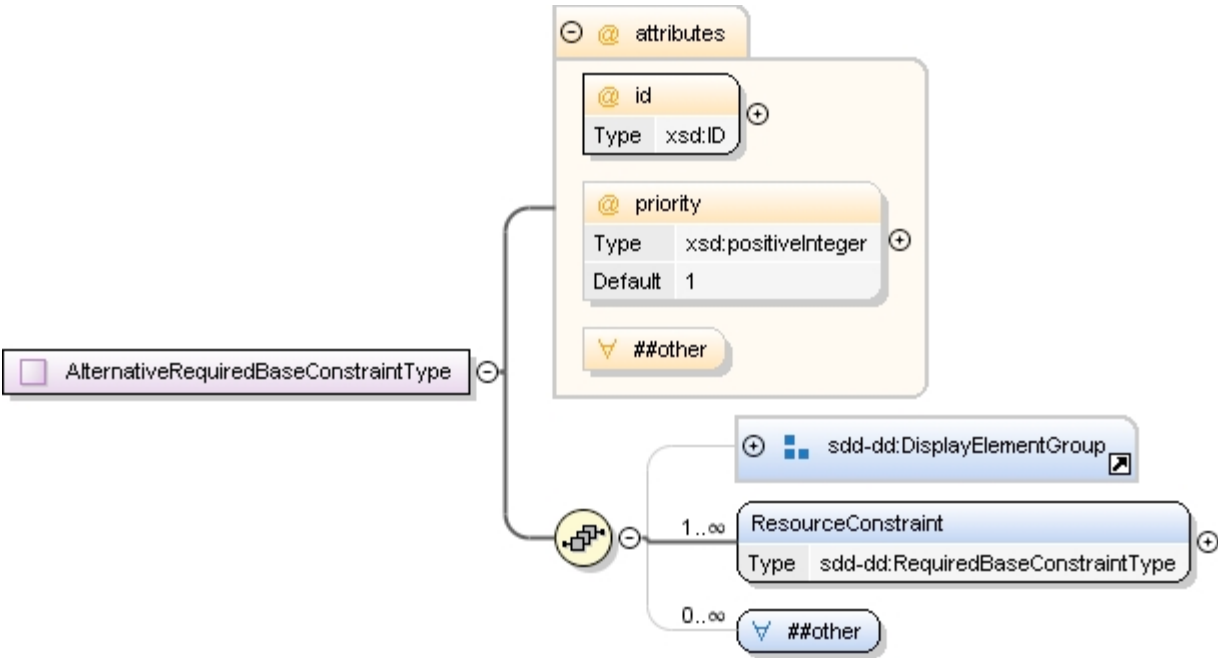

2595

Figure 74: AlternativeRequiredBaseConstraintType structure.

AlternativeRequiredBaseConstraintType provides the type definition for the *Alternative* elements used in *RequiredBase* and *LocalizationBase* elements.

4.7.10.1 AlternativeRequiredBaseConstraintType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	Name of the constraint.
Description	DisplayTextType	0..1	Description of the constraint.
ShortDescription	DisplayTextType	0..1	Short description of the constraint.
ResourceConstraint	RequiredBaseConstraintType	1..*	A set of requirements on one resource.
	xsd:any	0..*	
id	xsd:ID	1	Constraint identifier scoped to the deployment descriptor.
priority	xsd:positiveInteger	0..1	Assists in determining alternative selected when multiple alternatives evaluate to true. **default value="1"
	xsd:anyAttribute	0..*	

4.7.10.2 AlternativeRequiredBaseConstraintType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the alternative.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the alternative.

2606   The *Description* element MUST be defined if the *ShortDescription* element is defined.

2607   See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

2608   ▪   **ResourceConstraint**: *ResourceConstraints* defined here identify one or more particular resources
2609   that can serve as the update or localization base. If *ResourceConstraints* are defined for multiple
2610   resources, they are all updated or localized by application of the content element.

2611   See the *RequiredBaseConstraintType* section for structure and additional usage details [4.7.9].

2612   ▪   **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
2613   log and trace messages.

2614   ▪   **priority**: If there are multiple satisfied alternatives during a particular solution deployment, one of the
2615   alternatives must be selected. The *priority* attribute communicates the SDD author's prioritization of
2616   the alternatives. A lower number represents a higher priority with "1" representing the highest priority.
2617   Other inputs may also be used to select an alternative. The criteria for making this selection are
2618   outside of the scope of the SDD.

4.8 Resulting and Changed Resources

2620   Deployment of an SDD package creates or modifies software resources. These resources are included in
2621   the *Topology* definition and described in more detail in *ResultingResource* and *ResultingChange*
2622   elements.

2623   The SDD author can choose to model resulting and modified resources at a very granular level, at a very
2624   coarse level; at any level in between, or not at all. An example of modeling resulting resources at a
2625   granular level would be modeling every file created by the deployment as a resulting resource. An
2626   example of modeling resulting resources at a very coarse level would be modeling the software product
2627   created by deployment as a single resulting resource. The choice depends on the needs of the solution
2628   deployment. If a resource is not modeled in the SDD, no requirements can be expressed on it, no
2629   conditions can be based on it and no variables can be set from values of its properties. It cannot play any
2630   of the roles described for resources in the *ResourceType* section of this document [4.2.2].

4.8.1 ResultingResourceType

2632
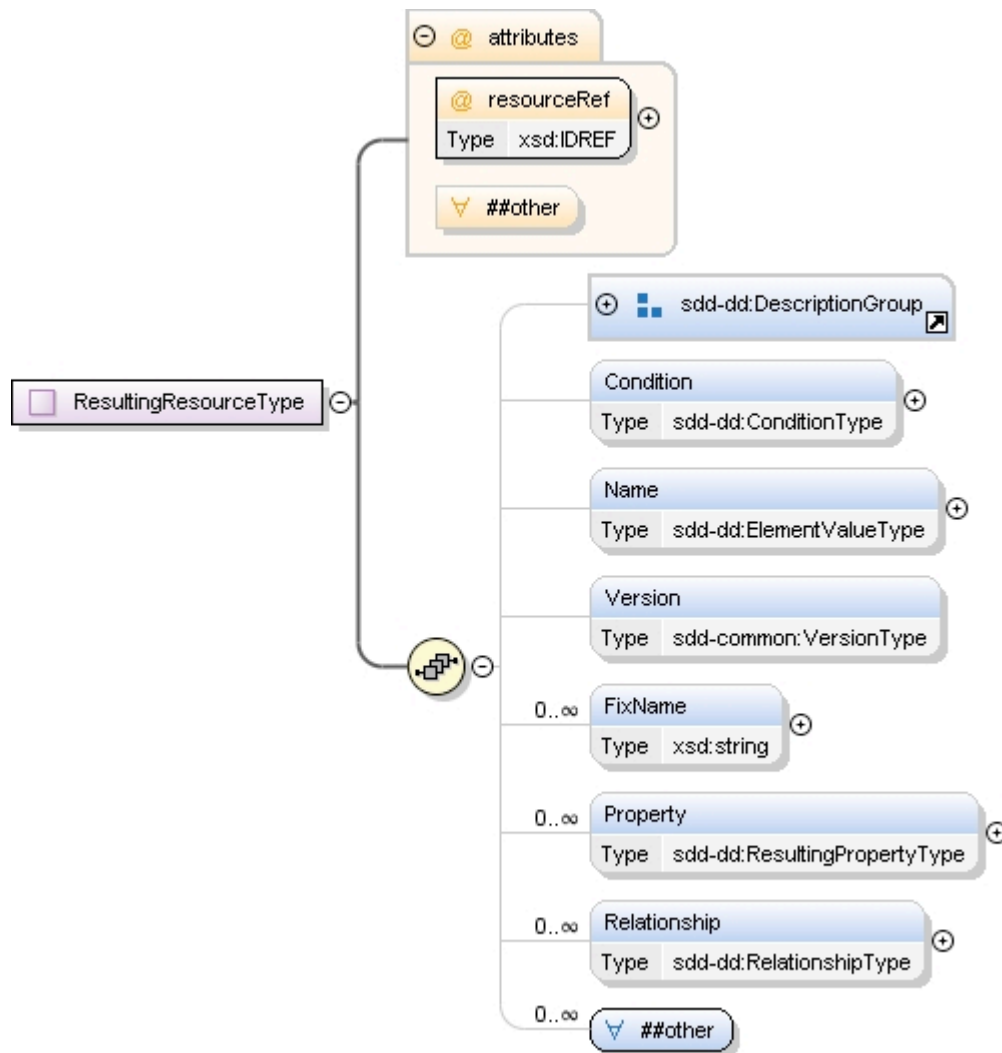
Figure 75: ResultingResourceType structure.

InstallableUnit and *LocalizationUnit* content elements can include zero or more *ResultingResource* elements that describe the key resources installed or updated when the content element's artifacts are processed. The type definition for these elements is provided by *ResultingResourceType*. *ResultingResource* elements refer to resources in topology and define characteristics of those resources that will become true when the artifact is applied. The deployment descriptor author MAY omit the *ResultingResource* element from the content element and the definition of the resource from *Topology* when no knowledge of their existence is required for deployment of the solution or for aggregation of the solution. Characteristics that exist in *ResultingResource* and elsewhere, such as *Topology* or *ResultingChange*, MUST NOT conflict.

> For example, if *Topology* specifies a property that indicates that a file must be writable, it would be incorrect for *ResultingResource* to specify that the resulting file resource is read-only.

Example uses of the *ResultingResource* element are to:

- determine whether potentially resulting resources will actually be installed or updated;
- identify the resource associated with a content element that may be subsequently uninstalled using the uninstall information in this SDD;
- discover the components of a logical solution resource previously installed using this SDD;
- check whether or not a content element has already been installed.

4.8.1.1 ResultingResourceType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	Description of the effect of the content element on the resulting resource.
ShortDescription	DisplayTextType	0..1	Short description of the effect of the content element on the resulting resource.
Condition	ConditionType	0..1	A condition that determines if the resulting resource definition is relevant to a particular deployment.
~~Name~~	~~VariableExpressionType~~	~~0..1~~	~~Name of the resulting resource as known in the deployment environment.~~ [DEPRECATED in SDD v2.0]
Version	VersionType	0..1	Version of the resulting resource.
FixName	xsd:string	0..*	Name of a resulting fix.
Property	ResultingPropertyType	0..*	A resulting property setting of the resulting resource.
Relationship	RelationshipType	0..*	A relationship that will exist after creating or updating the resource.
	xsd:any	0..*	
resourceRef	xsd:IDREF	1	Reference to a resource in topology.
	xsd:anyAttribute	0..*	

4.8.1.2 ResultingResourceType Property Usage Notes

▪ **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the effect of the content element on the resulting resource.

The *Description* element MUST be defined if the *ShortDescription* element is defined.

See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

▪ **Condition**: A *Condition* is used when the resulting resource will be created by the content element only when certain conditions exist in the deployment environment.

See the *ConditionType* section for structure and additional usage details [4.5.1].

▪ ~~**Name**: The name of the resulting resource SHOULD be defined in the *ResultingResource* element and not in *Topology* when the content element installs the resulting resource. The resource name comes into existence when the resulting resource is created. When the content element updates the resulting resource without changing the resource name, *Name* SHOULD be defined in *Topology*. *Name* SHOULD NOT be defined in both places. If a resource name is defined in both *Topology* and *ResultingResource*, the values MUST match.~~

~~See the *VariableExpressionType* section for structure and additional usage details [4.6.1].~~

[Starting with SDD v2.0, *Name* has been deprecated. See the *Property* element below for the appropriate method for specifying a resource identifier.]

▪ **Version**: This is the version of the resource after processing the content element's artifacts. *Version* SHOULD be defined for all resulting resources.

For example, when update artifacts are processed, this version describes the resource after the update is complete.

See the *VersionType* section for structure and additional usage details [3.10].

2675 ▪ **FixName**: Multiple *FixName* elements MAY be included to identify the resulting resource fixes that
2676 will exist once the content element is applied. The *FixName* SHOULD match the names of fixes that
2677 can be detected on the system.

2678 ▪ **Property**: *Property* elements SHOULD be included to identify property values of the resulting
2679 resource that will exist after applying the content element.

2680 Properties of the resulting resource SHOULD be defined in the *ResultingResource* element and not in
2681 *Topology*. They SHOULD NOT be defined in both places. If a property is defined in both *Topology*
2682 and *ResultingResource*, the values MUST match.

2683 If a resource can be identified by a property that represents the name for that resource, the SDD
2684 author SHOULD include a *Property* element and MUST set the value of *PropertyName* to "Name".

2685 See the *ResultingPropertyType* section for structure and additional usage details [4.2.4].

2686 ▪ **Relationship**: *Relationship* elements SHOULD be included to identify relationships that will exist after
2687 applying the content element.

2688 See the *RelationshipType* section for structure and additional usage details [4.8.3].

2689 ▪ **resourceRef**: The *resourceRef* attribute MUST identify the resource in *Topology* that will be installed
2690 or updated when the defining content element is applied.

2691 ## 4.8.2 ResultingChangeType

2692
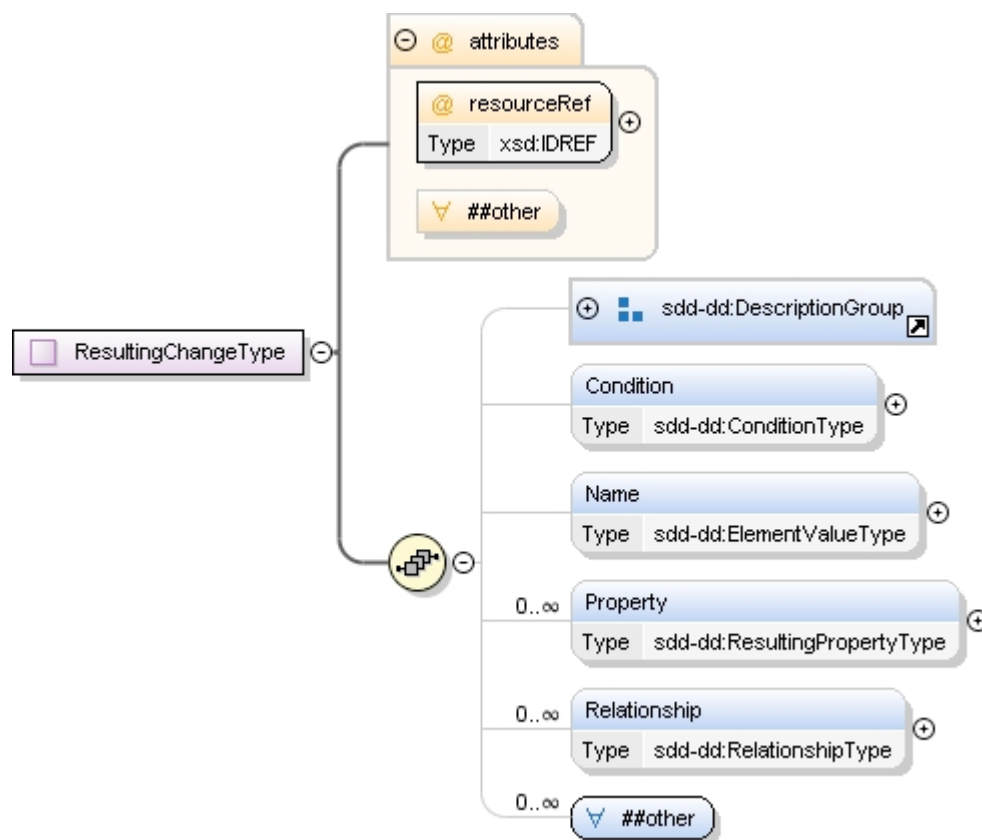
2693 **Figure 76: ResultingChangeType structure.**

2694 *InstallableUnit* and *ConfigurationUnit* content elements can include zero or more *ResultingChange*
2695 elements that describe the key resources whose configuration is modified when the content element's
2696 artifacts are processed. *ResultingChange* elements refer to resources in *Topology* and define
2697 characteristics of those resources that will become true when the content element is applied.

4.8.2.1 ResultingChangeType Property Summary

Name	Type	*	Description
Description	DisplayTextType	0..1	Description of the effect of the content element on the changing resource.
ShortDescription	DisplayTextType	0..1	Short description of the effect of the content element on the changing resource.
Condition	ConditionType	0..1	A condition that determines if the resulting change definition is relevant to a particular deployment.
~~Name~~	~~VariableExpressionType~~	~~0..1~~	~~Name of the resulting resource as known in the deployment environment.~~ [DEPRECATED in SDD v2.0]
Property	ResultingPropertyType	0..*	A resulting property setting of the changing resource.
Relationship	RelationshipType	0..*	Specifies a relationship(s) with another resource that will result from this deployment.
	xsd:any	0..*	
resourceRef	xsd:IDREF	1	Reference to the resource in topology that will be changed by application of the content element.
	xsd:anyAttribute	0..*	

4.8.2.2 ResultingChangeType Property Usage Notes

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the effect of the content element on the changing resource.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DescriptionGroup* section for structure and additional usage details [4.14.1].

- **Condition**: A *Condition* is used when the resulting change will be performed by applying the content element only when certain conditions exist in the deployment environment.

  See the *ConditionType* section for structure and additional usage details [4.5.1].

- ~~**Name**: The *Name* corresponds with the name of the changed resource as known in the deployment environment. *Name* SHOULD be defined in *Topology* and not in *ResultingChange*, because the name is not changed by processing the content elements artifacts. If *Name* is defined in both places, the values MUST match.~~

  ~~See the *VariableExpressionType* section for structure and additional usage details [4.6.1].~~

  [Starting with SDD v2.0, *Name* has been deprecated. See the *Property* element below for the appropriate method for specifying a resource identifier.]

- **Property**: *Property* elements MAY be included to identify property values of the identified resource as they will exist after applying the content element.

  Properties defined in *ResultingChange* MUST be properties that are modified by processing the content element's artifacts.

  If a resource can be identified by a property that represents the name for that resource, the SDD author SHOULD include a *Property* element and MUST set the value of *PropertyName* to "Name".

  See the *ResultingPropertyType* section for structure and additional usage details [4.2.4].

- **Relationship**: When application of the content element results in the creation or modification of relationships, the *Relationship* elements SHOULD be included to identify relationships as they will exist after application of the content element.

  See the *RelationshipType* section for structure and additional usage details [4.8.3].

2726 ▪ **resourceRef**: The *resourceRef* attribute MUST identify the resource whose configuration will be
2727   modified when the defining content element is applied.

2728   The value MUST reference the *id* of a resource specified in *Topology*.

4.8.3 RelationshipType
2729
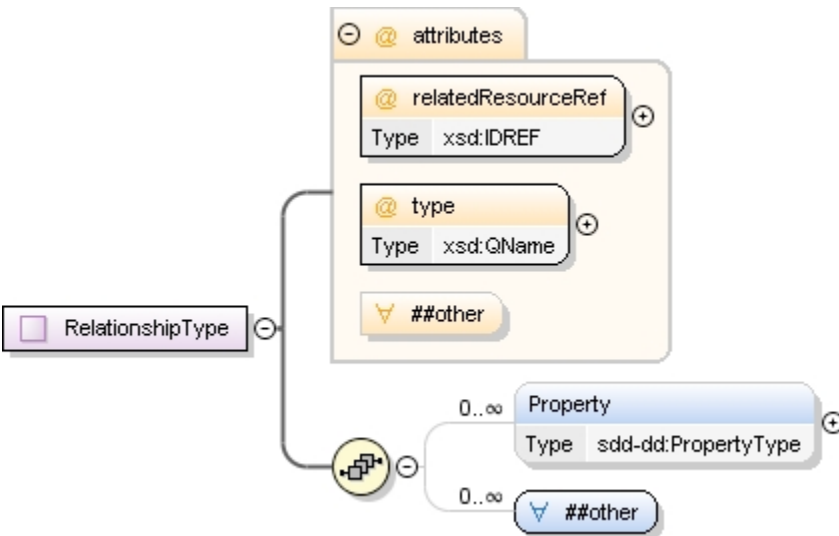
2730
2731 **Figure 77: RelationshipType structure.**

4.8.3.1 RelationshipType Property Summary
2732

Name	Type	*	Description
Property	PropertyType	0..*	A property definition that further constrains the relationship.
	xsd:any	0..*	
relatedResourceRef	xsd:IDREF	1	The second resource in the relationship.
type	xsd:QName	1	The type of the relationship.
	xsd:anyAttribute	0..*	

4.8.3.2 RelationshipType Property Usage Notes
2733

2734 ▪ **Property**: This element MAY be used to provide additional information about the relationship.

2735   For example, a connectivity relationship might specify additional information such as the specific
2736   protocol used (for instance, TCP/IP) and/or particular characteristics of a protocol (for instance,
2737   port number).

2738   See the *PropertyType* section for structure and additional usage details [4.2.3].

2739 ▪ **relatedResourceRef**: There are two resources in any relationship. The first is the resource defined in
2740   the *resourceRef* of the *ResultingResource* or *RelationshipConstraint* element that defines the
2741   *Relationship* element. The second resource is the one identified by *relatedResourceRef*.

2742   The value MUST reference the *id* of a resource specified in *Topology*.

2743 ▪ **type**: Values for relationship type are not defined by the SDD specification. This type may be
2744   specified in profiles [5.3].

4.9 Composite Content Elements

Composite content elements organize the content of an SDD but do not define artifacts used to deploy SDD content. There are three types of composite content elements: *CompositeInstallable*, *CompositeUnit* and *CompositeLocalizationUnit*.

CompositeInstallable is used any time that more than one content element is defined in support of one operation on the package; any time aggregation of SDDs is needed or any time the package includes selectable content.

CompositeInstallable is the root of a content hierarchy that supports a single deployment lifecycle operation. It can define a base content hierarchy, a localization content hierarchy, and/or a selectable content hierarchy and selection criteria. Base content defines content that is deployed by default. Selectable content defines content that can be selected or not by the deployer. Localization content defines content that provides language support. One SDD can have more than one *CompositeInstallable*–each supporting a different operation.

CompositeUnit is used to organize content elements within the base or selectable content hierarchies. *CompositeUnits* can define *InstallableUnits*, *ConfigurationUnits*, *ContainedPackages* and other *CompositeUnits*. Requirements, conditions and variables that are common to all content elements defined by the *CompositeUnit* can be defined on the *CompositeUnit* to avoid repetition. Within the selectable content hierarchy, a *CompositeUnit* can provide an efficient means for selection of a set of related content elements by a *Feature.*

CompositeLocalizationUnit is described in the Localization section [4.13].
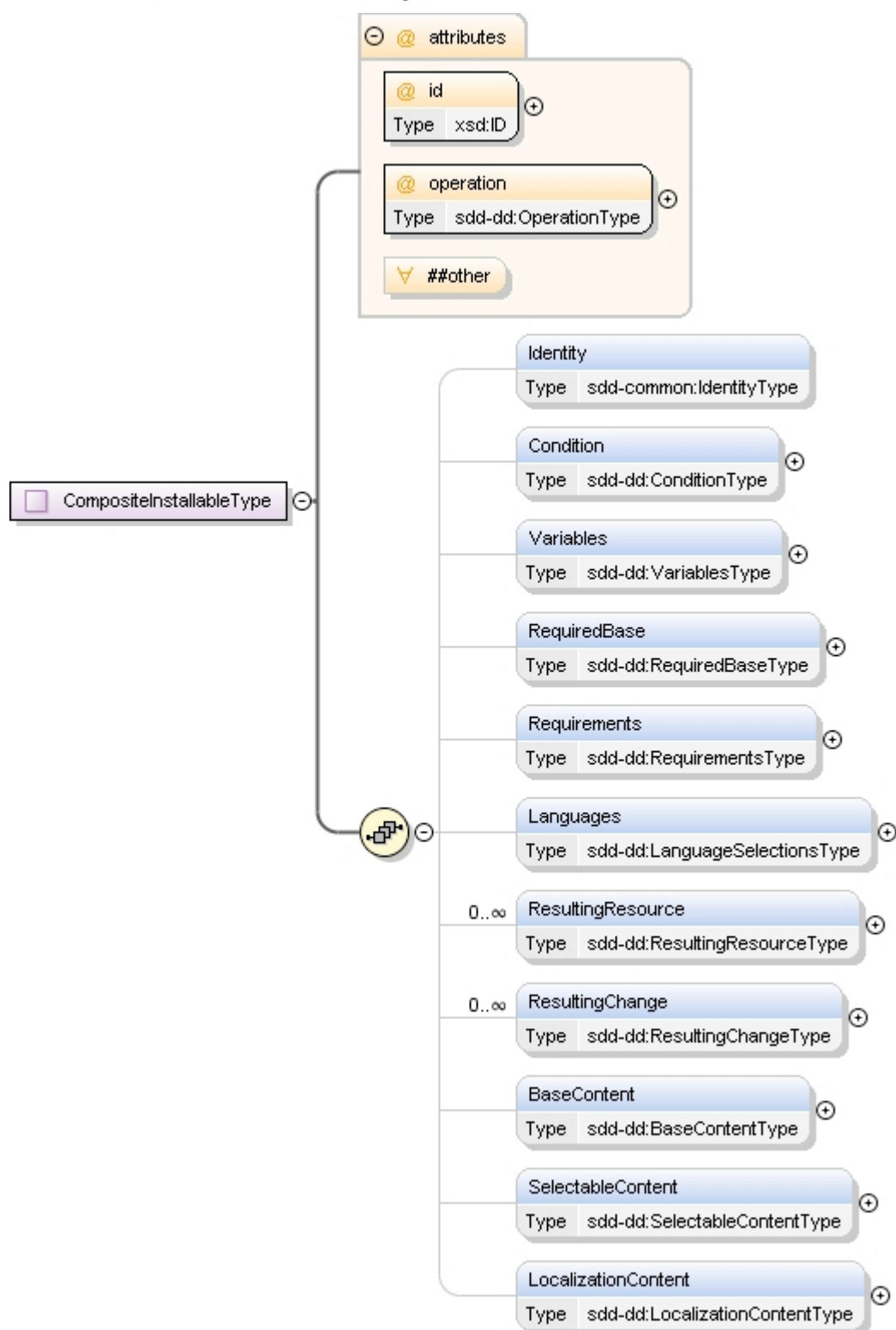
4.9.1 CompositeInstallableType

2766
Figure 78: CompositeInstallableType structure.

2768 A *CompositeInstallable* supports the definition of metadata about package content for one deployment
2769 lifecycle operation. One *CompositeInstallable* can be defined for each operation supported by the

2770 software package. When more than one *CompositeInstallable* is defined in an SDD, there MUST NOT be
2771 more than one *CompositeInstallable* in scope for a particular deployment defined for any one operation.

2772 4.9.1.1 CompositeInstallableType Property Summary

Name	Type	*	Description
Identity	IdentityType	0..1	Human-understandable identity information about the CompositeInstallable.
Condition	ConditionType	0..1	A condition that determines if the content of the CompositeInstallable is relevant to a particular deployment.
Variables	VariablesType	0..1	Variables for use anywhere below the CompositeInstallable and in Topology.
RequiredBase	RequiredBaseType	0..1	Resource or resources that can be updated by the CompositeInstallable.
Requirements	RequirementsType	0..1	Requirements that must be met before successful application of the CompositeInstallable.
Languages	LanguageSelectionsType	0..1	Defines required and selectable languages and groups of languages.
ResultingResource	ResultingResourceType	0..*	Resources that result from applying the CompositeInstallable.
ResultingChange	ResultingChangeType	0..*	Configuration changes that result from applying the CompositeInstallable.
BaseContent	BaseContentType	0..1	Defines content describing the deployment of core resources.
SelectableContent	SelectableContentType	0..1	Defines content describing the deployment of selectable resources.
LocalizationContent	LocalizationContentType	0..1	Defines content whose sole purpose is to provide language support.
id	xsd:ID	1	A unique identifier for the CompositeInstallable element.
operation	OperationType	1	The deployment lifecycle operation described by the CompositeInstallable definition.
	xsd:anyAttribute	0..*	

2773 4.9.1.2 CompositeInstallableType Property Usage Notes

2774 ▪ **Identity**: This identity MAY have values in common with the identity of a resulting resource created
2775 when artifacts defined by content of the composite are processed.

2776 If the unit of packaging described by the *CompositeInstallable* is known to a package management
2777 system, the *Identity* elements SHOULD correspond to values associated with that package in the
2778 package management system.

2779 See the *IdentityType* section for structure and additional usage details [3.4].

2780 ▪ **Condition**: When the condition defined in the *CompositeInstallable* is not met for a particular
2781 deployment, the *CompositeUnit* and all the content elements defined below the *CompositeUnit* are
2782 out of scope for that particular deployment.

2783 See the *ConditionType* section for structure and additional usage details [4.5.1].

2784 ▪ **Variables**: Variables defined here are visible throughout the *CompositeInstallable* and in *Topology*.

2785 See the *VariablesType* section for structure and additional usage details [4.6.5].

- **RequiredBase**: When a resource or resources corresponding to the overall software will be modified during deployment, that resource or those resources MAY be defined in the *RequiredBase* element. The *RequiredBase* definition represents a requirement that the described resource be available for modification to apply the single *operation* defined by the *CompositeInstallable*. When *RequiredBase* is defined, the *operation* defined by *CompositeInstallable* MUST be one of the following: *update*, *undo*, *uninstall*, or *repair*. By specifying the required base separately from other requirements, it is possible for consumers of the SDD to easily determine if the base is available before processing other requirements.

  See the *RequiredBaseType* section for structure and additional usage details [4.7.8].

- **Requirements**: These are requirements that must be met regardless of what content is selected for deployment and which conditions within the content hierarchy evaluates to true.

  Requirements that apply only to a portion of the content SHOULD be defined at the point in the content hierarchy where they apply.

  All requirements specified on content elements that are in scope for a particular deployment MUST be met. This represents a logical "AND" of the requirements. Care should be taken by the SDD author to ensure that conflicting requirements cannot be in scope for the same deployment.

  See the *RequirementsType* section for structure and additional usage details [4.7.1].

- **Languages**: When the SDD contains language support, the *Languages* element can be defined to describe the languages supported; which languages are required and which are selectable; and how language selections are grouped.

  Languages defined in the *Mandatory* element under *Languages* are always in scope. Languages defined in the *Optional* element under *Languages* are in scope if selected by the deployer.

  The *Languages* element is used to declare the mandatory and optional language support available in the package. Languages whose support is deployed by *LocalizationUnits* in *LocalizationContent* MUST be defined as either a mandatory language or an optional language. In addition, languages whose support is deployed along with other content by *InstallableUnits* in *BaseContent* or *SelectableContent* SHOULD be defined as a mandatory language.

  See the *LanguageSelectionsType* section for structure and additional usage details [4.13.4].

- **ResultingResource**: The software whose deployment is described by the SDD can be described in the *CompositeInstallable's ResultingResource* element. This software may consist of many resources that are described in the *ResultingResource* elements of the *InstallableUnits* and/or *LocalizationUnits* defined within the *CompositeInstallable*.

  See the *ResultingResourceType* section for structure and additional usage details [4.8.1].

- **ResultingChange**: Configuration changes that result from deployment regardless of selected content or condition evaluation can be described in the *CompositeInstallable's ResultingChange* element.

  Note that a *ResultingChange* is a change that is made to an existing resource. This is in contrast with *ResultingResource,* which describes newly created resources.

  See the *ResultingChangeType* section for structure and additional usage details [4.8.2].

- **BaseContent**: The base content hierarchy defines content elements that are in scope by default. These content elements MAY be conditioned out based on characteristics of the deployment environment, but are not optional from the deployer's perspective.

  See the *BaseContentType* section for structure and additional usage details [4.11.1].

- **SelectableContent**: Content that is selected by feature MUST be defined in the selectable content hierarchy. *Groups* and *Features* that select this content are also defined within *SelectableContent*.

  See the *SelectableContentType* section for structure and additional usage details [4.12.1].

- **LocalizationContent**: All *LocalizationUnits* and *ContainedLocalizationPackages* MUST be defined in the *LocalizationContent* hierarchy. Each *LocalizationUnit* contains information about the languages it supports and the resources it localizes. This information is evaluated to determine if the *LocalizationUnit* is in scope for a particular deployment.

2835   Each *LocalizationUnit* and *ContainedLocalizationPackage* defined in *LocalizationContent* MAY
2836   support any combination of *Mandatory* and *Optional* languages and can localize any combination of
2837   base and selectable resources, as well as resources already deployed.

2838   Some language support may be deployed incidentally by artifacts in an *InstallableUnit* along with
2839   deployment of other solution content. *LocalizationContent* holds only content elements whose sole
2840   purpose is to provide language support.

2841   *LocalizationContent* supports advanced management of language support, including definition of
2842   mandatory and optional languages and support of localization materials with a lifecycle that is
2843   somewhat independent of the resources localized. When an SDD author has no need for advanced
2844   management of language support, all language support MAY be delivered with other content in
2845   *InstallableUnits*.

2846   See the *LocalizationContentType* section for structure and additional usage details [4.13.1].

2847   ▪ **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
2848   log and trace messages.

2849   ▪ **operation**: This is the *operation* that may be applied to the SDD package whose metadata is
2850   described by the *CompositeInstallable*.

2851   See the *OperationType* section for enumeration values and their meaning [4.3.7].

4.9.2 CompositeUnitType
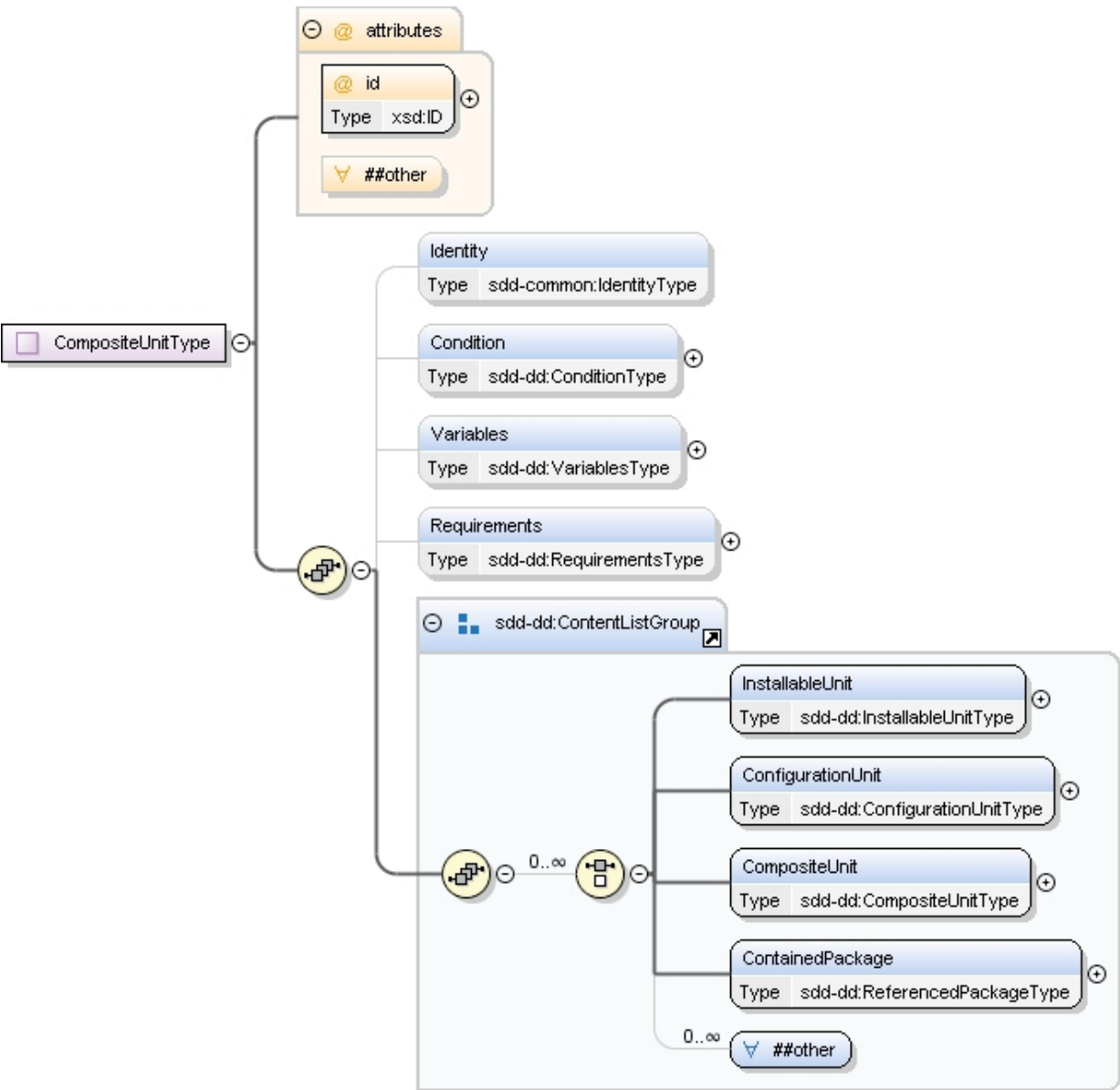
Figure 79: CompositeUnitType structure.

The *CompositeUnit* element is used to organize content elements within the base or selectable content hierarchies. It can define any number of *InstallableUnits*, *ConfigurationUnits*, *ContainedPackages* and other *CompositeUnits*. Composite units assist in organizing the deployment package. A composite unit can provide a convenient way to specify variables, requirements, conditions and other information that applies to every content element defined below the composite unit. Within the selectable content hierarchy, composite units can be used to group content elements that are selected by feature sets or groups. When a feature containing a composite unit is selected, all its child content elements are selected by association. Organization of content within a composite unit does not imply any relationships among the resources that result from deployment of the composite content.

4.9.2.1 CompositeUnitType Property Summary

Name	Type	*	Description
Identity	IdentityType	0..1	Human-understandable identity information about the

			CompositeUnit.
Condition	ConditionType	0..1	A condition that determines if the CompositeUnit and its child content elements are relevant to a particular deployment.
Variables	VariablesType	0..1	Variables for use within the CompositeUnit's and its child content elements' requirement and artifact definitions.
Requirements	RequirementsType	0..1	Requirements that must be met prior to successful processing of any of the CompositeUnit's content.
InstallableUnit	InstallableUnitType	0..*	An InstallableUnit that is part of the composite content.
ConfigurationUnit	ConfigurationUnitType	0..*	A ConfigurationUnit that is part of the composite content.
CompositeUnit	CompositeUnitType	0..*	A CompositeUnit that organizes a subset of the composite's content.
ContainedPackage	ReferencedPackageType	0..*	A ContainedPackage that is part of the composite content.
	xsd:any	0..*	
id	xsd:ID	1	An identifier for the CompositeUnit scoped to the deployment descriptor.
	xsd:anyAttribute	0..*	

4.9.2.2 CompositeUnitType Property Usage Notes

2865

2866 ▪ **Identity**: This identity MAY have values in common with the identity of a resulting resource created
2867 when artifacts defined by content of the composite are processed.

2868 If the unit of packaging described by the *CompositeUnit* is known to a package management system,
2869 some of the identity elements MAY correspond to values associated with that package in the package
2870 management system.

2871 See the *IdentityType* section for structure and additional usage details [3.4].

2872 ▪ **Condition**: When the condition defined in the *CompositeInstallable* is not met for a particular
2873 deployment, the *CompositeUnit* and all the content elements defined below the *CompositeUnit* are
2874 out of scope for that particular deployment.

2875 See the *ConditionType* section for structure and additional usage details [4.5.1].

2876 ▪ **Variables**: Variables defined here are visible within the *CompositeUnit* and every content element
2877 defined below the *CompositeUnit*.

2878 These variables are in scope for a particular deployment only if the *CompositeUnit* is in scope for that
2879 deployment.

2880 See the *VariablesType* section for structure and additional usage details [4.6.5].

2881 ▪ **Requirements**: These are requirements that must be met before any of the artifacts in the
2882 *CompositeUnit* hierarchy can be processed.

2883 These requirements are in scope for a particular deployment only if the *CompositeUnit* is in scope
2884 that deployment.

2885 The *operation* defined for a *Requirement* defined in a *CompositeUnit* MUST be the same as the
2886 *operation* defined by the *CompositeInstallable* containing the *CompositeUnit*.

2887 See the *RequirementsType* section for structure and additional usage details [4.7.1].

2888 ▪ **InstallableUnit**: See the *InstallableUnitType* section for structure and additional usage details [4.3.1].

2889 ▪ **ConfigurationUnit**: See the *ConfigurationUnitType* section for structure and additional usage details
2890 [4.3.2].

2891 ▪ **CompositeUnit**: A *CompositeUnit* element MAY contain child *CompositeUnits*.

▪ **ContainedPackage**: See the *ReferencedPackageType* section for structure and additional usage
details [4.10.1].
2894 ▪ **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
2895   log and trace messages.

4.10 Aggregation

2896

2897 SDD packages can aggregate other SDD packages. Metadata about the aggregation is defined in
2898 *ContainedPackage*, *ContainedLocalizationPackage* and *Requisite* elements. *ContainedPackage*
2899 elements are content elements that can be defined anywhere in the base and selectable content
2900 hierarchies. *ContainedLocalizationPackages* are content elements that can be defined in the localization
2901 content hierarchy. *Requisites* are packages that can be deployed, if necessary, to satisfy requirements in
2902 the aggregating SDD. They are not content of the SDD package. The type of all three of these elements
2903 is *ReferencedPackageType*. The term *referenced package* is used in this specification when referring to
2904 these elements as a group. The term *referenced SDD* is used when referring to any aggregated SDD.

2905 When an SDD aggregates other SDDs, the package descriptors of the aggregated SDDs are included in
2906 the *Contents* list in the package descriptor of the aggregating SDD (see Figure 80). The referenced
2907 package elements in the deployment descriptor identify a referenced SDD package by referencing its
2908 package descriptor definition in *Contents*. Each referenced package element can further constrain the
2909 deployment of the referenced SDD by defining additional requirements; by mapping resources defined in
2910 the aggregating SDD to those defined in the referenced SDD; and by determining feature selections for
2911 deployment of the referenced SDD.

2912
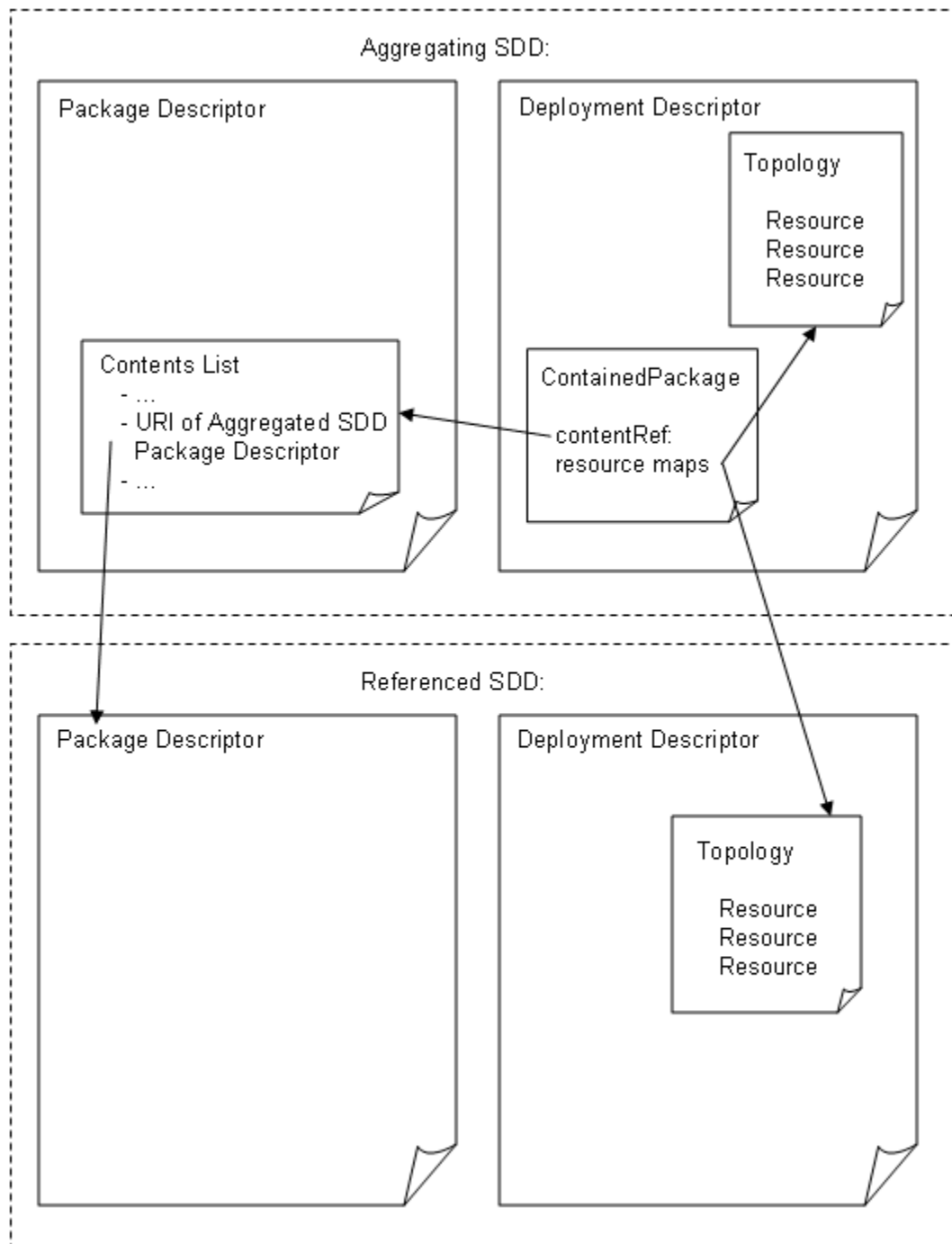
Figure 80: The aggregating SDD identifies the package descriptor of the aggregated SDD and maps resource definitions in the aggregating SDD to resource definitions in the aggregated SDD.

Referenced packages can create and modify software resources that may be required by the aggregating SDD or other SDDs in the aggregation. These resources are mapped to the associated resource definitions in the aggregating SDD by using the *ResultingResourceMap*, the *ResultingChangeMap* and the *RequiredResourceMap* elements of a referenced package element. The characteristics of these resources that other SDDs in the aggregation depend on in some way MUST be exposed in the *ResultingResourceMap,* the *ResultingChangeMap* and the *RequiredResourceMap* elements of the aggregating SDD (see Figure 81). These exposed characteristics are mapped to requirements, conditions and resource variables in the SDDs to determine if requirements are satisfied, conditions are met and to set the values of resource property variables (see Figure 82).

2924
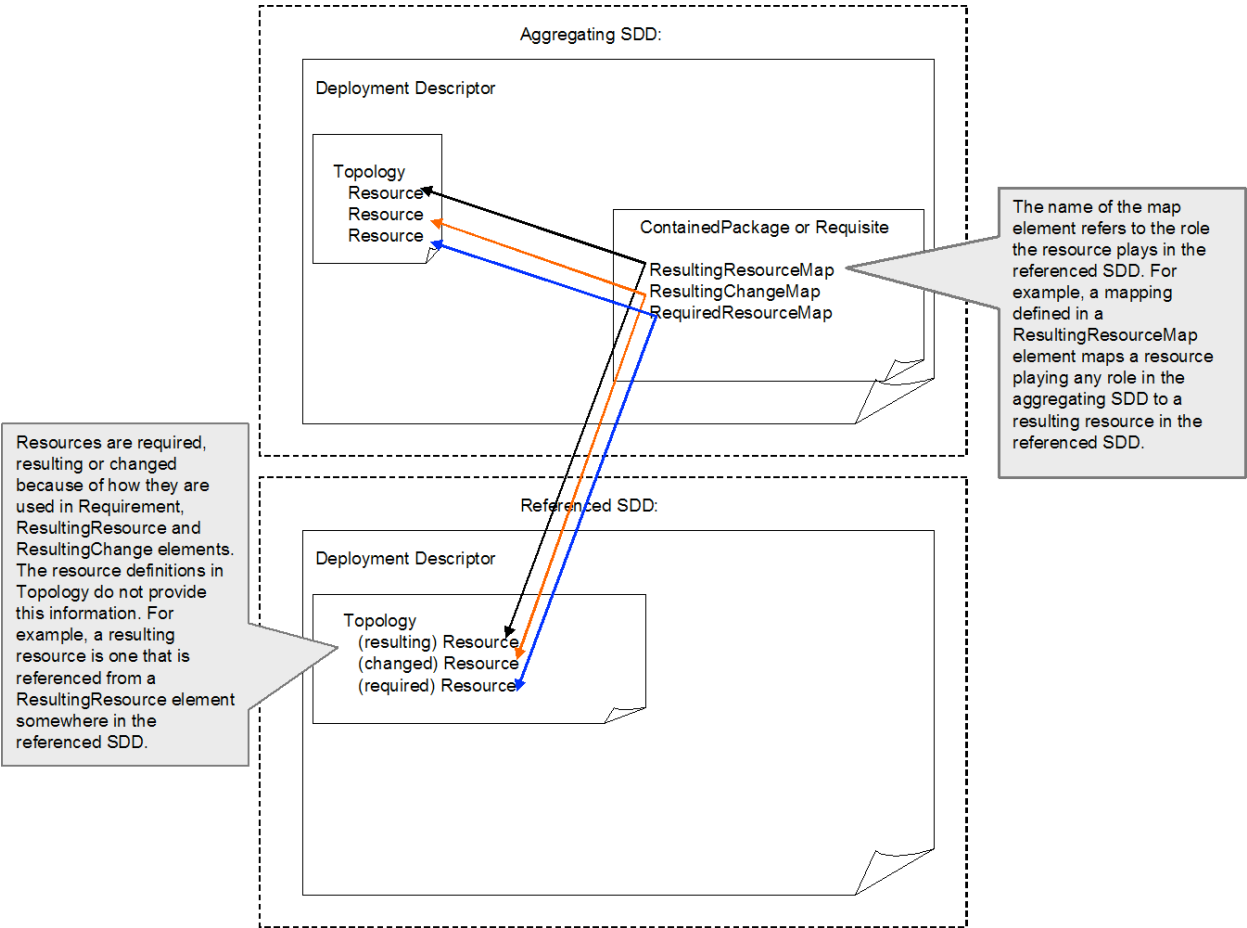
2925
2926 **Figure 81: The list of resource maps is segmented by the role the resource plays in the referenced**
2927 **SDD.**

2928

Figure 82: Arguments and OutputVariables of ReferencedPackageType map variables in the aggregating SDD to variables in the referenced SDD.

It is important to remember that all *id* attributes MUST be unique within a *DeploymentDescriptor* and MUST be unique across an aggregation of SDDs, including referenced SDDs.

4.10.1 ReferencedPackageType

Figure 83: ReferencedPackageType structure.

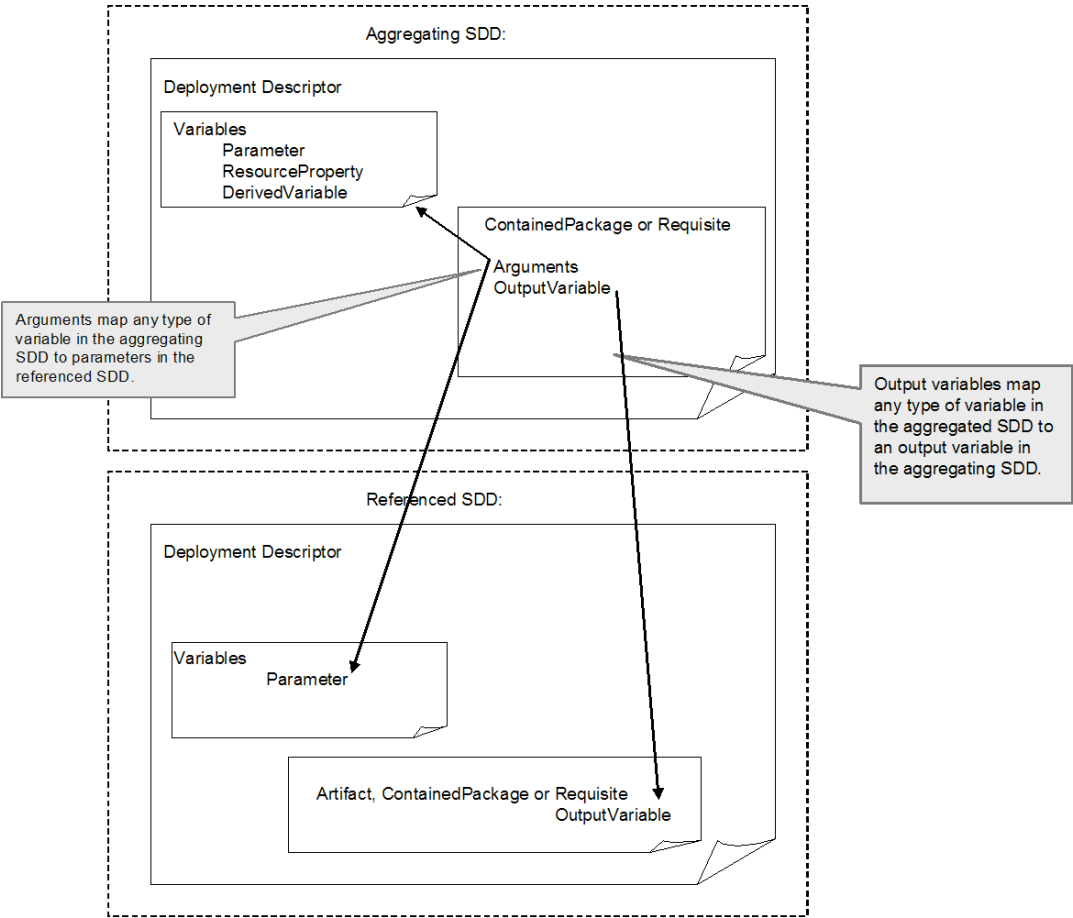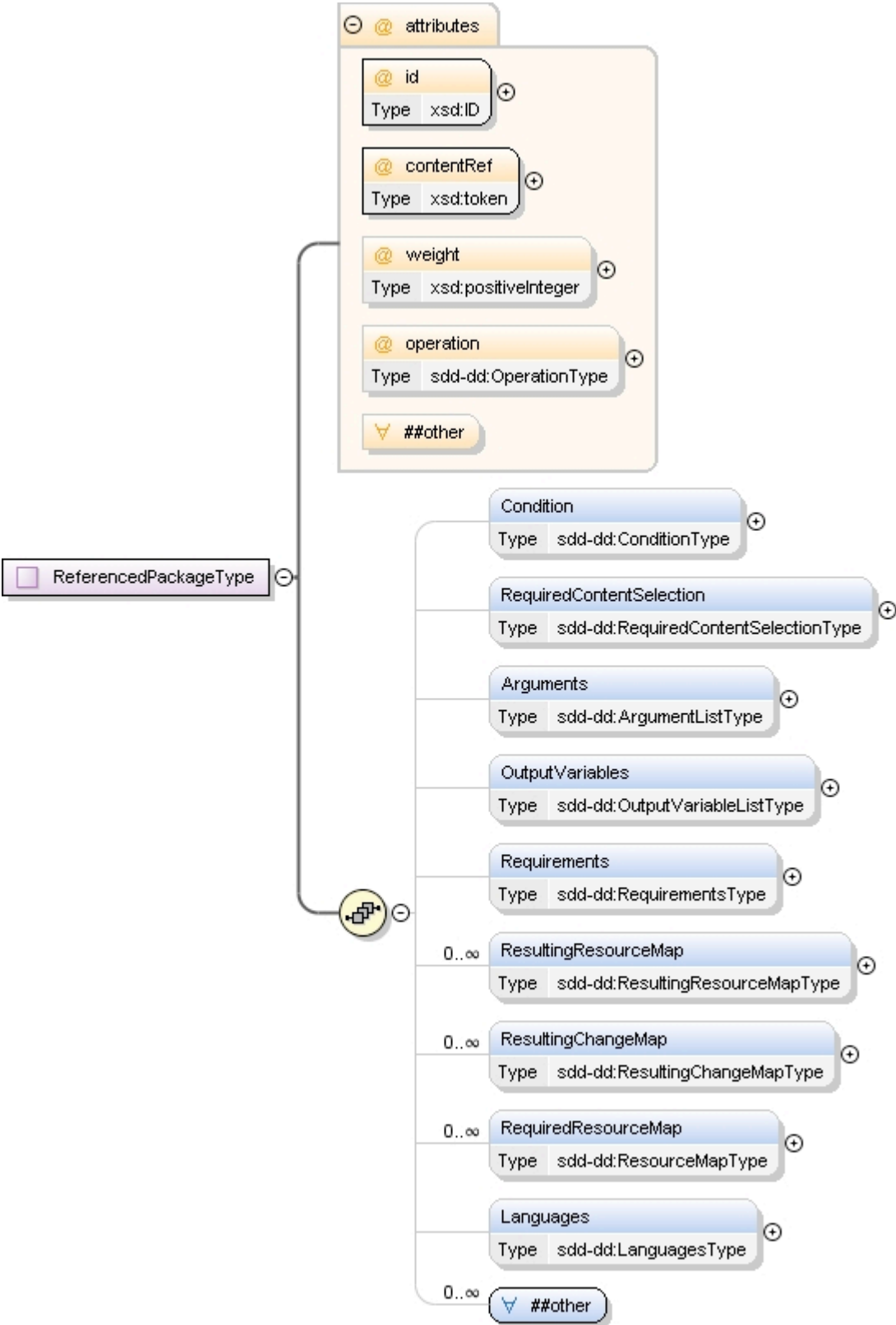2937 A referenced package identifies an aggregated SDD and describes the conditions of its aggregation.
2938 *ReferencedPackageType* provides the type definition for *ContainedPackage* and *Requisite* elements.
2939 *ContainedPackage* elements identify an SDD package that is treated like a content element of the
2940 defining SDD. *Requisite* elements identify an SDD package that can be deployed, if necessary, to satisfy
2941 resource constraints.

2942 ### 4.10.1.1 ReferencedPackageType Property Summary

Name	Type	*	Description
Condition	ConditionType	0..1	A condition that determines if the referenced package is relevant to a particular deployment.
RequiredContentSelection	RequiredContentSelectionType	0..1	A list of groups and features that MUST be selected when the referenced package is deployed.
Arguments	ArgumentListType	0..1	Inputs to the reference package.
OutputVariables	OutputVariableListType	0..1	Outputs from the referenced package.
Requirements	RequirementsType	0..1	Additional requirements for deploying the referenced package as part of the aggregation.
ResultingResourceMap	ResultingResourceMapType	0..*	Maps resulting resources in the referenced package to resources in the referencing package and exposes properties of the resulting resource.
ResultingChangeMap	ResultingChangeMapType	0..*	Maps changed resources defined in the referenced package to resources in the referencing package and exposes changed properties of the resource.
RequiredResourceMap	ResourceMapType	0..*	Maps required resources in the referenced package to resources in the referencing package.
Languages	LanguagesType	0..1	Languages supported by the referenced package.
	xsd:any	0..*	
id	xsd:ID	1	Identifier for the referenced package element that is unique within the deployment descriptor.
contentRef	xsd:token	1	Reference to the identifier of the package Content defined in the package descriptor which identifies the package descriptor of the referenced package.
weight	xsd:positiveInteger	0..1	The time required to process the referenced package relative to all artifacts and other referenced packages in the SDD.
operation	OperationType	0..1	Specifies which operation in the referenced SDD is performed.
	xsd:anyAttribute	0..*	

2943 ### 4.10.1.2 ReferencedPackageType Property Usage Notes

2944 ▪ **Condition**: A *Condition* is used when the *ReferencedPackage's* content should only be deployed
2945 when certain conditions exist in the deployment environment.

2946 See the *ConditionType* section for structure and additional usage details [4.5.1].

2947 ▪ **RequiredContentSelection**: Certain *Groups* or *Features* may need to be selected when deploying
2948   the referenced package. These can be identified in the *RequiredContentSelection* element.

2949   If one particular aggregated SDD requires the selection of different groups or features, depending on
2950   other choices made during a particular deployment, different *Requisite* or *ContainedPackage*
2951   elements can be defined in a way that will cause the correct combination of *Groups* and *Features* to
2952   be used in each situation.

2953   See the *RequiredContentSelectionType* section for structure and additional usage details [4.12.13].

2954 ▪ **Arguments**: Arguments are used to provide values for input variables defined in the deployment
2955   descriptor of the referenced package. The argument name specified MUST reference the *id* of a
2956   parameter in the referenced package.

2957   See the *ArgumentListType* section for structure and additional usage details [4.3.8].

2958 ▪ **OutputVariables**: The output variable mapping can be used to set variables to outputs created by
2959   processing the referenced SDD. The output variables in the referenced package are mapped to
2960   output variables in the aggregating SDD.

2961   Each output variable value specified MUST reference the *id* of an output variable in the referenced
2962   package. This can be an output variable from an artifact or an output variable from a referenced
2963   package defined within the referenced SDD.

2964   See the *OutputVariableListType* section for structure and additional usage details [4.3.10].

2965 ▪ **Requirements**: When the aggregating SDD has stricter requirements for the use of the referenced
2966   SDD than are defined by the referenced SDD itself, those requirements can be defined in
2967   *Requirements*. This is not intended to repeat requirements expressed in the referenced SDD, but
2968   rather to add additional stricter requirements.

2969   Requirements expressed in the referenced SDD need to be satisfied, in addition to the requirements
2970   expressed in the *Requisite* or *ContainedPackage* element of the aggregating SDD.

2971   Requirements expressed in the aggregating SDD MUST NOT conflict with requirements expressed in
2972   the referenced SDD. The requirements specified MUST further constrain the referenced package.

2973   See the *RequirementsType* section for structure and additional usage details [4.7.1].

2974 ▪ **ResultingResourceMap**: Resources created by the referenced package may be resources that are
2975   defined in the aggregating SDD. The *ResultingResourceMap* is used to identify the correspondence
2976   between resource definitions in the aggregating SDD and resulting resource definitions in the
2977   aggregated SDD.

2978   Characteristics of the resulting resources MAY be exposed in the *ResultingResourceMap* element.
2979   *ResourceConstraints* defined on those resources anywhere in the aggregation are mapped to the
2980   resource properties exposed in the resulting maps of the referenced package to determine if the
2981   referenced package will satisfy the constraints. Each individual constraint is considered met by the
2982   referenced package if a property exposed in the resulting resource map that is in scope for the
2983   particular deployment satisfies the constraint.

2984   For example, a property constraint in a *ResourceConstraint* element states that the property
2985   named "FileAttributes" has the value "Writeable". The *resourceRef* in the *ResourceConstraint*
2986   identifies a resource defined in *Topology* that is also identified in the *ResultingResourceMap* of a
2987   *Requisite* or *ContainedPackage* element that is in scope for the particular deployment. If the
2988   *ResultingResourceMap* element contains a statement that the property named "FileAttributes"
2989   has the value "Writeable", then the *ResourceConstraint* is met when the *Requisite* or
2990   *ContainedPackage* is deployed.

2991   This same logic applies to *ResourceConstraints* in aggregated packages. If the SDD in the preceding
2992   example also aggregates another SDD and maps the same resource to a required resource in that
2993   aggregated SDD, then all *ResourceConstraints* in the aggregated SDD are met only if the
2994   *ResultingResourceMap* of the referenced SDD that creates that resource contains a *Version* or
2995   *Property* definition that satisfies the constraint.

2996   See the *ResultingResourceMapType* section for structure and additional usage details [4.10.3].

2997　▪　**ResultingChangeMap**: Resources configured by the referenced package may be resources that are
2998　　defined in the aggregating SDD. The *ResultingChangeMap* is used to identify the correspondence
2999　　between resource definitions in the aggregating SDD and changed resources defined in
3000　　*ResultingChange* elements of the aggregated SDD.

3001　　Characteristics of resources that are changed by the referenced SDD MAY be exposed in the
3002　　*ResultingChangeMap*. These are correlated with *ResourceConstraints* on the changed resource in
3003　　the same manner as the exposed characteristics of a resulting resource. See the property usage
3004　　notes for *ResultingResourceMap* above.

3005　　See the *ResultingChangeMapType* section for structure and additional usage details [4.10.4].

3006　▪　**RequiredResourceMap**: When a resource required by the aggregated SDD is a resource also
3007　　defined in the aggregating SDD, the *RequiredResourceMap* is used to identify the correspondence.
3008　　This element is a simple mapping of a resource in one SDD to a resource in another. There is no
3009　　need to expose characteristics of the resource because it is not created or modified by the referenced
3010　　package.

3011　　One resource MAY be required, resulting, changed, all three or any combination of these within one
3012　　SDD. When a resource in the referenced SDD plays more than one role, the mapping MUST be
3013　　repeated everywhere it applies. This allows exposure of all the created or modified properties in the
3014　　*ResultingChangeMap* and *ResultingResourceMap*. In this situation–when one resource in the
3015　　referenced SDD plays more than one of the roles identified earlier (required, resulting or changed)–all
3016　　mappings MUST be to the same resource in the aggregating SDD. Only the exposed resulting and
3017　　changed properties differ.

3018　　See the *ResourceMapType* section for structure and additional usage details [4.10.2].

3019　▪　**Languages**: Languages supported by the referenced package MAY be identified here. This list does
3020　　not identify mandatory versus optional languages; it is for informational purposes only. The SDD
3021　　author is not limiting use of the referenced package to deployments where all in-scope languages are
3022　　found in this list. There may be cases where aggregated packages are deployed even though they
3023　　cannot support all of the languages supported by the aggregation as a whole.

3024　　Each language specified MUST match a language in the referenced package.

3025　　See the *LanguagesType* section for structure and additional usage details [4.13.6].

3026　▪　**id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
3027　　log and trace messages.

3028　▪　**contentRef**: The package descriptor of an SDD that aggregates other SDDs, either through
3029　　*ContainedPackage* elements or *Requisite* elements, will list the package descriptor files of the
3030　　aggregated SDDs in its content list. The *contentRef* attribute of a referenced package element MUST
3031　　be a reference to the *id* of a *Content* element in the aggregating SDD's package descriptor that
3032　　defines the aggregated package descriptor.

3033　▪　**weight**: Defining weights for all artifacts and referenced packages in an SDD provides useful
3034　　information to software that manages deployment. The weight of the referenced package refers to the
3035　　relative time taken to deploy the referenced package with respect to other packages in this SDD.

3036　　　For example, if the referenced package takes twice as long to deploy as a particular install artifact
3037　　　whose weight is "4", then the weight of the referenced package would be "8". The weight numbers
3038　　　have no meaning in isolation and do not describe actual time elapsed. They simply provide an
3039　　　estimate of relative time.

3040　▪　**operation**: The referenced SDD may support more than one deployment lifecycle operation. The
3041　　*operation* attribute MUST include the operations that are applicable when this is the case.

3042　　See the *OperationType* section for enumeration values and their meaning [4.3.7].
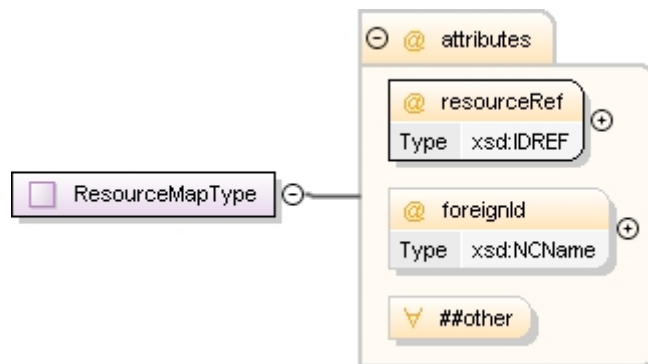
4.10.2 ResourceMapType

3044

Figure 84: ResourceMapType structure.

ResourceMapType is used in the definition of elements that map resources in an SDD to resources in a referenced SDD. The purpose of a resource map is to identify when two resources in separate SDDs MUST resolve to the same resource instance during any particular deployment. The characteristics of a mapped resource that are defined in the topology sections of the two SDDs MUST NOT conflict.

> For example, if a *Property* definition is included for the same property in both SDDs, the value MUST be the same.

Additional characteristics of a mapped resource may be constrained by *Requirements* or *Conditions* in either SDD. All constraints on a mapped resource that are in scope for a particular deployment MUST NOT conflict.

Resources that are not mapped between the two SDDs MAY resolve to the same instance when their characteristics defined in topology do not conflict and when the constraints in scope for any particular deployment do not conflict.

The *RequiredResourceMap*, *ResultingResourceMap* and *ResultingChangeMap* elements all use *ResourceMapType*, either directly or as a base type that is extended.

4.10.2.1 ResourceMapType Property Summary

Name	Type	*	Description
resourceRef	xsd:IDREF	1	Reference to a resource defined in the deployment descriptor.
foreignID	xsd:NCName	0..1	Reference to a resource defined in a referenced deployment descriptor.
	xsd:anyAttribute	0..*	

4.10.2.2 ResourceMapType Property Usage Notes

- **resourceRef**: The value of the *resourceRef* MUST be set to the *id* of the resource in the SDD that is mapped to a resource in a referenced SDD.

- **foreignID**: The value MUST reference the *id* of a resource in the referenced package. This is the resource in the referenced SDD that MUST resolve to the same resource instance as the resource identified in *resourceRef*.
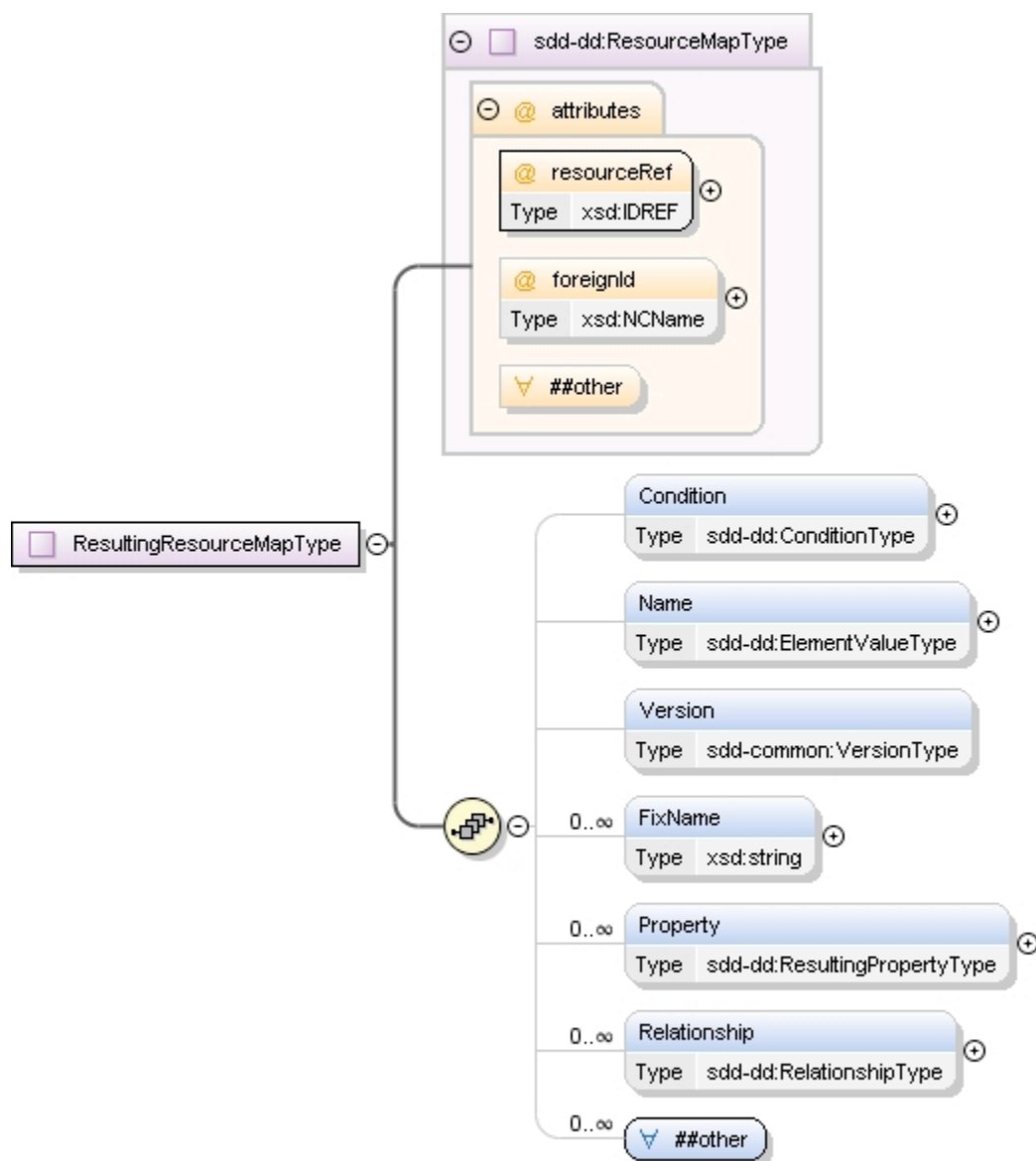

4.10.3 ResultingResourceMapType

3068

Figure 85: ResultingResourceMapType structure.

ResultingResourceMapType defines an element type that maps resources that result from deployment of a referenced SDD to a resource in the referencing SDD. In addition to identifying the two resources that MUST resolve to the same resource instance, the resulting resource map allows characteristics of the resulting resource to be exposed. Constraints might be defined on the mapped resource in the referencing SDD or any referenced SDD. These constraints can be evaluated by comparing the constraint to the characteristics that are exposed in the resulting resource map. The resulting resource map MUST expose sufficient characteristics of the resulting resource to enable the evaluation of constraints on that resource.

For example, suppose that an SDD defines a resource with id="Database" in its topology. The solution can work with Database_Product_A or Database_Product_B. Database_Product_A is created by a referenced SDD defined in a *Requisites* element. The referencing SDD contains *Requirements* and/or *Conditions* that have *Alternatives* for each of the two alternative database products. All constraints on the Database resource that apply to Database_Product_A must be satisfied by a resource characteristic that is exposed in the *ResultingResourceMap* element of the *Requisite* element that references the SDD that deploys Database_Product_A.

4.10.3.1 ResultingResourceMapType Property Summary

Name	Type	*	Description
	[extends] ResourceMapType		See the ResourceMapType section for additional properties [4.10.2].
Condition	ConditionType	0..1	A condition that determines if the resulting resource definition is relevant to a particular deployment.
~~Name~~	~~VariableExpressionType~~	~~0..1~~	~~The name of the resource created or updated by the referenced SDD.~~ [DEPRECATED in SDD v2.0]
Version	VersionType	0..1	The version of the resource created or updated by the referenced SDD.
FixName	xsd:string	0..*	Names of fixes to the mapped resource that are created by the referenced SDD.
Property	ResultingPropertyType	0..*	Properties set when the mapped resource is created or updated by the referenced SDD.
Relationship	RelationshipType	0..*	Relationship that will exist after creating or updating the resource.
	xsd:any	0..*	

4.10.3.2 ResultingResourceMapType Property Usage Notes

See the *ResourceMapType* section for details about the inherited attributes and elements [4.10.2].

- **Condition**: A *Condition* is used when the resulting resource will be created by the referenced package only when certain conditions exist in the deployment environment.

  See the *ConditionType* section for structure and additional usage details [4.5.1].

- ~~**Name**: The *Name* of the resulting resource created or updated by the referenced SDD MUST be defined if it is not defined elsewhere and there are constraints on this resource that contain a *Name* element. "Defined elsewhere" means defined in the topology of the referencing SDD or in the topology of any other referenced SDD for a resource that is also mapped to the same resource. "Constraints on this resource" means a constraint that applies to the particular instantiation of the resource that is created or updated by the referenced SDD, for example a constraint that needs to successfully map to the referenced SDD for the referenced SDD to be used in a particular deployment.~~

  ~~See the *VariableExpressionType* section for structure and additional usage details [4.6.1].~~

  [Starting with SDD v2.0, *Name* has been deprecated. See the *Property* element below for the appropriate method for specifying a resource identifier.]

- **Version**: The *Version* of the resulting resource created or updated by the referenced SDD MUST be defined if it is not defined elsewhere and version constraints are defined on this resource. "Defined elsewhere" means defined in the topology of the referencing SDD or in the topology of any other referenced SDD for a resource that is also mapped to the same resource.

  See the *VersionType* section for structure and additional usage details [3.10].

- **FixName**: One or more names of fixes to the resulting resource created or updated by the referenced SDD MUST be defined if they are not defined elsewhere and version constraints are defined on this resource that include fix names. (See the usage note for *Version* above for a definition of "defined elsewhere".)

- **Property**: A *Property* of the resulting resource created or updated by the referenced SDD MUST be defined if it is not defined elsewhere and property constraints are defined on this property. (See the usage note for *Version* above for a definition of "defined elsewhere".)

  If a resource can be identified by a property that represents the name for that resource, the SDD author SHOULD include a *Property* element and MUST set the value of *PropertyName* to "Name".

3116   See the *ResultingPropertyType* section for structure and additional usage details [4.2.4].

3117 ▪ **Relationship**: Any number of *Relationship* elements can be included to identify relationships that will
3118   exist after applying the referenced package.

3119   See the *RelationshipType* section for structure and additional usage details [4.8.3].
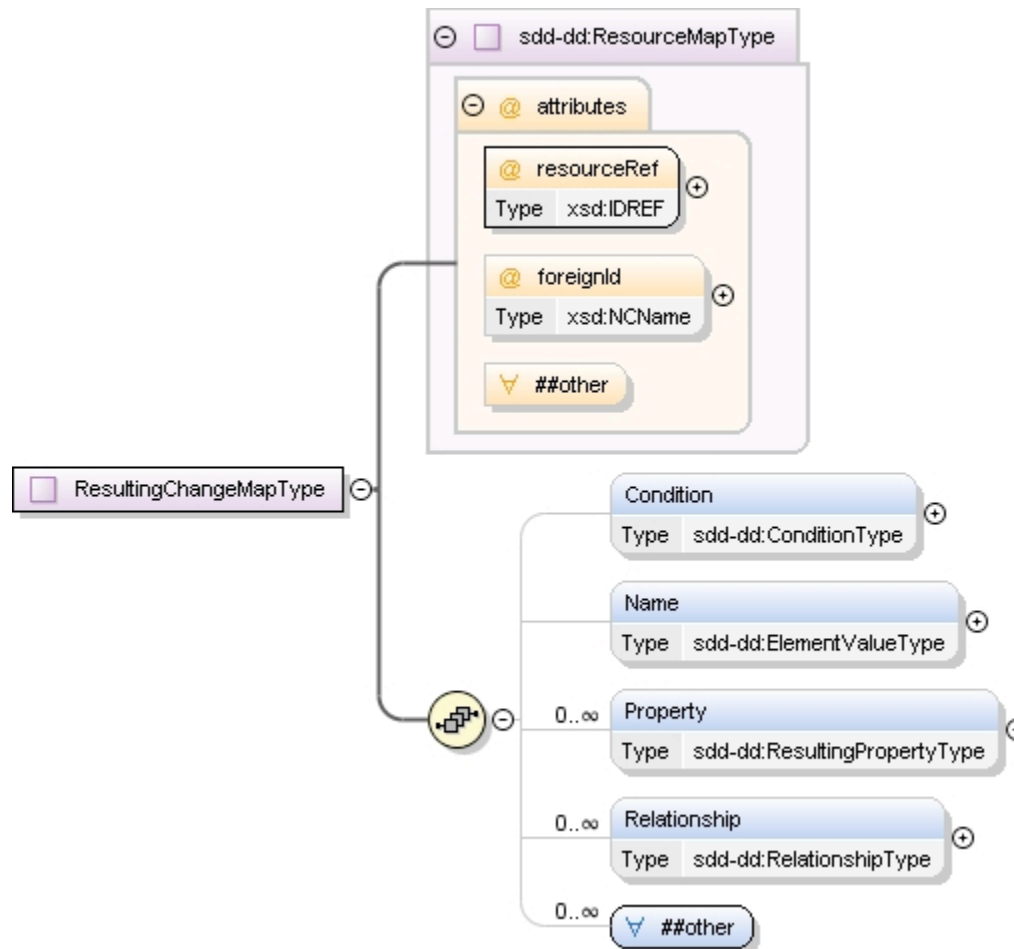
4.10.4 ResultingChangeMapType

3121
Figure 86: ResultingChangeMapType structure.

3123 *ResultingChangeMapType* is similar to *ResultingResourceMapType*. It defines an element type that maps
3124 resources that are changed by deployment of the referenced SDD to a resource in the referencing SDD.
3125 In addition to identifying the two resources that MUST resolve to the same resource instance, the
3126 resulting change map allows characteristics of the modified resource to be exposed. Constraints may be
3127 defined on the mapped resource in the referencing SDD or any referenced SDD. These constraints can
3128 be evaluated by comparing the constraint to the characteristics that are exposed in the resulting change
3129 map. The resulting change map MUST expose sufficient characteristics of the resulting change to enable
3130 the evaluation of constraints on that resource.

3131   For example, suppose that an SDD defines a resource with id="OS" in its topology. The solution can
3132   work with Windows or Linux. Linux is configured by a referenced SDD defined in a *Requisites*
3133   element. The referencing SDD contains *Requirements* and/or *Conditions* that have *Alternatives* for
3134   Windows and for Linux. All constraints on the modified characteristics of Linux must be satisfied by a
3135   resource characteristic that are exposed in the *ResultingChangeMap* element of the *Requisite*
3136   element that references the SDD that configures Linux.

### 4.10.4.1 ResultingChangeMapType Property Summary

Name	Type	*	Description
	[extends] ResourceMapType		See the ResourceMapType section for additional properties [4.10.2].
Condition	ConditionType	0..1	A condition that determines if the resulting change definition is relevant to a particular deployment.
~~Name~~	~~VariableExpressionType~~	~~0..1~~	~~The name of the modified resource.~~ [DEPRECATED in SDD v2.0]
Property	ResultingPropertyType	0..*	A modified property of the resource.
Relationship	RelationshipType	0..*	Relationship that will exist after the change is applied to the resource.
	xsd:any	0..*	

4.10.4.2 ResultingChangeMapType Property Usage Notes

See the *ResourceMapType* section for details about the inherited attributes and elements [4.10.2].

- **Condition**: A *Condition* is used when the resource mapped from the external package will be changed only when certain conditions exist in the deployment environment.

  See the *ConditionType* section for structure and additional usage details [4.5.1].

- ~~**Name**: The *Name* of the resource that is modified by the referenced SDD is defined here to assist with identifying the resource instance that is changed. It is not an indication that the resource name itself is modified by the referenced SDD. If resource characteristics defined in the topology of any SDD defining a resource mapped to the changed resource are sufficient to identify the resource, then *Name* SHOULD NOT be defined in the *ResultingChangeMap*.~~

  ~~See the *VariableExpressionType* section for structure and additional usage details [4.6.1].~~

  [Starting with SDD v2.0, *Name* has been deprecated. See the Property element below for the appropriate method for specifying a resource identifier.]

- **Property**: A modified property MUST be exposed in a *ResultingChangeMap* if it is not defined elsewhere and property constraints are defined on the modified property. "Defined elsewhere" means defined in the topology of the referencing SDD or in the topology of any other referenced SDD for a resource that is also mapped to the same resource. "Constraints on the modified property" means a property constraint that applies to the particular instantiation of the resource that is modified by the referenced SDD; for example, a constraint that must map to the referenced SDD, if the referenced SDD is to be used in a particular deployment.

  If a resource can be identified by a property that represents the name for that resource, the SDD author SHOULD include a *Property* element and MUST set the value of *PropertyName* to "Name".

  See the *ResultingPropertyType* section for structure and additional usage details [4.2.4].

- **Relationship**: *Relationship* elements SHOULD be included to identify relationships that will exist after the application of the referenced package.

  Relationships that need to be known by the aggregate MUST be mapped. Relationships need to be known when they are referred to in one or more resource constraints.

  See the *RelationshipType* section for structure and additional usage details [4.8.3].

4.10.5 RequisitesType
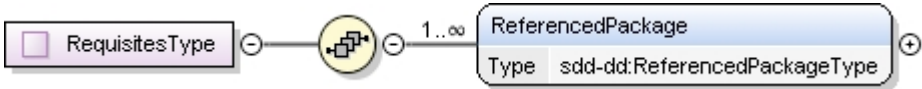
Figure 87: RequisitesType structure.

3169　The *Requisites* element contains a list of references to SDD packages that can be used to satisfy one or
3170　more of the requirements defined by content elements. The definition of a requisite does not imply that it
3171　must be used; only that it is available for use if needed.

3172　Requisite definitions can map values and resources defined in the SDD to inputs and resources defined
3173　in the requisite SDD.

4.10.5.1 RequisitesType Property Summary
3174

Name	Type	*	Description
ReferencedPackage	ReferencedPackageType	1..*	An SDD package that can, but is not required to, be deployed to satisfy a requirement.

4.10.5.2 RequisitesType Property Usage Notes
3175

3176　▪　**ReferencedPackage**: See the *ReferencedPackageType* section for structure and additional usage
3177　　details [4.10.1].

4.11 Base Content
3178

3179　Base content is the default content for the deployment lifecycle operation associated with the
3180　*CompositeInstallable* that contains the base content. This is content that is deployed whenever the
3181　associated operation is performed on the SDD package. Base content may be conditioned on
3182　characteristics of the deployment environment but it is not selectable by the deployer.

3183　Resources associated with base content for one operation may be different from resources associated
3184　with base content for a different operation in the same SDD package.

3185　　For example, base content in the *CompositeInstallable* for the configuration operation may configure
3186　　resources that were created by selectable content in the *CompositeInstallable* for the install
3187　　operation. In this example, the configuration is in base content because it must be done if the
3188　　resource exists. It is not selectable by the deployer during the configuration operation.

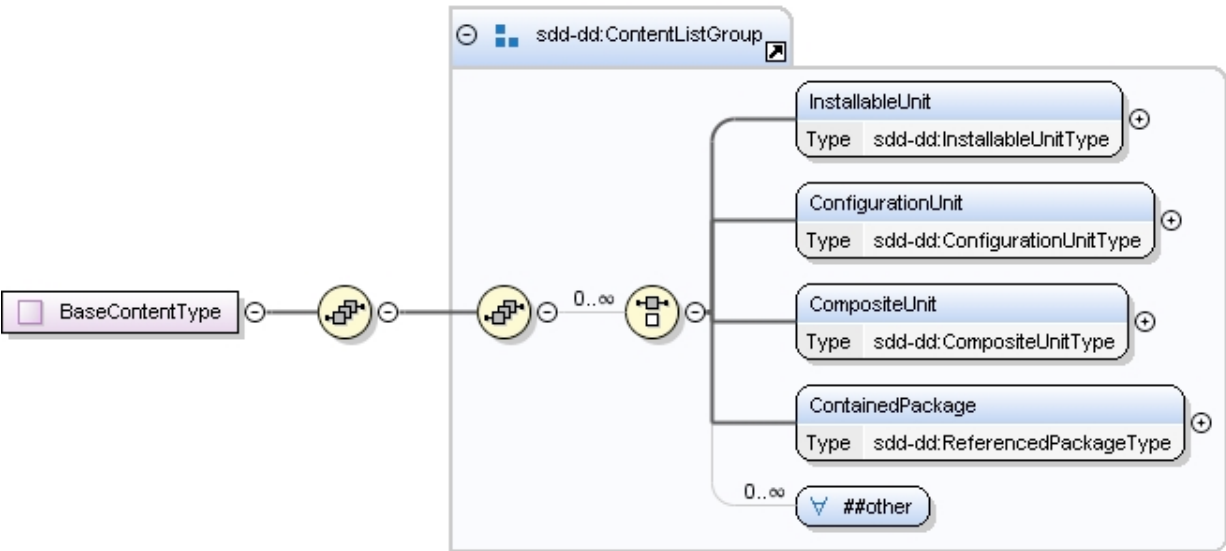### 4.11.1 BaseContentType
3189

3190
3191　**Figure 88: BaseContentType structure.**

3192　The *BaseContent* hierarchy defines the default content for the deployment operation described by the
3193　*CompositeInstallable*. This content MAY be conditioned.

4.11.1.1 BaseContentType Property Summary

Name	Type	*	Description
InstallableUnit	InstallableUnitType	0..*	An InstallableUnit that defines base content.
ConfigurationUnit	ConfigurationUnitType	0..*	A ConfigurationUnit that defines base configuration content.
CompositeUnit	CompositeUnitType	0..*	A CompositeUnit that organizes base content.
ContainedPackage	ReferencedPackageType	0..*	An SDD whose content is considered to be base content in the context of this aggregation.
	xsd:any	0..*	

3195 **4.11.1.2 BaseContentType Property Usage Notes**

3196 ▪ **InstallableUnit**: See the *InstallableUnitType* section for structure and additional usage details [4.3.1].

3197 ▪ **ConfigurationUnit**: See the *ConfigurationUnitType* section for structure and additional usage details
3198   [4.3.2].

3199 ▪ **CompositeUnit**: See the *CompositeUnitType* section for structure and additional usage details
3200   [4.9.2].

3201 ▪ **ContainedPackage**: See the *ReferencedPackageType* section for structure and additional usage
3202   details [4.10.1].

3203 # 4.12 Content Selectability

3204 The SDD author MAY define selectable subsets of content using *Groups* and *Features*. Selectability, as
3205 used in the SDD, is a characteristic of the deployment lifecycle operation and the package. The decision
3206 to provide selectability for one operation in one package has no semantic relationship to the selectability
3207 provided in another package related to the same software. It also has no semantic relationship to the
3208 selectability provided for a different operation within the same package.

3209   For example, when the SDD author chooses to create a feature in a maintenance package, that
3210   feature is designed to allow selectable application of the maintenance, not to reflect the original set of
3211   features for the base content.

3212 ## 4.12.1 SelectableContentType
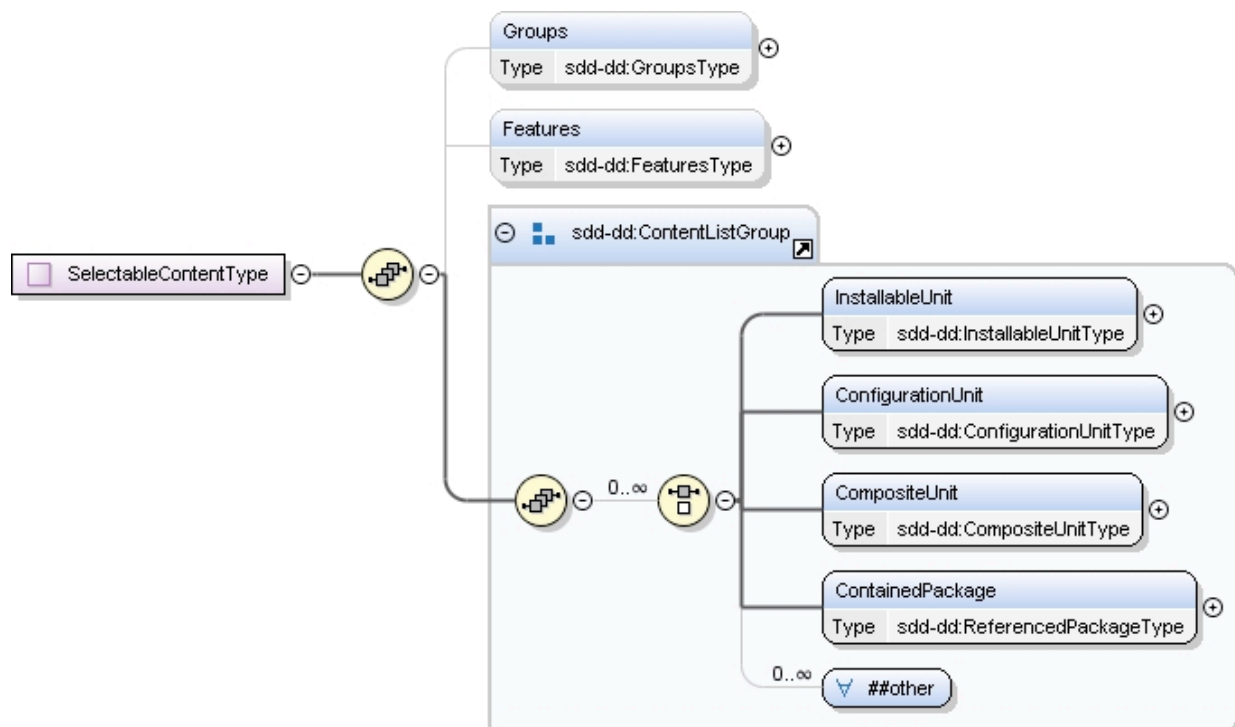

3213
3214 **Figure 89: SelectableContentType structure.**

3215 Content elements defined here make up the selectable content hierarchy. These elements are selected
3216 via *Groups* and *Features* also defined under *SelectableContent*.

3217 ## 4.12.1.1 SelectableContentType Property Summary

Name	Type	*	Description
Groups	GroupsType	0..1	Groups of features that can be selected as a unit.
Features	FeaturesType	0..1	A definition of user-selectable content.
InstallableUnit	InstallableUnitType	0..*	An InstallableUnit that defines selectable content.
ConfigurationUnit	ConfigurationUnitType	0..*	A ConfigurationUnit that defines selectable configuration.
CompositeUnit	CompositeUnitType	0..*	A CompositeUnit that organizes content elements that define selectable content.
ContainedPackage	ReferencedPackageType	0..*	An SDD package whose content is selectable in the context of the aggregating SDD.
	xsd:any	0..*	

3218 ## 4.12.1.2 SelectableContentType Property Usage Notes

3219 ▪ **Groups**: *Groups* can be used by the SDD author to define a convenient way for deployers to select a
3220   group of features.

3221   "Typical" and "Custom" are examples of groups that are commonly presented in installation
3222   interfaces.

3223   See the *GroupsType* section for structure and additional usage details [4.12.2].

3224 ▪ **Features**: *Features* can be used to organize optional functionality into meaningful selections.
3225   *Features* should be meaningful from the deployer's point of view.

3226   See the *FeaturesType* section for structure and additional usage details [4.12.4].

3227 ▪ **InstallableUnit**: See the *InstallableUnitType* section for structure and additional usage details [4.3.1].

3228 ▪ **ConfigurationUnit**: See the *ConfigurationUnitType* section for structure and additional usage details
3229   [4.3.2].

3230 ▪ **CompositeUnit**: See the *CompositeUnitType* section for structure and additional usage details
3231   [4.9.2].

3232 ▪ **ContainedPackage**: See the *ReferencedPackageType* section for structure and additional usage
3233   details [4.10.1].

4.12.2 GroupsType
3234
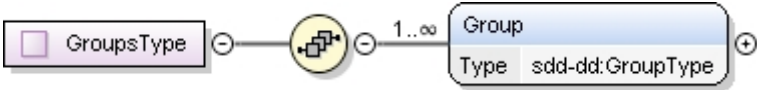
3235

3236 **Figure 90: Groups structure.**

3237 *GroupsType* is used in *SelectableContent* to provide a list of one or more *Group* elements.

4.12.2.1 GroupsType Property Summary
3238

Name	Type	*	Description
Group	GroupType	1..*	A group of features that can be selected together.

4.12.2.2 GroupsType Property Usage Notes
3239

3240 ▪ **Group**: Associating features in a *Group* is based on the characteristics of the package and the ways
3241   in which the SDD author chooses to expose function variability to the deployer.

3242   One example is a "Typical" group that allows easy selection of the most common grouping of
3243   features, along with a "Custom" group that allows an advanced user to select from among all
3244   features. Another example is a "Client" group that selects features that deploy the client software
3245   for an application, along with a "Server" group that selects features that deploy the server
3246   software for the same application.

3247   If alternative sets of selections are desired, Groups MUST be used to define these sets. Zero or one
3248   set can be selected for any particular deployment.

3249   See the *GroupType* section for structure and additional usage details [4.12.3].
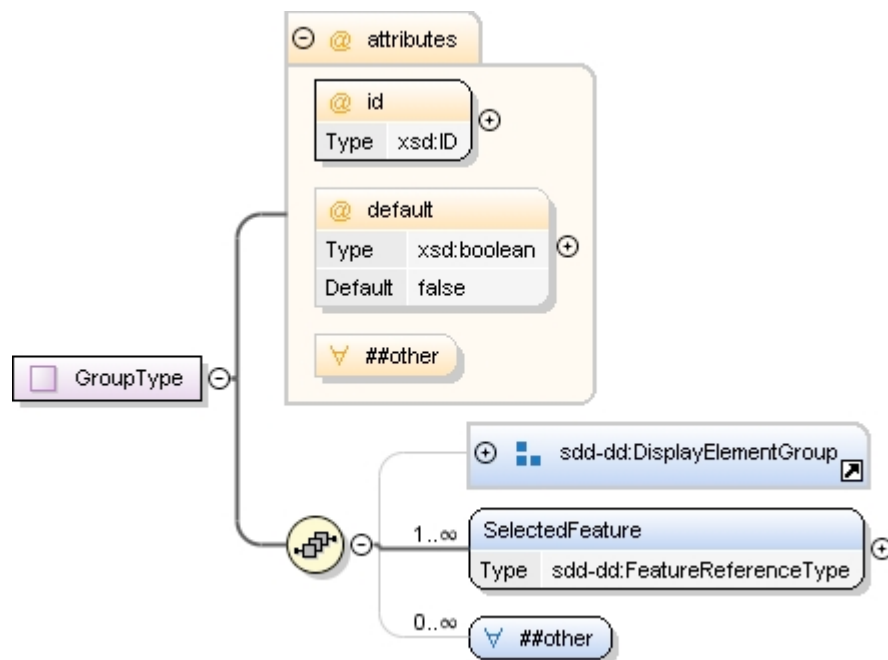
4.12.3 GroupType

3251
Figure 91: GroupType structure.

GroupType provides the type definition for each *Group* element in *SelectableContent's* list of *Groups*. For a particular deployment, zero or one groups may be selected by the deployer.

4.12.3.1 GroupType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	A human-readable name for the group.
Description	DisplayTextType	0..1	A human-readable description of the group.
ShortDescription	DisplayTextType	0..1	A human-readable short description of the group.
SelectedFeature	FeatureReferenceType	1..*	A feature that is part of the group.
	xsd:any	0..*	
id	xsd:ID	1	An identifier of the group that is unique within the descriptor.
default	xsd:boolean	0..1	Indicates that the group is selected by default when no selections are provided by the deployer. **default value="false"
	xsd:anyAttribute	0..*	

4.12.3.2 GroupType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the group.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the group.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

3263 See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

3264 ▪ **SelectedFeature**: Each *SelectedFeature* is considered selected if inputs identify the group as
3265 selected.

3266 Selection of a nested feature causes its parent feature to be selected.

3267 See the *FeatureReferenceType* section for structure and additional usage details [4.12.8].

3268 ▪ **id**: The group's *id* may be used to refer to the group when aggregating the SDD into another SDD.

3269 The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
3270 log and trace messages.

3271 ▪ **default**: Multiple default *Groups* MUST NOT be defined.

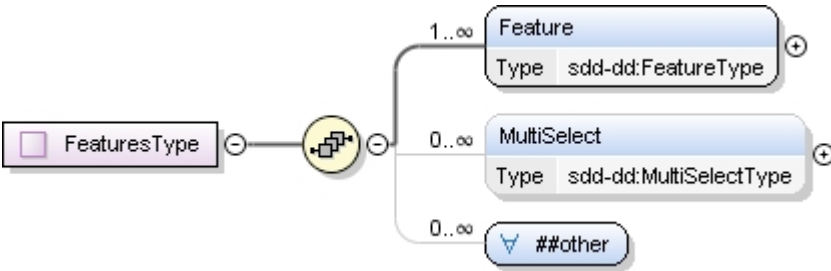## 3272 4.12.4 FeaturesType

3273
3274 Figure 92: FeaturesType structure.

3275 *FeaturesType* provides the type definition for the single, optional, *Features* element in *SelectableContent*.
3276 Features defined directly under the *Features* element in *SelectableContent* are the top level features. A
3277 *Features* element may also include a *MultiSelect* element that refers to features whose selections are
3278 interdependent.

3279 4.12.4.1 FeaturesType Property Summary

Name	Type	*	Description
Feature	FeatureType	1..*	A top level feature in the hierarchy of features defined in SelectableContent.
MultiSelect	MultiSelectType	0..*	A list of feature references whose selection is controlled as a multi-select list with defined minimum and maximum selections.
	xsd:any	0..*	

3280 4.12.4.2 FeaturesType Property Usage Notes

3281 ▪ **Feature**: Each top level *Feature* can define *NestedFeatures*. All features can define required
3282 relationships with other features that cause the required feature to be selected.

3283 See the *FeatureType* section for structure and additional usage details [4.12.5].

3284 ▪ **MultiSelect**: The *MultiSelect* element MUST refer to *Feature* or *NestedFeature* elements.

3285 See the *MultiSelectType* section for structure and additional usage details [4.12.15].
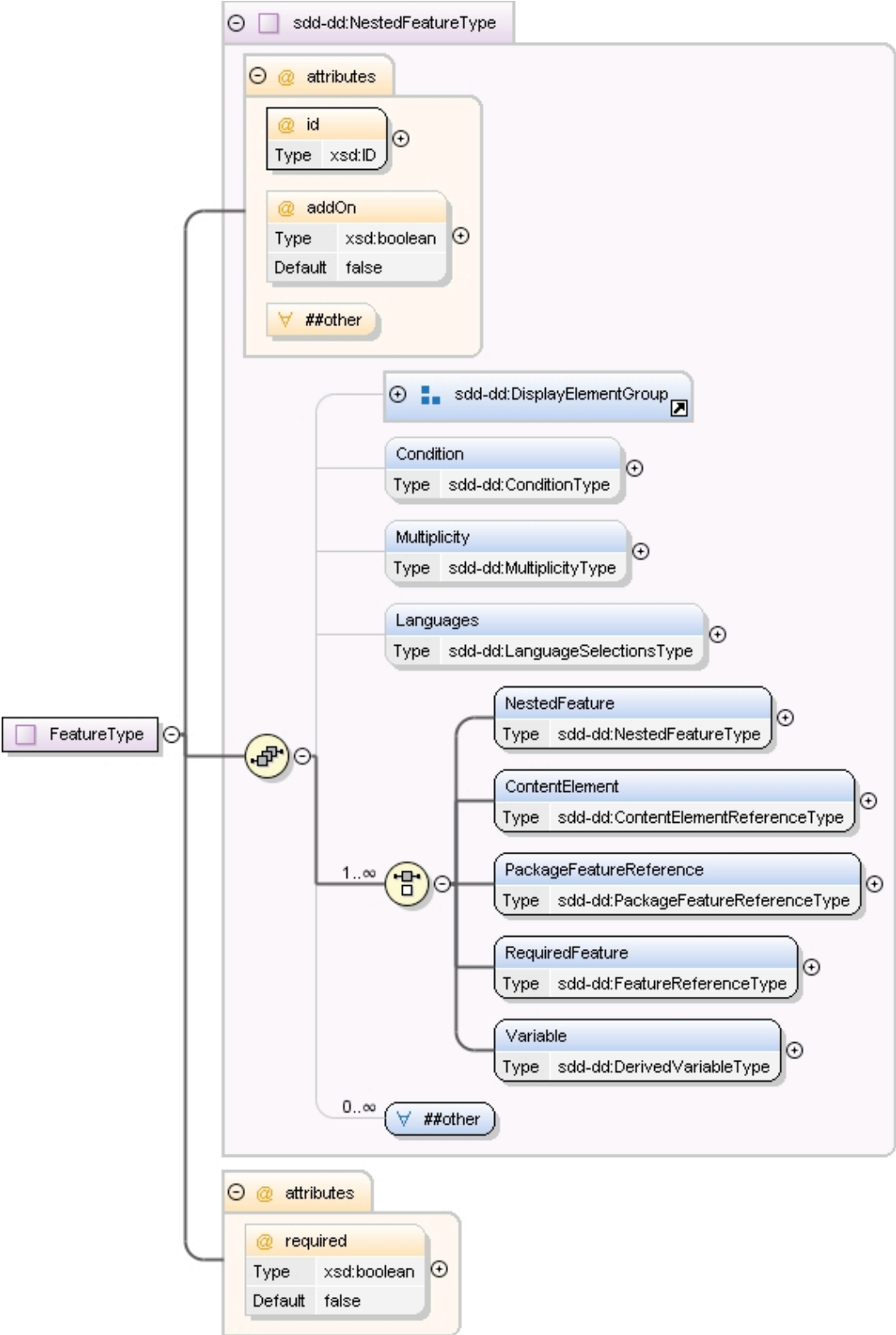
3286 **4.12.5 FeatureType**

3289 *FeatureType* provides the type definition for each feature defined directly below *SelectableContent*. A
3290 *Feature* can define *NestedFeatures* and identify *ContentElements* and other features that will be selected
3291 when the feature is selected. A feature can also be defined to be available for selection only under certain
3292 conditions.

3293 4.12.5.1 FeatureType Property Summary

Name	Type	*	Description
	[extends] NestedFeatureType		See the NestedFeatureType section for additional properties [4.12.6].
required	xsd:boolean	0..1	Indicates the feature must be selected. **default value="false"

3294 4.12.5.2 FeatureType Property Usage Notes

3295 See the *NestedFeatureType* section for details about the inherited attributes and elements [4.12.6].

3296 ▪ **required**: A top level *Feature* MUST be selected when the value of the *required* attribute is "true". In
3297 this case, the user cannot choose to deselect this top level *Feature*.

3298 In *Features* that define *Multiplicity,* the SDD author can state a minimum number of instances of the
3299 *Feature*. This minimum applies only if the *Feature* is selected. The *required* attribute can be used to
3300 indicate that the *Feature* is always selected and so the minimum number of instances applies.

3301 The *required* attribute SHOULD be used only when *Multiplicity* is applied to the *Feature*.

4.12.6 NestedFeatureType
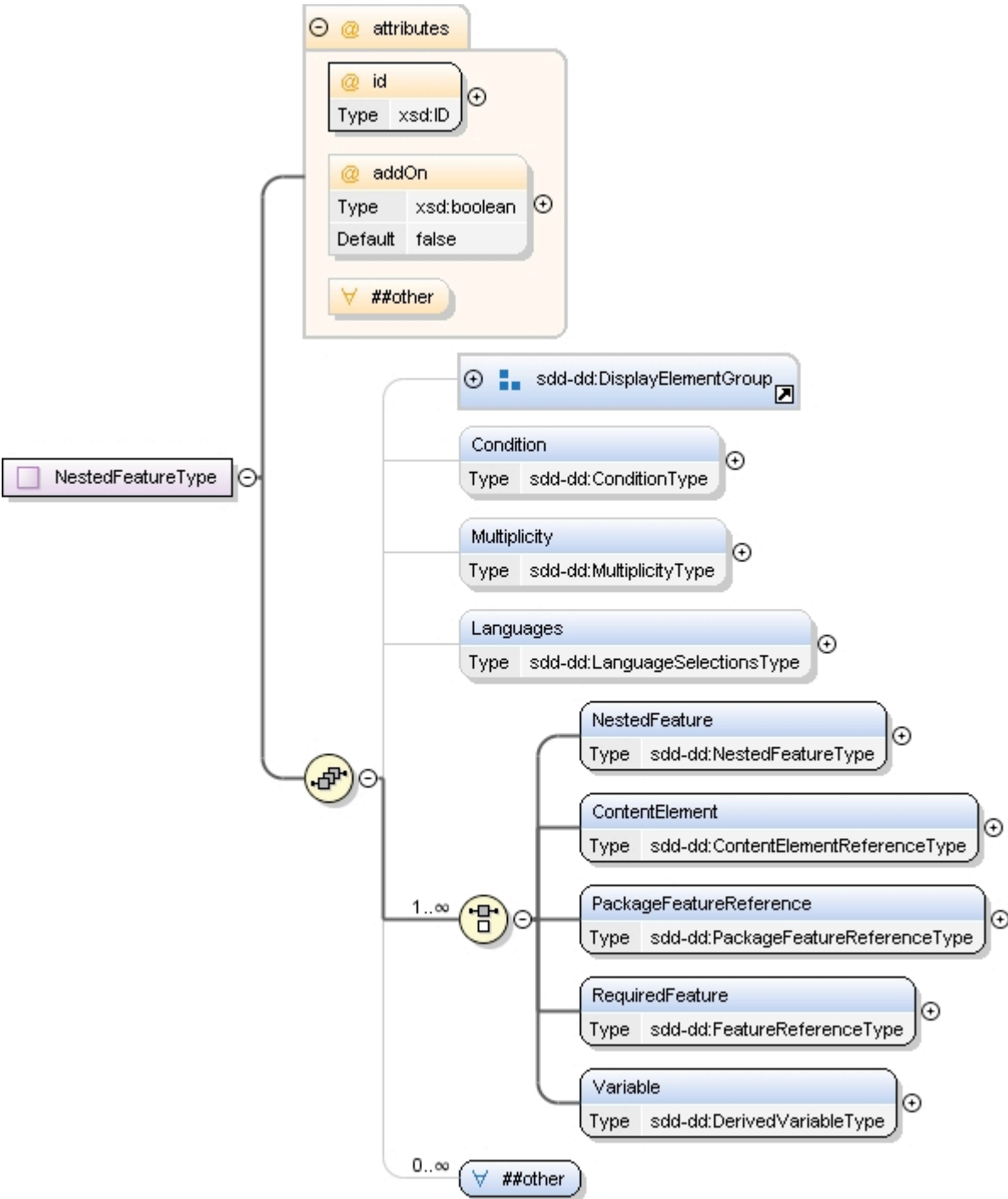
3303
Figure 94: NestedFeatureType structure.

3305 *NestedFeatureType* is identical to *FeatureType* except that *NestedFeatureType* does not define a
3306 *required* attribute. All features other than those defined directly below *SelectableContent* use the
3307 *NestedFeatureType*.

4.12.6.1 NestedFeatureType Property Summary

Name	Type	*	Description

DisplayName	DisplayTextType	0..1	A human-readable name for the feature.
Description	DisplayTextType	0..1	A human-readable description of the feature.
ShortDescription	DisplayTextType	0..1	A human-readable short description of the feature.
Condition	ConditionType	0..1	A condition that determines if the feature is relevant to a particular deployment.
Multiplicity	MultiplicityType	0..1	Both an indication that multiple instances of the feature can be selected and the specification of their constraints.
Languages	LanguageSelectionsType	0..1	A list of language support available for the feature's content.
NestedFeature	NestedFeatureType	0..*	A nested feature.
ContentElement	ContentElementReferenceType	0..*	A reference to a content element to be deployed when the feature is selected.
PackageFeatureReference	PackageFeatureReferenceType	0..*	A reference to a feature to be selected in a ContainedPackage defined in either the BaseContent or SelectableContent hierarchies.
RequiredFeature	FeatureReferenceType	0..*	A reference to a feature that is required when the defining feature is selected and so is selected automatically.
Variable	DerivedVariableType	0..*	The definition of a variable that can be used anywhere in any variable expression in the SDD.
	xsd:any	0..*	
id	xsd:ID	1	Used within the SDD to refer to the feature.
addOn	xsd:boolean	0..1	A "true" value indicates that the feature can be added to a deployed instance of the solution. **default value="false"
	xsd:anyAttribute	0..*	

4.12.6.2 NestedFeatureType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the nested feature.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the nested feature.

  The *Description* element MUST be defined if the *ShortDescription* element is defined.

  See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Condition**: If the features and its nested features are only applicable in certain environments, a *Condition* can be defined. When the *Condition* is not met, the feature and its nested features are not in scope.

  For example, some features may be available only on a Linux operating system, even though the software can be applied on other operating systems. In this case, a *Condition* can be defined to cause the feature to be ignored when the operating system is not Linux.

3323   See the *ConditionType* section for structure and additional usage details [4.5.1].

3324   ▪ **Multiplicity**: When multiple instances of a feature can be selected, a *Multiplicity* element MUST be
3325   defined.

3326   For example, a solution that includes a server and a client may allow the deployment of multiple
3327   clients. In this situation, a feature that defines a *Multiplicity* element would select the content
3328   elements that deploy the client software.

3329   See the *MultiplicityType* section for structure and usage details [4.12.7].

3330   ▪ **Languages**: Sometimes language support for a feature is different than that available for the overall
3331   solution. This is especially likely when features are implemented by aggregation of packages
3332   provided by different teams. When language support differs, the *Languages* element of the feature
3333   MUST be defined to state which languages are supported for the feature.

3334   When *Languages* is defined in a feature, it overrides the global declaration of supported languages
3335   and MUST declare the complete set of language support available for that feature.

3336   If *Languages* is not defined, the global declaration of supported languages in *CompositeInstallable*
3337   applies for the feature.

3338   See the *LanguageSelectionsType* section for structure and additional usage details [4.13.4].

3339   ▪ **NestedFeature**: A *NestedFeature* must be explicitly selected. It is not assumed to be selected when
3340   the parent feature is selected. Selection of a nested feature causes its parent feature to be selected,
3341   but not vice-versa. The definition of a *NestedFeature* indicates that application of the *NestedFeature*
3342   is dependent on application of the parent feature.

3343   ▪ **ContentElement**: The *ContentElement* referred to MUST be in the selectable content hierarchy
3344   defined by the *SelectableContent* element.

3345   When the content reference is to a *CompositeUnit*, the composite and all content elements below it in
3346   the content hierarchy are considered to be in scope when the feature is selected. Ease of referencing
3347   a group of content from a feature can be one reason for using a composite in the content hierarchy.

3348   See the *ContentElementReferenceType* section for structure and additional usage details [4.12.9].

3349   ▪ **PackageFeatureReference**: Selection of a feature may result in selection of an aggregated
3350   package's feature identified by a *ContainedPackage* element anywhere in the *BaseContent* or
3351   *SelectableContent* hierarchies. A *PackageFeatureReference* identifies both the *ContainedPackage*
3352   and the applicable features to be selected in that package.

3353   See the *PackageFeatureReferenceType* section for structure and additional usage details [4.12.10].

3354   ▪ **RequiredFeature**: When the selection of one feature requires the selection of another feature, the
3355   *RequiredFeature* can be used to specify this requirement.

3356   When two features identify each other as required features, they are always selected together.

3357   The selection of the defining feature MUST cause the required feature to be selected.

3358   See the *FeatureReferenceType* section for structure and additional usage details [4.12.8].

3359   ▪ **Variable**: *Variables* defined in features are useful when inputs to an artifact need to vary based on
3360   which features are selected for a particular deployment. Artifact arguments can be defined in terms of
3361   feature *Variables* to allow for this variation. When an artifact deploys selectable content, inputs to the
3362   artifact that indicate the selections for a particular deployment can be associated with feature
3363   selection in the SDD via feature *Variables.*

3364   For example, a *Feature* that deploys a trace facility might define a *Variable* called
3365   "TraceSettings". The value of an argument to a base content artifact might define its value as
3366   "$(TraceSettings)". If the feature is selected, this argument would be used and its value would be
3367   taken from the feature *Variable*. If the feature is not selected, the argument would be ignored.

3368   A *Variable* defined in a feature differs from *Variable* elements defined in content elements in one
3369   important way. A reference to an undefined feature *Variable* is treated as an empty string and is
3370   considered to be defined.

3371   See the *DerivedVariableType* section for structure and additional usage details [4.6.19].

3372 ▪ **id**: Provides the means to reference a feature from other features.

3373 The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
3374 log and trace messages.

3375 ▪ **addOn**: When a solution and the artifacts that deploy the various parts of the solution are designed in
3376 a way that supports the addition of a particular feature at a later time (after the deployment of the
3377 base solution), the *addOn* attribute is set to "true".
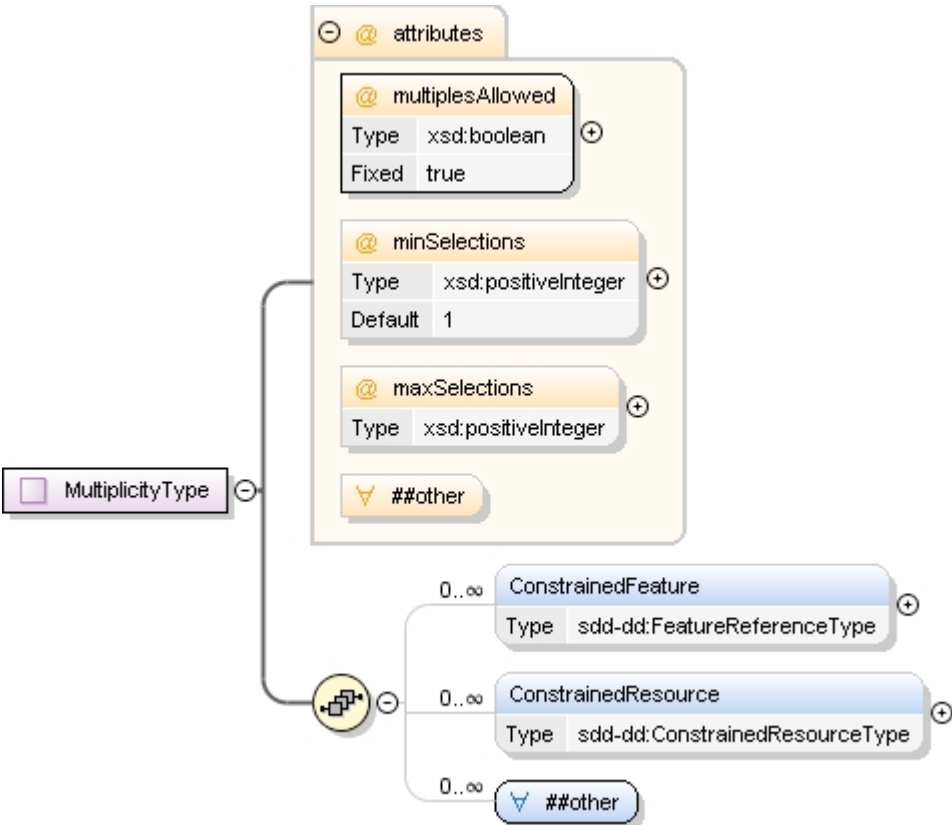
4.12.7 MultiplicityType

3378

3379
3380 **Figure 95: MultiplicityType structure.**

3381 Some solutions allow multiple instances of some portion of the solution's resources to be deployed as
3382 part of the solution.

3383 For example, a solution that includes a server and a client may allow the deployment of multiple
3384 clients. The deployment of each client may involve content elements that represent several different
3385 resulting resources, features that control optional functionality of the client and configuration elements
3386 that configure the client. All of these can be defined within a "Client" feature that declares a *Multiplicity*
3387 element that indicates that multiple clients are allowed. Each selection or "instance" of the feature
3388 results in the deployment of a client.

3389 The phrase "feature instance" is used to refer to the set of instances of all resources deployed when the
3390 feature is selected. It does not imply that features themselves are represented as having lifecycle or that
3391 features in the SDD correspond with feature instances in the deployment environment.

4.12.7.1 MultiplicityType Property Summary

3392

Name	Type	*	Description
ConstrainedFeature	FeatureReferenceType	0..*	A nested feature whose selection must be the same for all instances of the defining feature in a particular deployment.

ConstrainedResource	ConstrainedResourceType	0..*	A resource that must resolve to the same resource instance for all instances of the feature in a particular deployment.
	xsd:any	0..*	
multiplesAllowed	xsd:boolean	1	Indicates that multiple instances of the feature are allowed. **fixed value="true"
minSelections	xsd:positiveInteger	0..1	The minimum number of instances of the feature that must be selected if the feature is selected at all. **default value="1"
maxSelections	xsd:positiveInteger	0..1	That maximum number of instances of the feature that can be selected.
	xsd:anyAttribute	0..*	

4.12.7.2 MultiplicityType Property Usage Notes

- **ConstrainedFeature**: A feature with multiplicity may contain *NestedFeature* elements. When a *NestedFeature* is identified in a *ConstrainedFeature*, then all instances of the defining *Feature* MUST make the same selection choice for that *NestedFeature*.

  See the *FeatureReferenceType* section for structure and additional usage details [4.12.8].

- **ConstrainedResource**: The content elements selected by a feature may express constraints on resources. When the resource constraints for each instance of a feature must resolve to the same resource instance, or when all must resolve to unique resource instances, the resource is referred to and the constraint type is identified in the *ConstrainedResource* element.

  See the *ConstrainedResourceType* section for structure and additional usage details [4.12.11].

- **multiplesAllowed**: This is an attribute with a fixed value of "true". It is included because all other elements and attributes of *MultiplicityType* are optional. A feature that allows multiples but has no need to define constraints on resources, features or number of instances would define a *Multiplicity* element that had only the *multiplesAllowed* attribute.

- **minSelections**: When a feature is selected, if more than one instance of the feature is required, *minSelections* MUST be specified.

- **maxSelections**: When a feature is selected, if there is a limit on the number of instances of the feature that can be selected, *maxSelections* MUST be specified. If *maxSelections* is defined, it MUST be equal to or greater than *minSelections*.
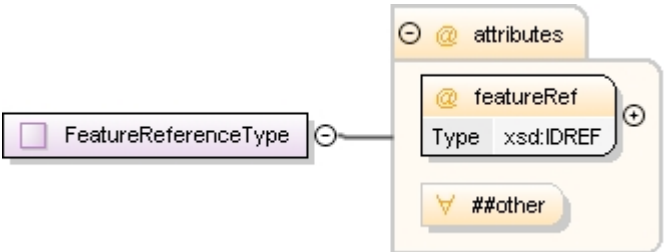
4.12.8 FeatureReferenceType

Figure 96: FeatureReferenceType structure.

FeatureReferenceType provides a way to reference a feature defined in the SDD from within the SDD.

4.12.8.1 FeatureReferenceType Property Summary

Name	Type	*	Description

featureRef	xsd:IDREF	1	Reference to a feature defined in the deployment descriptor.
	xsd:anyAttribute	0..*	

4.12.8.2 FeatureReferenceType Property Usage Notes

▪ **featureRef**: The value MUST reference the *id* of a feature in the deployment descriptor.

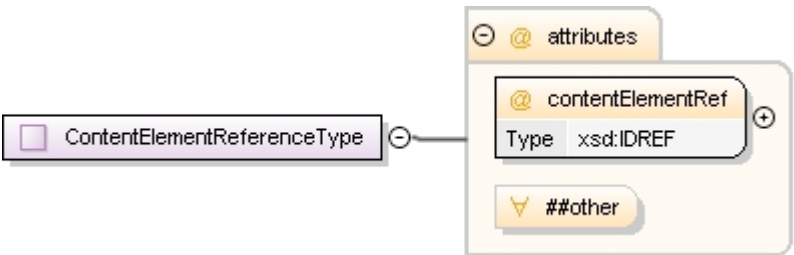## 4.12.9 ContentElementReferenceType

Figure 97: ContentElementReferenceType structure.

ContentElementReferenceType provides a way to reference a content element defined in the SDD from within a feature.

4.12.9.1 ContentElementReferenceType Property Summary

Name	Type	*	Description
contentElementRef	xsd:IDREF	1	Reference to a content element in the deployment descriptor's selectable content.
	xsd:anyAttribute	0..*	

4.12.9.2 ContentElementReferenceType Property Usage Notes

▪ **contentElementRef**: The value MUST reference the *id* of a content element in the deployment descriptor.
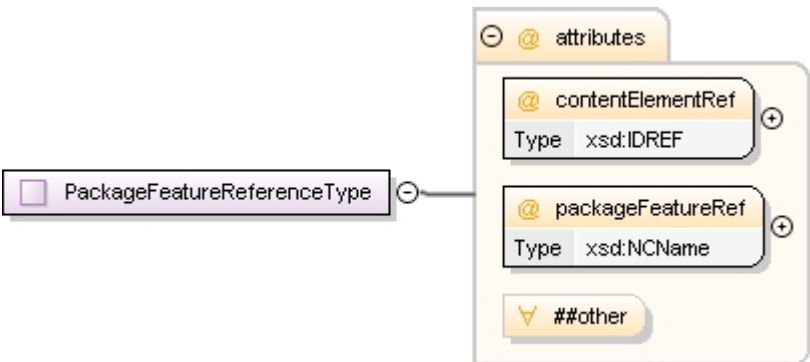
4.12.10 PackageFeatureReferenceType

Figure 98: PackageFeatureReferenceType structure.

PackageFeatureReferenceType provides a way to reference a feature defined in a referenced SDD. It identifies the *ContainedPackage* element that references the SDD and the feature in the referenced SDD.

4.12.10.1 PackageFeatureReferenceType Property Summary

Name	Type	*	Description
contentElementRef	xsd:IDREF	1	Reference to a content element in the deployment descriptor.
packageFeatureRef	xsd:NCName	1	The feature's id as defined in the referenced package's deployment descriptor.
	xsd:anyAttribute	0..*	

4.12.10.2 PackageFeatureReferenceType Property Usage Notes

- **contentElementRef**: This value MUST reference the *id* of a *ContainedPackage* element in *SelectableContent* or *BaseContent.* This reference does not cause the *ContainedPackage* to be in scope.

- **packageFeatureRef**: Specifies the value of the *id* of a feature element from the SDD of the *ContainedPackage* identified in *contentElementRef*. This feature reference is ignored when the *ContainedPackage* identified in *contentElementRef* is not in scope for a particular deployment.

4.12.11 ConstrainedResourceType
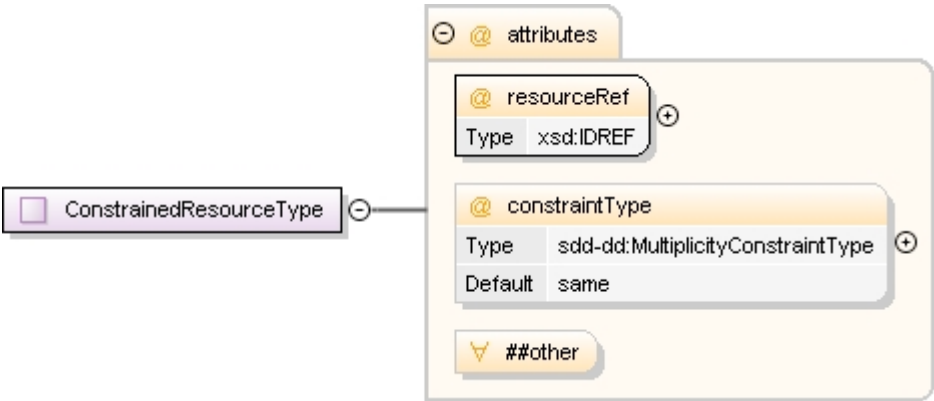
Figure 99: ConstrainedResourceType structure.

A resource may be required during deployment of the content selected by a *Feature* instance. The requirement may exist because the resource is used in a *Requirement* statement, referred to in a *Variable* whose value is in scope for the particular deployment or referred to in a constraint in a *Condition* that is satisfied for the particular deployment. This is an in-scope, required resource for the particular deployment. The SDD author may wish to constrain in-scope, required resources to resolve to the same resource instance for all *Feature* instances or to resolve to unique resource instances for each *Feature* instance. This is done using a *ConstrainedResource* element.

4.12.11.1 ConstrainedResourceType Property Summary

Name	Type	*	Description
resourceRef	xsd:IDREF	1	A reference to the constrained resource.
constraintType	MultiplicityConstraintType	0..1	Indicates whether the constraint requires every instance of the resource to be the same or requires every instance to be different. **default value="same"
	xsd:anyAttribute	0..*	

4.12.11.2 ConstrainedResourceType Property Usage Notes

- **resourceRef**: The value MUST reference the *id* of a resource element in *Topology*.

- **constraintType**: If there is a constraint, *constraintType* indicates that all resource instances be unique or that all resource instances be the same.

    For example, all clients for a particular solution may need to connect to the same database. In this case, *constraintType* would be set to *same*. In other cases, each of the deployed resources might need to use its own unique instance of a required resource. If there could be only one client per operating system, a constraint on the operating system resource would set *constraintType* to *unique*.

    See the *MultiplicityConstraintType* section for the enumeration values for *constraintType* [4.12.12].

4.12.12 MultiplicityConstraintType

This is a simple type that is used to indicate how resources declared in the *Multiplicity* element should be treated. Enumeration values are *same*, *unique*, or if a value is not specified, the SDD author is indicating that it doesn't matter.

4.12.12.1 MultiplicityConstraintType Property Usage Notes

- **same**: The value *same* is used to indicate that the constraint requires all resource instances MUST be the same.

- **unique**: The value *unique* is used to indicate that each resource instance MUST be unique.

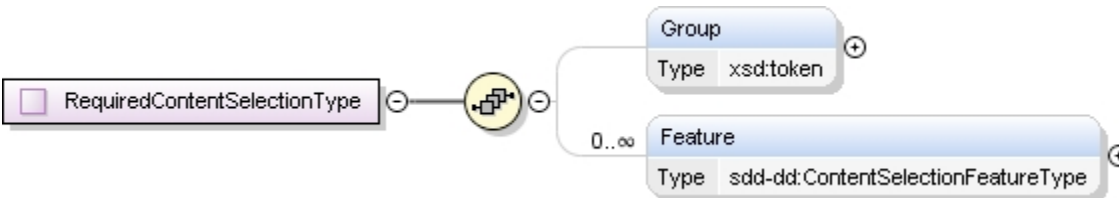## 4.12.13 RequiredContentSelectionType

Figure 100: RequiredContentSelectionType structure.

When one SDD aggregates another, there needs to be an indication of which *Groups* and/or *Features* in the aggregated SDD should be selected. The *RequiredContentSelection* of the referenced package element identifies which elements MUST be selected when the defining package is selected.

4.12.13.1 RequiredContentSelectionType Property Summary

Name	Type	*	Description
Group	xsd:token	0..1	A reference to the group to be selected.
Feature	ContentSelectionFeatureType	0..*	A reference to a feature to be selected.

4.12.13.2 RequiredContentSelectionType Property Usage Notes

- **Group**: The *Group* value is the identifier of a *Group* in the aggregated SDD. This value MUST reference the *id* of a *Group* element in the deployment descriptor denoted by the referenced package.

- **Feature**: The *Feature* element value is the identifier of the feature in the aggregated SDD. Attributes indicating the number of selections to be made can be included. The feature value MUST be the *id* of a feature element in the deployment descriptor denoted by the referenced package.

    If *Group* is also defined, *Feature* SHOULD be a feature that is not selected by the *Group*.

    See the *ContentSelectionFeatureType* section for structure and additional usage details [4.12.14].

4.12.14 ContentSelectionFeatureType

3486

Figure 101: ContentSelectionFeatureType structure.

3488 The *ContentSelectionFeatureType* allows for the definition of the number of times a feature can be
3489 referenced if that feature includes a *Multiplicity* element.

3490 For example, a software package has a server and client; the server can be deployed only on one
3491 machine, but the client can be deployed on multiple machines and configured to reference the one
3492 server. The server, for performance reasons, is limited to 10 client connections. To limit the number of
3493 times the client can be deployed, the *Selections* element expression should be set to "10".

4.12.14.1 ContentSelectionFeatureType Property Summary

Name	Type	*	Description
	[extends] xsd:token		See the xsd:token definition in **[XSD].**
Selections	ElementValueType	0..1	The number of times a feature with Multiplicity in the referenced package should be deployed.

4.12.14.2 ContentSelectionFeatureType Property Usage Notes

3496 See the `xsd:token` definition in **[XSD]** for inherited attributes and elements.

3497 ▪ **Selections**: The value of *Selections* MUST be, or resolve to, a positive integer that is within the
3498 bounds of the *minSelections* and *maxSelections* attributes defined in the *Multiplicity* element of the
3499 referenced feature.

3500 A *pattern* of wildcard is not supported and MUST NOT be used with the *Selections* element.

3501 See the *ElementValueType* section for structure and additional usage details [4.6.2].
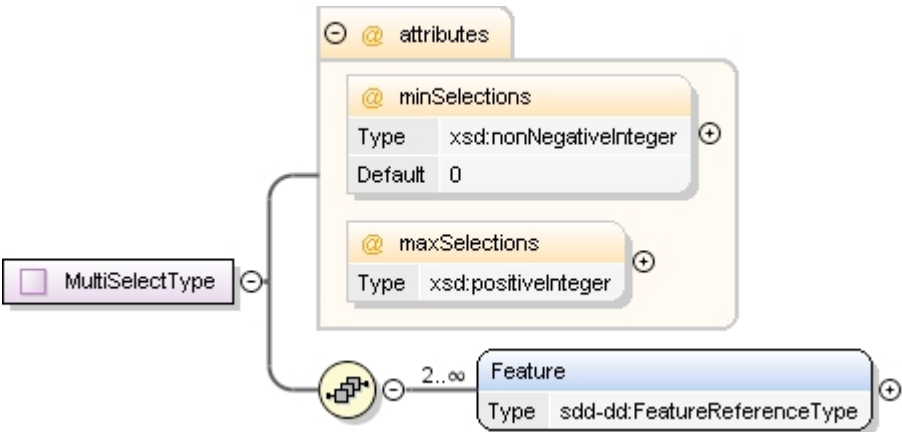
4.12.15 MultiSelectType

3503

Figure 102: MultiSelectType structure.

3505 *MultiSelectType* defines a way to associate features with a defined minimum and maximum number of
3506 selections allowed. A *MultiSelect* element MAY be used to support identification of mutually exclusive
3507 features.

4.12.15.1 MultiSelectType Property Summary

Name	Type	*	Description
Feature	FeatureReferenceType	2..*	A reference to a feature in the list of features defined in the MultiSelect element.
minSelections	xsd:nonNegativeInteger	0..1	Minimum number of features that must be selected. **default value="0"
maxSelections	xsd:positiveInteger	0..1	Maximum number of features that can be selected.

4.12.15.2 MultiSelectType Property Usage Notes

- **Feature**: The value MUST reference the *id* of a feature element.

  See the *FeatureReferenceType* section for structure and additional usage details [4.12.8].

- **minSelections, maxSelections**: When it is not necessary that any of the features in the *MultiSelect* list be selected, the default of "0" can be used.

  Mutually exclusive features can be defined using a *MultiSelect* element with two features, *minSelections* set to "0" and *maxSelections* set to "1".

  If multiple instances of a single feature are selected via multiplicity, the set of multiple instances count only once toward the minimum and maximum. In other words, the count is based solely on the features selected, not on how many instances of each feature are selected.

  When *maxSelections* is not defined, all of the features in the *MultiSelect* MAY be selected for a particular deployment.

  If defined, the *maxSelections* value MUST be greater than or equal to the *minSelections* value and MUST be less than or equal to the number of referenced features.

4.13 Localization

Localization refers to enabling a particular piece of software to support one or more languages. Anything that needs to be deployed to provide support for a particular language in that software is considered localization content. Translated materials are a primary, but not the only, example of localization content.

Localization content is similar in many ways to other content, but there are important differences in how localization content is selected for deployment that lead to the need for a separate content hierarchy and separate types. Two criteria determine whether or not localization content is in scope for a particular deployment:

- The first criterion has to do with the language or languages supported by the localization content. At least one of the languages must be in scope for the content to be selected.

- The second criterion has to do with the availability of the resources to be localized–the localization base. The localization base may be a resource deployed by base or selectable content, or it may be a resource previously deployed and found in the deployment environment.

The types described in this section support definition of metadata describing the criteria for determining when localization content is in scope.
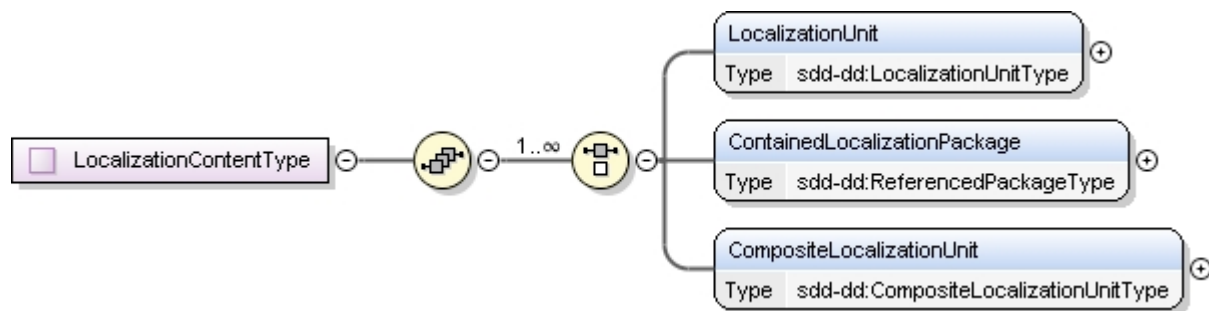
4.13.1 LocalizationContentType

Figure 103: LocalizationContentType structure.

The *LocalizationContent* tree contains all content created specifically to provide localization by deploying language-specific materials for a particular location. The localization support provided can be for content defined in the SDD or it can be for resources in the deployment environment that are not created or modified by deployment of the SDD. Each element defined in the *LocalizationContent* hierarchy is in scope for a particular deployment when it supports a language that is in scope for that deployment and when its localization base, if any, is available.

### 4.13.1.1 LocalizationContentType Property Summary

Name	Type	*	Description
LocalizationUnit	LocalizationUnitType	0..*	Contains artifacts that create, modify or delete language support.
ContainedLocalizationPackage	ReferencedPackageType	0..*	Identifies an SDD whose contents are aggregated to create, modify or delete language support.
CompositeLocalizationUnit	CompositeLocalizationUnitType	0..*	An organizational element that groups localization content and defines metadata common to all the grouped content.

4.13.1.2 LocalizationContentType Property Usage Notes

- **LocalizationUnit**: When there is no need to group a *LocalizationUnit* with other units that have common metadata, the *LocalizationUnit* is defined at the top level of the hierarchy. A *LocalizationUnit* defined at the top level of the *LocalizationContent* hierarchy is in scope for a particular deployment when its *Condition* and *LocalizationBase*, if any, evaluate to true and its *Languages* element, if any, defines a language that is in scope for the deployment.

  See the *LocalizationUnitType* section for structure and additional usage details [4.13.2].

- **ContainedLocalizationPackage**: *ContainedLocalizationPackage* definitions include a list of languages supported by the contained package. The package need not be processed if none of those languages is in scope for a particular deployment.

  See the *ReferencedPackageType* section for structure and additional usage details [4.10.1].

- **CompositeLocalizationUnit**: *CompositeLocalizationUnit* is a construct that allows organization of localization content in a way that is meaningful to the SDD author.

  One example use of a *CompositeLocalizationUnit* is to group a set of *LocalizationUnits* that provide support for a variety of languages for the same resource. This eliminates the need to define identical *LocalizationBase* elements in every *LocalizationUnit*. It can be defined once in the *CompositeLocalizationUnit*.

  If evaluation of the *CompositeLocalizationUnit's Condition*, *Languages* and *LocalizationBase* determines that it is not selected for deployment, none of the content elements defined below it in the hierarchy are selected.

3568   *Requirements*, *Variables*, *Conditions* and *Completion* elements common to all child content elements
3569   MAY be defined once in the *CompositeLocalizationUnit* rather than once in each nested element.

3570   See the *CompositeLocalizationUnitType* section for structure and additional usage details [4.13.3].

3571   4.13.2 LocalizationUnitType
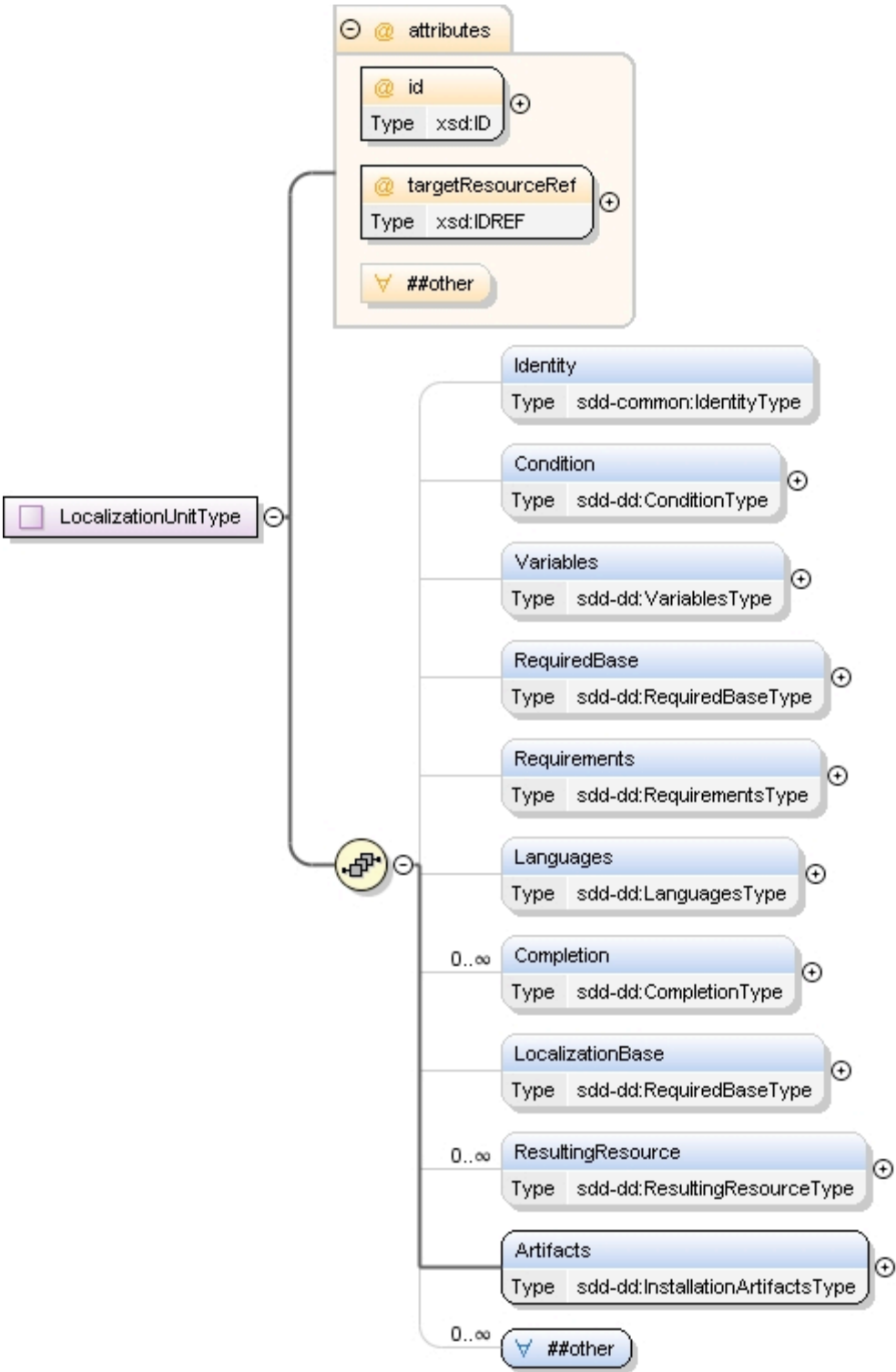
3572
3573   Figure 104: LocalizationUnitType structure.

3574 The *LocalizationUnit* element defines artifacts that deploy localization content for one group of resources
3575 whose translations are packaged together. Localization content consists of materials that have been
3576 translated into one or more languages.

4.13.2.1 LocalizationUnitType Property Summary

3577

Name	Type	*	Description
Identity	IdentityType	0..1	Human-understandable identity information about the LocalizationUnit.
Condition	ConditionType	0..1	A condition that determines if the content element is relevant to a particular deployment.
Variables	VariablesType	0..1	Variables that can be referenced in the LocalizationUnit's requirement and artifact definitions.
RequiredBase	RequiredBaseType	0..1	A resource that will be updated when the LocalizationUnit's UpdateArtifact is processed.
Requirements	RequirementsType	0..1	Requirements that must be met prior to successful processing of the LocalizationUnit's artifacts.
Languages	LanguagesType	0..1	The LocalizationUnit's artifacts contain materials translated into these languages.
Completion	CompletionType	0..*	Describes completion actions such as restart and the conditions under which the action is applied.
LocalizationBase	RequiredBaseType	0..1	A resource whose translatable characteristics will be localized by processing the LocalizationUnit's InstallArtifact.
ResultingResource	ResultingResourceType	0..*	A resource that will be installed or updated by processing the LocalizationUnit's artifacts.
Artifacts	InstallationArtifactsType	1	The set of artifacts associated with the LocalizationUnit.
	xsd:any	0..*	
id	xsd:ID	1	An identifier for the LocalizationUnit scoped to the deployment descriptor.
targetResourceRef	xsd:IDREF	1	Reference to the resource that can process the LocalizationUnit's artifacts.
	xsd:anyAttribute	0..*	

4.13.2.2 LocalizationUnitType Property Usage Notes

3578

3579 ▪ **Identity**: The *Identity* element defines human-understandable information that reflects the identity of
3580 the provided localization resources as understood by the end user of the solution. *Identity* has
3581 elements that are common with elements in the corresponding *PackageDescriptor's PackageIdentity*
3582 element, for example, *Name* and *Version*. The values of these common elements SHOULD be the
3583 same as the corresponding *PackageIdentity* element values.

3584 See the *IdentityType* section for structure and additional usage details [3.4].

3585 ▪ **Condition**: A *Condition* is used when the *LocalizationUnit's* content should be deployed only when
3586 certain conditions exist in the deployment environment.

3587 For example, for a package that has one artifact that should be processed when the operating
3588 system is Linux and another artifact that should be processed when the operating system is

3589   Windows, the *LocalizationUnit* defining metadata for the Linux artifact would have a condition on
3590   the operating system being Linux. The *LocalizationUnit* defining metadata for the Windows
3591   artifact would have a condition on the operating system being Windows.

3592   *Conditions* should not be used to identify the resource that will be localized by the *LocalizationUnit*.
3593   The *LocalizationBase* element is used for that purpose. A *LocalizationUnit* can have both a *Condition*
3594   and a *LocalizationBase*.

3595   See the *ConditionType* section for structure and additional usage details [4.5.1].

3596   ▪ **Variables**: A *Variables* element defines variables that can be used in the definition of requirements
3597   and artifact parameters.

3598   When the deployment descriptor defines a single *LocalizationUnit* at the top level, that is, not inside a
3599   *CompositeInstallable*, the variables it defines can also be referred to in any element under *Topology*.

3600   See the *VariablesType* section for structure and additional usage details [4.6.5].

3601   ▪ **RequiredBase**: *RequiredBase* identifies the resource that must exist prior to applying the
3602   *LocalizationUnit's* update artifact.

3603   See the *RequiredBaseType* section for structure and additional usage details [4.7.8].

3604   ▪ **Requirements**: *Requirements* MUST be met prior to processing the *LocalizationUnit's* artifacts.

3605   See the *RequirementsType* section for structure and additional usage details [4.7.1].

3606   ▪ **Languages**: *Languages* lists the languages of the translated material deployed by the
3607   *LocalizationUnit*.

3608   See the *LanguagesType* section for structure and additional usage details [4.13.6].

3609   ▪ **Completion**: A *Completion* element MUST be included if the artifact being processed requires a
3610   system operation such as a reboot or logoff to occur to function successfully after deployment or if the
3611   artifact executes a system operation to complete deployment of the contents of the artifact.

3612   There MUST be an artifact associated with the operation defined by a *Completion* element.

3613   For example, if there is a *Completion* element for the *install* operation, the *LocalizationUnit* must
3614   define an *InstallArtifact*.

3615   See the *CompletionType* section for structure and additional usage details [4.3.14].

3616   ▪ **LocalizationBase**: *LocalizationBase* identifies the resource or resources that can be localized by
3617   processing the *LocalizationUnit*. A resource that satisfies the constraints defined in the
3618   *LocalizationBase* is one that can be localized by applying the *LocalizationUnit*.

3619   If no resource is found that meets the constraints defined in *LocalizationBase* during a particular
3620   deployment, then the *LocalizationUnit* is not considered to be in scope for that deployment. This does
3621   not represent an error.

3622   Translations created or modified by the *LocalizationUnit* are for human-readable text included with the
3623   *LocalizationBase* resources.

3624   See the *RequiredBaseType* section for structure and additional usage details [4.7.8].

3625   ▪ **ResultingResource**: The *ResultingResources* for a *LocalizationUnit* MUST NOT identify resources
3626   other than localization resources.

3627   See the *ResultingResourceType* section for structure and additional usage details [4.8.1].

3628   ▪ **Artifacts**: When the *LocalizationUnit* is a singleton defined outside of a *CompositeInstallable*, it
3629   MUST define at least one artifact element and MAY define one of each type of artifact element
3630   allowed for its type. The inclusion of an artifact element in a singleton *LocalizationUnit* implies support
3631   for the associated operation.

3632   When the *LocalizationUnit* is defined within a *CompositeInstallable*, it MUST define exactly one
3633   artifact. The artifact defined MAY be any artifact allowed in a *LocalizationUnit* and it MUST support
3634   the single top level *operation* defined by the *CompositeInstallable*. This does not mean the operation
3635   associated with the artifact has to be the same as the one defined by the *CompositeInstallable*.

3636   For example, an install of a localization resource may be required during the update of the overall
3637   solution, in which case the *LocalizationUnit* would define an *InstallArtifact* to support the top level
3638   update operation.

3639   See the *InstallationArtifactsType* section for structure and additional usage details [4.3.4].

3640   ▪ **id**: The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
3641   log and trace messages.

3642   ▪ **targetResourceRef**: The *targetResourceRef* attribute MUST reference the *id* of a resource element
3643   in *Topology* that will process the *LocalizationUnit's* artifacts to create or modify the localization
3644   resources identified in the *LocalizationUnit's ResultingResource* elements.

4.13.3 CompositeLocalizationUnitType
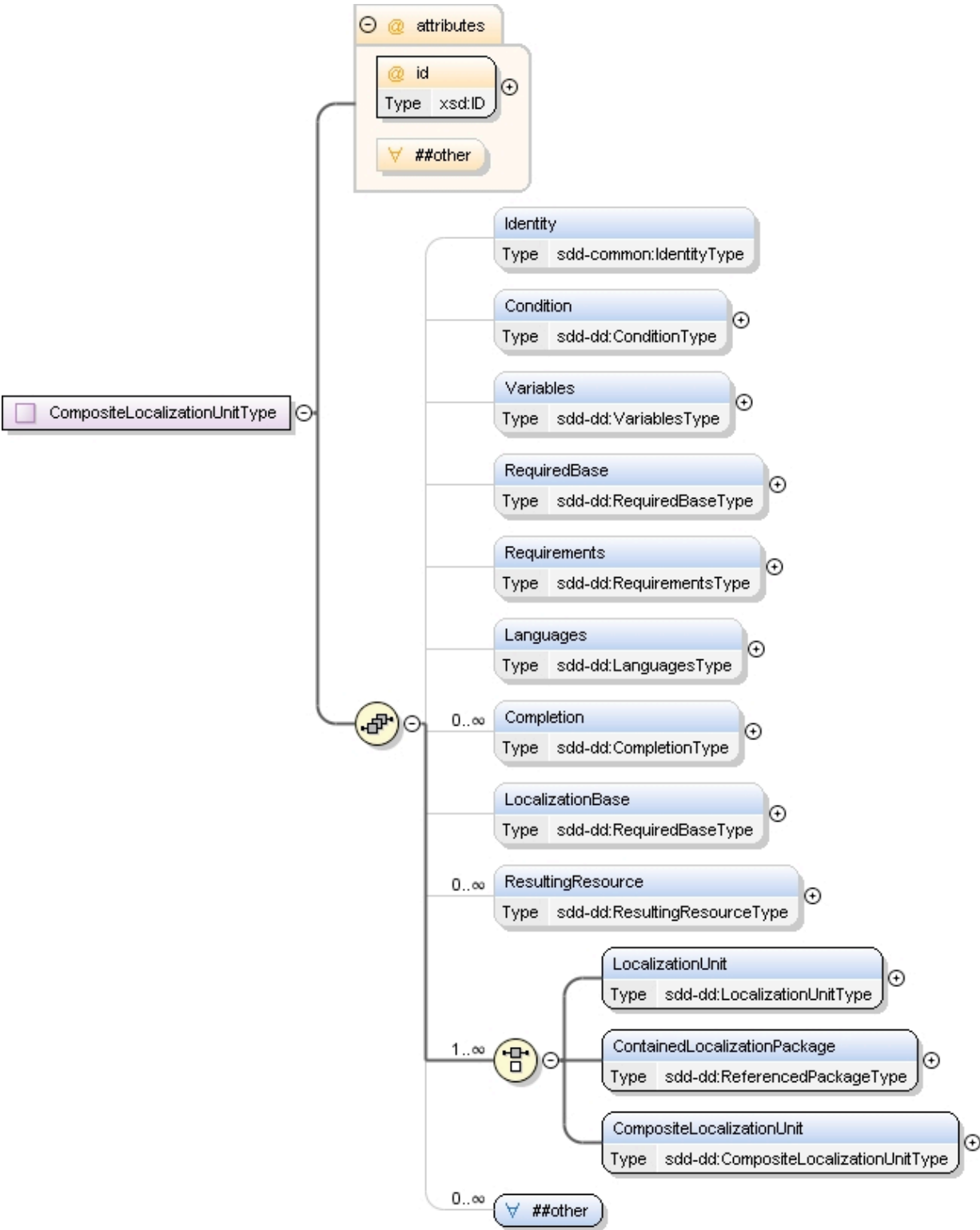
3646

Figure 105: CompositeLocalizationUnitType structure

CompositeLocalizationUnitType provides the type definition for all *CompositeLocalizationUnit* elements in
the *LocalizationContent* hierarchy. *CompositeLocalizationUnit* elements define nested localization content
elements and metadata that applies to all of the nested elements.

4.13.3.1 CompositeLocalizationUnitType Property Summary

Name	Type	*	Description
Identity	IdentityType	0..1	Human-understandable identity information about the CompositeLocalizationUnit.
Condition	ConditionType	0..1	A condition that determines if the CompositeLocalizationUnit is relevant to a particular deployment.
Variables	VariablesType	0..1	Variables for use within the CompositeLocalizationUnit and content elements nested beneath it in the hierarchy.
RequiredBase	RequiredBaseType	0..1	A resource that will be updated when the nested elements are processed.
Requirements	RequirementsType	0..1	Requirements that must be met prior to successful processing of the nested content elements.
Languages	LanguagesType	0..1	Localization elements defined within CompositeLocalizationUnit contain materials translated into these languages.
Completion	CompletionType	0..*	Describes completion actions such as restart and the conditions under which the action is applied.
LocalizationBase	RequiredBaseType	0..1	A resource whose translatable characteristics will be localized by processing the nested content elements.
ResultingResource	ResultingResourceType	0..*	A localization resource that will be installed or updated by processing the nested content elements.
LocalizationUnit	LocalizationUnitType	0..*	Contains artifacts that will create, modify or delete language support.
ContainedLocalizationPackage	ReferencedPackageType	0..*	Identifies an SDD whose contents are aggregated to create, modify or delete language support.
CompositeLocalizationUnit	CompositeLocalizationUnitType	0..*	An organizational element that groups localization content and defines metadata common to all the grouped content.
	xsd:any	0..*	
id	xsd:ID	1	An identifier for the CompositeLocalizationUnit that is unique within the deployment descriptor.
	xsd:anyAttribute	0..*	

4.13.3.2 CompositeLocalizationUnitType Property Usage Notes

3653 ▪ **Identity**: The *CompositeLocalizationUnit*, like all content elements, is a unit of packaging. Its identity
3654 is the identity of a unit of packaging and may be useful to package management tools. The identity
3655 MAY be similar or identical to the identity of the *ResultingResource(s)*.

3656   See the *IdentityType* section for structure and additional usage details [3.4].

3657 ▪ **Condition**: If the composite and the elements nested beneath it are applicable only in certain
3658   environments, a *Condition* can be defined. When the *Condition* is not met, the composite and its
3659   nested elements are not in scope.

3660   See the *ConditionType* section for structure and additional usage details [4.5.1].

3661 ▪ **Variables**: *Variables* used by more than one nested element can be defined in the
3662   *CompositeLocalizationUnit* for efficiency both in composing and processing the SDD. *Variables* are
3663   visible to all nested content elements.

3664   See the *VariablesType* section for structure and additional usage details [4.6.5].

3665 ▪ **RequiredBase**: If the processing of all the update artifacts in the nested content elements results in a
3666   single resource being updated, that resource can be defined in the *CompositeLocalizationUnit's*
3667   *RequiredBase* element.

3668   See the *RequiredBaseType* section for structure and additional usage details [4.7.8].

3669 ▪ **Requirements**: When a *CompositeLocalizationUnit* is in scope for a particular deployment–as
3670   determined by evaluation of its *LocalizationBase* and *Languages* properties–then its requirements
3671   MUST be met.

3672   See the *RequirementsType* section for structure and additional usage details [4.7.1].

3673 ▪ **Languages**: The *Languages* element in the *CompositeLocalizationUnit* MUST NOT be defined or
3674   MUST define the union of all languages supported by the nested content elements. For nested
3675   content elements to be evaluated to determine if they are in scope, the *CompositeLocalizationUnit*
3676   must be in scope. When *Languages* is present in the *CompositeLocalizationUnit*, it must define one of
3677   the languages in scope for the particular deployment if any of the nested elements are to be
3678   evaluated. If *Languages* is not present in a *CompositeLocalizationUnit*, evaluation of all the child
3679   elements still is required, as long as the other elements of *CompositeLocalizationUnit* have evaluated
3680   to true. When the *Languages* and/or the *LocalizationBase* element in a *CompositeLocalizationUnit* is
3681   not defined, the nested content elements must be evaluated to determine if they are in scope.

3682   See the *LanguagesType* section for structure and additional usage details [4.13.6].

3683 ▪ **Completion**: When a particular completion action applies to all nested elements and should be
3684   performed only once for the entire group, it can be defined in the *CompositeLocalizationUnit* rather
3685   than in each individual element.

3686   See the *CompletionType* section for structure and additional usage details [4.3.14].

3687 ▪ **LocalizationBase**: A *LocalizationBase* element evaluates to true when the resource identified in the
3688   base is created by a content element that is in scope for the deployment or it already exists in the
3689   deployment environment.

3690   When the *LocalizationBase* is defined it must evaluate to true for any of the nested content elements
3691   to be evaluated. If it evaluates to false, none of the nested content elements are in scope. If it
3692   evaluates to true, the nested content elements may be in scope.

3693   When the *LocalizationBase* and/or the *Languages* element in a *CompositeLocalizationUnit* is not
3694   defined, the nested content elements must be evaluated to determine if they are in scope.

3695   See the *RequiredBaseType* section for structure and additional usage details [4.7.8].

3696 ▪ **ResultingResource**: If there are one or more resources that will be created when the nested content
3697   elements are processed, they can be defined here.

3698   See the *ResultingResourceType* section for structure and additional usage details [4.8.1].

3699 ▪ **LocalizationUnit**: *LocalizationUnits* defined within the composite typically have common metadata.
3700   Metadata defined in the composite does not need to be repeated in the nested element. Definitions in
3701   the nested *LocalizationUnit* are additions to those defined in the composite.

3702   See the *LocalizationUnitType* section for structure and additional usage details [4.13.2].

3703 ▪ **ContainedLocalizationPackage**: A *ContainedLocalizationPackage* is defined in a
3704   *CompositeLocalizationUnit* for the same reasons that a *LocalizationUnit* is–because it has metadata
3705   in common with other elements defined in the composite.

3706   See the *ReferencedPackageType* section for structure and additional usage details [4.10.1].

3707   ▪ **CompositeLocalizationUnit**: A *CompositeLocalizationUnit* can be nested inside another
3708   *CompositeLocalizationUnit* when some of the metadata is shared only by a subset of the elements
3709   nested in the higher level composite.

3710   For example, the higher level composite might contain operating system requirements that apply
3711   to all localization content and nested composites might group localization content by localization
3712   base.

3713   ▪ **id**: This *id* is not referred to by any other element in the deployment descriptor.

3714   The *id* attribute may be useful to software that processes the SDD, for example, for use in creating
3715   log and trace messages. It also may be useful for associating custom discovery logic with the
3716   *CompositeLocalizationUnit's* resource-related elements.

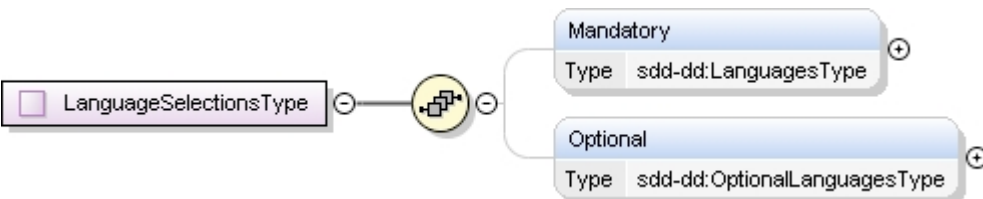## 4.13.4 LanguageSelectionsType

3718
Figure 106: LanguageSelectionsType structure.

3720   *LanguageSelectionsType* provides the type definition for the *Languages* element in *CompositeInstallable*
3721   that describes the languages supported by the SDD as a whole. It also provides the type definition for the
3722   *Languages* element in features that allows a feature to override the SDD-wide definitions.

4.13.4.1 LanguageSelectionsType Property Summary

Name	Type	*	Description
Mandatory	LanguagesType	0..1	The set of languages that will be deployed.
Optional	OptionalLanguagesType	0..1	The set of language selections available to the deployer.

4.13.4.2 LanguageSelectionsType Property Usage Notes

3725   ▪ **Mandatory**: The deployer has no ability to determine if a mandatory language will be deployed.

3726   See the *LanguagesType* section for structure and additional usage details [4.13.6].

3727   ▪ **Optional**: Each language group in the list of optional languages defines a list of one or more
3728   languages that can be selected together.

3729   Language groups defined in *LanguageSelections* MAY be used to allow the deployer to select
3730   individual languages or to allow selection of multiple languages as a single choice.

3731   See the *OptionalLanguagesType* section for structure and additional usage details [4.13.5].
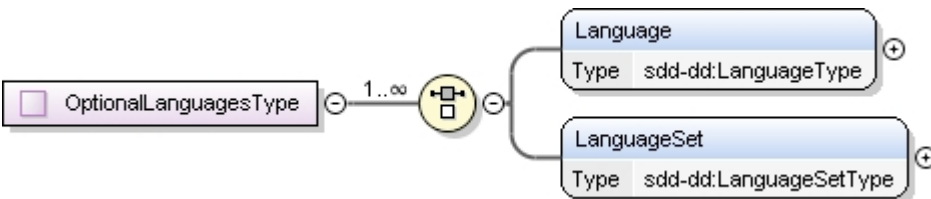
4.13.5 OptionalLanguagesType

3733
Figure 107: OptionalLanguagesType structure

OptionalLanguagesType supports definition of a language or sets of languages that the deployer can optionally choose for deployment. This type is used to define the global set of optional languages in *CompositeInstallable* as well as any *Feature*-specific set that overrides the global set for a particular *Feature*.
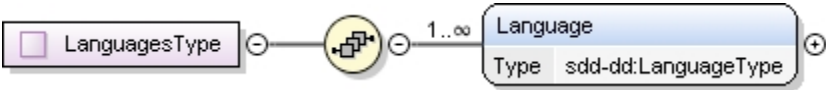
4.13.5.1 OptionalLanguagesType Property Summary

Name	Type	*	Description
Language	LanguageType	1..*	A single language that can be chosen individually.
LanguageSet	LanguageSetType	1..*	A set of languages that can be chosen together.

4.13.5.2 OptionalLanguagesType Property Usage Notes

- **Language**: When the SDD author allows the deployer to individually select a language for deployment, it is defined in a *Language* element within *OptionalLanguages*.

  See the *LanguageType* section for structure and usage details [4.13.7].

- **LanguageSet**: When the SDD author allows the deployer to select languages for deployment as a set, it is defined in a *LanguageSet* element within *OptionalLanguages*.

  One example of a reason to define optional languages in a set rather than individually is for a group of languages that are packaged together and whose deployment cannot be separated.

  See the *LanguageSetType* section for structure and additional usage details [4.13.8].

4.13.6 LanguagesType

Figure 108: LanguagesType structure.

LanguagesType supports expression of a list of languages. It is used in the *Languages* elements of content elements to list languages supported by that content element. It is also used as the type of the *Mandatory* element that lists languages that are deployed by default.

4.13.6.1 LanguagesType Property Summary

Name	Type	*	Description
Language	LanguageType	1..*	A single language definition.

4.13.6.2 LanguagesType Property Usage Notes

- **Language**: Each language definition MAY include display information as well as the language code that identifies the language.

  See the *LanguageType* section for structure and additional usage details [4.13.7].
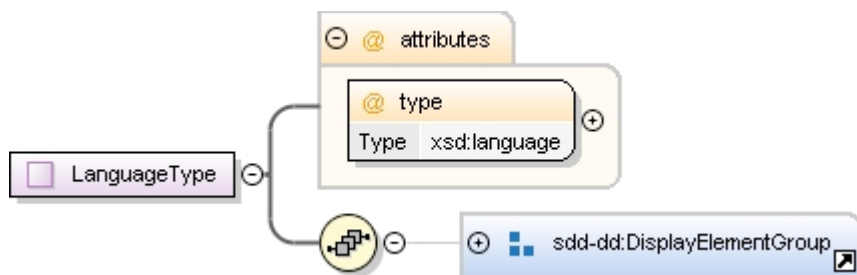
4.13.7 LanguageType

Figure 109: LanguageType structure.

LanguageType supports the definition of display information and the language code for one language. It is used everywhere a language is defined in the SDD.

4.13.7.1 LanguageType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	A name for the language.
Description	DisplayTextType	0..1	A description of the language.
ShortDescription	DisplayTextType	0..1	A short description of the language.
type	xsd:language	1	The locale code for the language.

4.13.7.2 LanguageType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the language.

   See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the language.

   The *Description* element MUST be defined if the *ShortDescription* element is defined.

   See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **type**: The *type* attribute MUST be defined as a value that conforms to the set of language codes defined by **[RFC3066]**.

   For example, "de" is a locale code for German and "en-US" is the locale code for English in the United States.
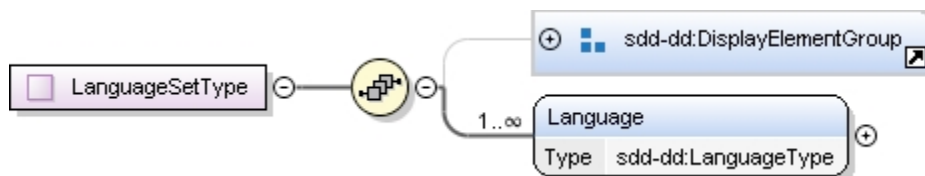
4.13.8 LanguageSetType

Figure 110: LanguageSetType structure.

LanguageSetType provides the type definition for the *OptionalLanguages* elements of *CompositeInstallable* and *Feature*. It defines a set of languages that can be selected together.

4.13.8.1 LanguageSetType Property Summary

Name	Type	*	Description
DisplayName	DisplayTextType	0..1	A name for the set of languages.
Description	DisplayTextType	0..1	A description of the set of languages.
ShortDescription	DisplayTextType	0..1	A short description of the set of languages.
Language	LanguageType	1..*	A set of one or more language codes.

### 4.13.8.2 LanguageSetType Property Usage Notes

- **DisplayName**: This element MAY be used to provide human-understandable information. If used, it MUST provide a label for the set of languages.

    For example, "Eastern European Languages" or "French, English and German".

    See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Description, ShortDescription**: These elements MAY be used to provide human-understandable information. If used, they MUST provide a description of the set of languages.

    The *Description* element MUST be defined if the *ShortDescription* element is defined.

    See the *DisplayElementGroup* section for structure and additional usage details [4.14.2].

- **Language**: The languages defined in this element MUST be selected together.

    See the *LanguageType* section for structure and additional usage details [4.13.7].

4.14 Display Information

There are many places throughout the SDD where translatable information intended for display to humans MAY be defined. All display information definitions can include a *translationKey* that can be used as an index into a file containing translations.

4.14.1 DescriptionGroup
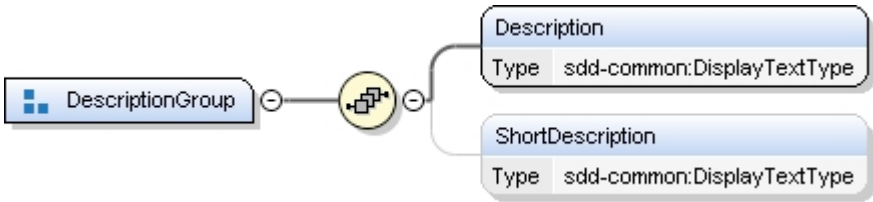
Figure 111: DescriptionGroup structure.

The *DescriptionGroup* type is used throughout the SDD to provide human-readable, translatable, descriptive-text elements.

4.14.1.1 DescriptionGroup Property Usage Notes

- **Description**: This is a description of the defining element unless usage notes for that element state otherwise. It can be as long as necessary to provide a useful description.

    The *Description* element MUST be defined if the *ShortDescription* element is defined.

    See the *DisplayTextType* section for details about associating this text with translated text [4.14.3].

- **ShortDescription**: This is a short description of the defining element unless usage notes for that element state that it refers to something else. It SHOULD provide a limited description that can be used by tools where limited text is allowed, for example, fly-over help.

    See the *DisplayTextType* section for details about associating this text with translated text [4.14.3].
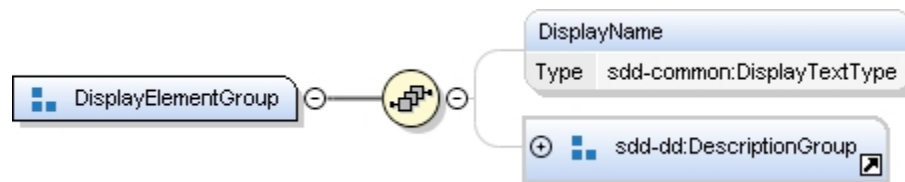
4.14.2 DisplayElementGroup

Figure 112: DisplayElementGroup structure.

The *DisplayElementGroup* is used throughout the package descriptor and deployment descriptor to provide human-readable, translatable names, descriptions and/or short descriptions for a variety of elements.

4.14.2.1 DisplayElementGroup Property Usage Notes

- **DisplayName**: This is a label for the defining element unless usage notes for that element state otherwise.

  See the *DisplayTextType* section for details about associating this text with translated text [4.14.3].
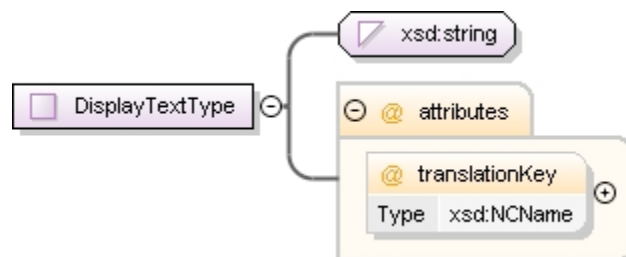
4.14.3 DisplayTextType

Figure 113: DisplayTextType Structure.

Elements of *DisplayTextType* define translatable strings and an optional key to translated text in language bundle files. *DisplayTextType* extends the `xsd:string` type with an optional *translationKey* attribute.

4.14.3.1 DisplayTextType Property Usage Notes

- **translationKey**: The *translationKey* attribute is a value that can be used as an index into a file containing translations of *DisplayTextType* elements in the *DeploymentDescriptor* and/or *PackageDescriptor*. The value of the *translationKey* MUST match an entry in the message bundle referenced by the *descriptorLanguageBundle* attribute in the package descriptor.

  See the *PackageDescriptor* section for details on how to specify the appropriate message bundle in the *descriptorLanguageBundle* attribute [3.1].

5 Conformance

5.1 General Conformance Statements

An implementation MAY claim conformance to the entirety of the SDD specification (including all conformance levels) or one or more particular conformance levels, and/or one or more particular profiles (SDD conformance levels and profiles are detailed next).

5.2 Conformance Levels

An SDD conformance level (CL) is defined, consistent with **[CONFORM]**, as a subset of the schema intended to enable a certain set of capabilities to be achieved, based on SDDs that restrict their content to the particular CL. The purpose of conformance levels is to allow subsets of the full set of capabilities that can be expressed using an SDD to be implemented. The proper subsets are expected to be easier to implement, but still offer features, value and interoperability that make it worthwhile to implement a particular CL in certain circumstances.

SDD conformance levels are designated as CL1 and CL2. CL1 is a proper subset of the schema; CL2 represents the full schema. CL1 is the minimal set or core of the specification that shall be implemented by all products. CL2 includes all of CL1 and consists of the entire specification.

The following sections describe the defined CLs for SDD.

5.2.1 CL Capabilities

Table 1 expresses the capabilities for each defined CL.

	Conformance Level 1	Conformance Level 2
Description	Single target, simple package.	Multi-target, aggregated packages; full deployment capabilities with all functions enabled by the SDD schema.
Objective	Serve as the "on-ramp" for SDD adoption. Deploy pre-prepared content that needs limited customization (basic parameters). Descriptors serve as contract between assembly and operations. Exemplary use case is "wrappering" existing packages in SDD.	Serve as the expected level for newly-authored non-legacy SDDs. Deploy newly-prepared content that has related components in a solution, with various topologies. Most robust specification (and corresponding run-time implementations) of SDD. Exemplary use case is non-trivial, non-legacy solution deployment.
Included Schema Functions	• Solution package with single component (singleton IU, CU, or LU; no composite) and single target topology • Solution package dependency checking for given environment • base installations and maintenance • Simple uninstall (based on information in single descriptor) • Ability to deploy existing artifact formats appropriate for the target	All functions, including: • Aggregation (composites) • Features • Selectable features • Conditional content • Variable-target topology • Robust localization

	environment • Some localization possible (localization of the units that are supplied)	
Excluded Functions	• Features • Selectable content • Requisites • Aggregation • Multi-target topology • Robust localization • Replacements and modifications that change base resource/solution composition (including obsolescence) • Backwards compatibility, range enforcement • Verification of installation and configuration	None

3854 **Table 1: SDD conformance level capabilities summary.**

5.2.2 Conformance Level Differences

3855

3856 CL1 SDDs can be used to describe the inputs, requirements and results of processing a single
3857 deployment artifact. This artifact could be one that deploys, updates, configures or localizes software
3858 resources. This is useful for simple deployments that require only a single artifact. CL2 SDDs add support
3859 for aggregation of multiple artifacts and SDDs into solutions; definition of features that optionally select
3860 content; and requisite software that can be deployed if needed to satisfy requirements. CL1 SDDs can be
3861 aggregated by CL2 SDDs.

3862   For example, CL2 SDDs can describe a solution that consists of a Web server, an application server,
3863   a database and one or more applications, in which each of these components is described by its own
3864   individual SDD and an aggregating CL2 SDD aggregates them into the composite solution.

3865 The differences between CL1 and CL2 that are summarized in Table 1 are detailed next. These make
3866 use of the information that is in the SDD schema; see **[CL1_Schema]** for the CL1 schema files, and
3867 **[CL2_Schema]** for the CL2 schema files. The differences between the CL1 and CL2 schema files are
3868 isolated to the "sdd-dd" namespace. The "sdd-common" and "sdd-pd" namespaces contain identical
3869 schema files for each namespace with respect to CL1 and CL2.

5.2.2.1 Type Definitions Modified in CL2

3870

3871 A few SDD types used in CL1 have additional elements added in CL2. The types listed in the left column
3872 of Table 2 exist in both CL1 and CL2 with different definitions. The elements in the right column are the
3873 sub-elements added to the type definitions in CL2.

3874

Type Name	CL2 Sub-Element Names
DeploymentDescriptorType	Requisites CompositeInstallable
InstallationArtifactsType	RepairArtifact
ResultingResourceType	Relationship
ResultingChangeType	Relationship
ResourceConstraintGroup	UniquenessConstraint

Type Name	CL2 Sub-Element Names
	RelationshipConstraint
ConditionalResourceConstraintType	UniquenessConstraint
	RelationshipConstraint
RequirementType	Dependency
AlternativeRequirementType	Dependency

Table 2. Modified Types.

5.2.2.2 Type Structures Modified in CL2

Several SDD types have altered structure between CL1 and CL2. The types listed in the left column of Table 3 are valid for both CL1 and CL2; however, valid structure for these types differs between CL1 and CL2, as shown in the center and right columns.

Type	CL1 Structure	CL2 Structure
DeploymentDescriptorType	Choice of one of the following: *InstallableUnit*, *ConfigurationUnit*, or *LocalizationUnit*	Choice of one of the following: *InstallableUnit*, *ConfigurationUnit*, or *LocalizationUnit*; or one or more *CompositeInstallable* elements
RequirementType	Sequence of *ResourceConstraint* elements	Unbounded choice of *ResourceConstraint* elements and *Dependency* elements
AlternativeRequirementType	Sequence of *ResourceConstraint* elements	Unbounded choice of *ResourceConstraint* elements and *Dependency* elements

Table 3. Altered types in CL2.

5.2.2.3 SDD Types Introduced in CL2

As seen in Table 2, CL2 adds two new elements to *DeploymentDescriptor*. The *CompositeInstallable* element provides the definition of an aggregate deployment. *CompositeInstallable* is a complex element with many sub-elements. The second element added to *DeploymentDescriptor* is *Requisites*. *Requisites* is a list of references to SDDs that can be used, if needed, to satisfy deployment requirements defined in the *CompositeInstallable*.

Table 4 includes the CL2 types that are introduced in support *CompositeInstallable* and *Requisites*

BaseContentType	FeatureType	PackageFeatureReferenceType
CompositeInstallableType	GroupsType	ReferencedPackageType
CompositeLocalizationType	GroupType	RelationshipConstraintType
CompositeUnitType	InternalDependencyType	RelationshipType
ConstrainedResourceType	LanguageSelectionType	RequiredContentSelectionType
ContentElementReferenceType	LocalizationContentType	RequisitesType
ContentListGroup	MultiplicityConstraintType	ResourceMapType

ContentSelectionFeatureType	MultiplicityType	ResultingChangeMapType
DependencyType	MultiSelectType	ResultingResourceMapType
FeatureReferenceType	NestedFeatureType	SelectableContentType
FeaturesType	OptionalLanguagesType	UniquenessConstraintType

Table 4 SDD types introduced in CL2.

5.2.2.4 Extended Enumeration Value in CL2

One SDD type has an additional enumeration value that is valid only for CL2-based implementations. The type listed in the left column of Table 5 is valid for both CL1 and CL2; however, the value in the right column is valid only for CL2.

Type	CL2 Enumeration Value
OperationType	repair

Table 5 Extended enumeration value in CL2.

5.3 Profiles

Profiles are intended to specify detailed information that can be used in an SDD to promote interoperability. An SDD profile is defined consistent with **[CONFORM]**, to identify the functionality, parameters, options and/or implementation requirements necessary to satisfy the requirements of a particular community of users. SDD profiles are intended to enable a specific set of use cases, typically in a particular domain. Profiles are considered largely orthogonal to CLs; whereas a CL is a subset of the schema, a profile specifies the usage of the schema, including appropriate conventions and content values, to accomplish a particular set of use cases (typically in a particular domain).

A *starter profile* is initially defined with version 1.0 of this specification and is published separately. This starter profile defines terms and patterns that can be used to generate other specific profiles and addresses the content values that are required to support the SDD XML examples that also are published separately.

The starter profile is not intended to be a complete vocabulary for all SDDs, but rather to illustrate the format and provide example content so that additional profiles can be generated in the future. The starter profile leverages and extends the CIM standard **[CIM]** for many content values, but other profiles MAY use other content values.

Other profiles MAY be published by the TC in the future, and new profiles can be created as specified in 5.3.1.

An implementation MAY claim conformance to one or more particular profiles.

5.3.1 Profile Creation

The SDD TC has created a starter profile as described in 5.3. The SDD TC MAY create additional profiles in the future.

Others MAY create SDD profiles for use cases, domains, or user communities that are not addressed by the currently available profiles from the SDD TC. When creating new profiles, it is RECOMMENDED that profile creators follow the model of the starter profile and any existing profiles and reuse content from existing standards where possible. It is also RECOMMENDED that implementations publish the profile(s) that they support.

5.3.2 Profile Publication

The SDD TC publishes the starter profile and MAY publish any other profiles created by the SDD TC.

3926   Profiles created by the SDD TC SHALL be made available by the SDD TC.

3927   Profiles created by others MAY be published and made available by those parties and/or submitted to the
3928   SDD TC for consideration for publication by the SDD TC, according to the OASIS policies and
3929   procedures, including intellectual property rights. The SDD TC MAY publish and make available the new
3930   profiles through majority vote of the TC.

3931 5.3.3 Profile Applicability

3932   Profiles are applicable to particular usage models, domains and/or user communities. An implementation
3933   MAY claim conformance to one or more particular profiles.

3934 5.4 Compatibility Statements

3935   Versions of the specification use the version value defined in the *schemaVersion* attribute described in
3936   section 3.2. New versions of the specification MAY update the conformance level contents. Changes
3937   made to the specification for each new version are listed in Appendix [B].

3938   Profiles also use the *schemaVersion* attribute described in section 3.2. New versions of profiles MAY
3939   update the profile contents.

3940   Minor version updates of the schema, specification and profiles SHALL be backward-compatible with
3941   proceeding major versions (for example, all "1.x" versions are backward-compatible with version "1.0").

3942   Moreover, minor version updates of the schema, specification and profiles SHALL be backward-
3943   compatible with proceeding minor versions of the same major version (for example, version "1.4" is
3944   backward-compatible with versions "1.3", "1.2", "1.1" and "1.0").

3945   Major version updates of the schema, specification and profiles are NOT REQUIRED to be backward-
3946   compatible with previous versions and MAY NOT be backward-compatible with previous versions. For
3947   example, if non-backward-compatible changes occur in version "1.x", the new version is "2.0". Although
3948   new major versions MAY have substantial backward compatibility, backward compatibility is not
3949   guaranteed for all aspects of the schema across major versions.

3950 5.5 Conformance Clause

3951 5.5.1 Conformance for Users of This Specification

3952   An SDD conforms to this specification if it conforms to the SDD schema and follows the syntax and
3953   semantics defined in the normative portions of this specification. An SDD MAY conform to conformance
3954   levels CL1 or CL2.

3955   An implementation conforms to this specification if it conforms to, at minimum, conformance level CL1 of
3956   the SDD schema; supports at least one SDD profile; and follows the syntax and semantics defined in the
3957   normative portions of this specification. An implementation MAY support conformance levels CL1 or CL2
3958   and MAY support additional SDD profiles.

3959 5.5.2 Conformance for This Specification Itself

3960   This section is the conformance claim for how this document conforms to **[CONFORM]**. The conformance
3961   issues in section 8 of **[CONFORM]** apply to this document as follows:

3962   1. This document is applicable to SDDs as defined in this specification. To claim conformance to this
3963   document, all the requirements in section 5.5.1 SHALL be met.

3964   2. This document MAY be implemented in its entirety or in defined conformance levels CL1 and CL2.
3965   This document does not define profiles, but the SDD TC MAY define profiles that MAY be
3966   implemented.

3967   3. This document allows extensions. Each implementation SHALL fully support all required
3968   functionality of the specification exactly as specified. The use of extensions SHALL NOT
3969   contradict nor cause the non-conformance of functionality defined in the specification.

3970   4. This document contains no discretionary items.

3971   5. This document's normative language is English. Translation into other languages is permitted.

3972

A. Schema and Non-Normative Resource File List

The SDD schema is implemented by multiple schema files. Types defined in each file are identified by a specific namespace prefix, as indicated in the following list:

▪ **sdd-common-2.0.xsd** (prefix: sdd-common)

  Contains definitions of common types used in the SDD specification, including identity and fix-identity types, UUID and version types, and the display text type. The following namespace document describes this namespace and contains a directory of links to related resources, including the CL1 and Full Schema files:

  http://docs.oasis-open.org/sdd/v2.0/sdd-common-2.0.html

▪ **sdd-deploymentDescriptor-2.0.xsd** (prefix: sdd-dd)

  Contains the deployment descriptor specification, including various content types. The following namespace document describes this namespace and contains a directory of links to related resources, including the CL1 and Full Schema files:

  http://docs.oasis-open.org/sdd/v2.0/sdd-deploymentdescriptor-2.0.html

▪ **sdd-packageDescriptor-2.0.xsd** (prefix: sdd-pd)

  Contains the package descriptor specification, including types related to packages and files. The following namespace document describes this namespace and contains a directory of links to related resources, including the CL1 and Full Schema files:

  http://docs.oasis-open.org/sdd/v2.0/sdd-packagedescriptor-2.0.html

Additional non-normative files referenced by the SDD specification include example SDDs, the SDD Primer, and the SDD Starter Profile. These documents are provided as supplemental resources for SDD authors.

Example SDDs showing the use of the schema can be found at the following address:

  http://docs.oasis-open.org/sdd/v2.0/wd01/sdd-examples-v2.0-wd01.zip

The SDD Primer can be found at:

  http://docs.oasis-open.org/sdd/v2.0/wd01/sdd-primer-v2.0-wd01.doc

The SDD Starter Profile can be found at:

  http://docs.oasis-open.org/sdd/v2.0/wd01/sdd-starter-profile-v2.0-wd01.doc

HTML and PDF versions of the SDD Primer and the SDD Starter Profile are also available in the locations provided above.

B. Changes from previous versions

The following are the changes between the "Solution Deployment Descriptor v1.0" specification and the "Solution Deployment Descriptor v2.0" specification.

Items highlighted below in **bold** are elements that are deprecated starting with the "Solution Deployment Descriptor v2.0" specification. These elements were also changed in the SDD v2.0 schemas to a new type in order to remain compatible. However, in order to reduce confusion and to emphasize that these elements should no longer be used, the referenced sections in the SDD v2.0 specification for these elements were not updated to reflect the new type; instead, they have been updated to indicate that the element is deprecated. The text is unchanged from the SDD v1.0 specification.

- Changed version number from 1.0 to 2.0.
- Refreshed all images and updated diagram conventions in section [1.11].
- Added new Non-Normative Reference to SDD Examples in section [1.13].
- Added clarification to *Requirements* [2.7].
- Removed *descriptorLanguageBundle* attribute from *DescriptorInfoGroup* [3.2] and *DeploymentDescriptor* [4.1]; moved *descriptorLanguageBundle* attribute to *PackageDescriptor* [3.1].
- Updated version number in *schemaVersion* attribute in *DescriptorInfoGroup* [3.2].
- Added clarification to *ContentType* [3.12].
- Added clarification to *pathname* attribute in *ContentType* [3.12].
- Added clarification to the *descriptorLanguageBundle* enumeration value within the purpose attribute in *ContentType* [3.12].
- Added clarification to the *ResourceType* [4.2.2].
- Added *implementedBy* attribute to *ResourceType* [4.2.2].
- **Deprecated *Name* element from *ResourceType* [4.2.2].**
- Added clarification to the *Property* element in *ResourceType* [4.2.2].
- Changed *Value* element to *ElementValueType* [4.6.2] in *PropertyType* [4.2.3].
- Changed *Value* element to *ElementValueType* [4.6.2] in *ResultingPropertyType* [4.2.4].
- Added clarification/example to *targetResourceRef* attribute in *InstallableUnitType* [4.3.1].
- Added clarification to *OperationType* [4.3.7].
- Changed *name* and *value* attributes to *ElementValueType* [4.6.2] and renamed to *Name* and *Value* in *ArgumentType* [4.3.9]. Added clarification to both specific to use of *ElementValueType* [4.6.2].
- Added clarification to *Pattern* element in *SubstitutionType* [4.3.13].
- Changed *Value* element to *ElementValueType* [4.6.2] in *SubstitutionType* [4.3.13] and added clarification to *Value* specific to the use of *ElementValueType* [4.6.2].
- Added clarification to the *Constraints* [4.4] introduction.
- Changed *Minimum*, *Maximum*, *MinimumRecommended*, and *MaximumRecommended* elements to *ElementValueType* [4.6.2] in *CapacityValueType* [4.4.2] and added clarification to all specific to use of *ElementValueType* [4.6.2].
- Changed *ConsumptionConstraintValueType* [4.4.4] to extend *ElementValueType* [4.6.2].
- Changed *Value* element to *ElementValueType* [4.6.2] in *PropertyConstraintType* [4.4.5].
- Added clarification to the *Value* element in *PropertyConstraintType* [4.4.5].
- Changed *Value* element to *ElementValueType* [4.6.2] in *PropertyValueListType* [4.4.6].
- Added clarification to the *Value* element in *PropertyValueListType* [4.4.6].
- Added clarification to the *VersionConstraintType* [4.4.7].

4048　▪　Added clarification to the *Supported* and *Certified* elements in *VersionConstraintType* [4.4.7].

4049　▪　Added new *AuthorizationConstraintType* [4.4.14].

4050　▪　Added clarification and example to *ConditionalResouceConstraintType* [4.5.3].

4051　▪　**Deprecated *Name* element from *ConditionalResourceConstraintType* [4.5.3].**

4052　▪　Added *AuthorizationConstraint* element to *ConditionalResourceConstraintType* [4.5.3].

4053　▪　Added clarification to *testValue* attribute in *ConditionalResourceConstraintType* [4.5.3].

4054　▪　Changed *ConditionalPropertyConstraintType* [4.5.3] to reference *ElementValueType* [4.6.2] instead of
4055　　*VariableExpressionType* [4.6.1].

4056　▪　Improved description in *Variables* [4.6] introduction.

4057　▪　Added clarification to *VariableExpressionType* [4.6.1] and moved general variable content to
4058　　*Variables* [4.6] introduction section.

4059　▪　Added new *ElementValueType* [4.6.2].

4060　▪　Added new *StringPatternType* [4.6.3].

4061　▪　Added *ComplexParameter* and *ArrayParameter* elements to *ParametersType* [4.6.6].

4062　▪　Changed *defaultValue* attribute to *ElementValueType* [4.6.2] and renamed to *DefaultValue* in
4063　　*BaseParameterType* [4.6.6]. Added clarification to *DefaultValue* specific to *ElementValueType* [4.6.2].

4064　▪　Added clarification to *id* and *required* attributes in *BaseParameterType* [4.6.6].

4065　▪　Changed *LowerBound* and *UpperBound* elements to *ElementValueType* [4.6.2] in *BoundaryType*
4066　　[4.6.9] and added clarification to both specific to *ElementValueType* [4.6.2].

4067　▪　Added clarification to *BooleanParameterType* [4.6.12] specific to *ElementValueType* [4.6.2].

4068　▪　Added new *ComplexParameterType* [4.6.13.2].

4069　▪　Added new *ArrayParameterType* [4.6.15].

4070　▪　Added new *IntegerDataType* [4.6.16].

4071　▪　Added new *StringDataType* [4.6.17].

4072　▪　Added clarification to the *ResourcePropertyType* [4.6.18].

4073　▪　Added clarification to the *Requirements* [4.7] introduction.

4074　▪　Added clarification to the *id* attribute in *RequirementType* [4.7.2].

4075　▪　Added *AuthorizationConstraint* element to *ResourceConstraintGroup* [4.7.4].

4076　▪　**Deprecated *Name* element from *RequirementResourceConstraintType* [4.7.5].**

4077　▪　Added *AuthorizationConstraint* element to *RequirementResourceConstraintType* [4.7.5].

4078　▪　Added clarification to *testValue* attribute in *RequirementResourceConstraintType* [4.7.5].

4079　▪　**Deprecated *Name* element from *RequiredBaseConstraintType* [4.7.8].**

4080　▪　Added clarification to *testValue* attribute in *RequiredBaseConstraintType* [4.7.9].

4081　▪　**Deprecated *Name* element from *ResultingResourceType*. [4.8.1]**

4082　▪　Added clarification to *Property* element in *ResultingResourceType* [4.8.1].

4083　▪　**Deprecated *Name* element from *ResultingChangeType* [4.8.2].**

4084　▪　Added clarification to *Property* element in *ResultingChangeType* [4.8.2].

4085　▪　Added clarification to *ResultingResourceMap* element in *ReferencedPackageType* [4.10.1].

4086　▪　Added clarification to the *ResourceMapType* [4.10.2].

4087　▪　**Deprecated *Name* element from *ResultingResourceMapType* [4.10.3].**

4088　▪　Added clarifications to *ResultingResourceMapType* [4.10.3] and to the *Version*, *FixName*, and
4089　　*Property* elements in *ResultingResourceMapType* [4.10.3].

4090　▪　**Deprecated *Name* element from *ResultingChangeMapType* [4.10.4].**

| 4091 | ▪ | Added clarifications to *ResultingChangeMapType* [4.10.4] and to the *Property* element in |
| 4092 | | *ResultingChangeMapType* [4.10.4]. |

4091 ▪ Added clarifications to *ResultingChangeMapType* [4.10.4] and to the *Property* element in
4092 *ResultingChangeMapType* [4.10.4].

4093 ▪ Changed *selections* attribute to *ElementValueType* [4.6.2] and renamed to *Selections* in
4094 *ContentSelectionFeatureType* [4.12.9]. Added clarification to *Selections* specific to
4095 *ElementValueType* [4.6.2].

4096 ▪ Added clarification to *translationKey* attribute in *DisplayTextType* [4.14.3].

4097 ▪ Incorporated v1.0 Errata into Appendix [A].

4098 ▪ Added new Appendix [B] to list summary of changes to the specification from version to version.

4099

C. Acknowledgements

4100

4101 The following individuals have participated in the creation of this specification and are gratefully
4102 acknowledged: