Service Modeling Language, Version 1.1

...

Table of Contents

Deleted: Rules

Deleted: Validation

Deleted: sml:targetElement

Deleted: sml:targetRequired

Deleted: sml:targetType

Deleted: 4.4.2.4 Derivation by Restriction 4.4.2.5 Target Constraints and SML Reference Categories

Deleted: 1

Deleted: 2

Deleted: 3

...

4.4 Constraints on References

SML supports the following attributes for expressing constraints on reference elements.

Table 4-1. Attributes

Name	Description
`sml:acyclic`	Used to specify whether cycles are prohibited for a reference.
`sml:targetRequired`	Used to specify that a reference's target element is required to be present in the model.
`sml:targetElement`	Used to constrain the name of the reference's target.
`sml:targetType`	Used to constrain the type of the reference's target.

Deleted: supported

Deleted: `sml:targetRequired` ... [1]

Deleted: schema

Comment: A bookmark (or anchor) is added here for later reference. Similarly for target*.

SML defines a new property for every Complex Type Definition component:

1. {acyclic} An xs:boolean value. Required.

Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.74 cm + Tab after: 1.37 cm + Indent at: 1.37 cm

The value of {acyclic} for `xs:anyType` is `false`.

And 3 new properties for every Element Declaration component:

1. {target required} An xs:boolean value. Required.
2. {target element} An Element Declaration component. Optional.
3. {target type} A Type Definition component. Optional.

4.4.1 sml:acyclic

`sml:acyclic` is used to specify that a cycle is not allowed for an SML reference type. Model validators that conform to this specification MUST support the `sml:acyclic` attribute on any <xs:complexType> element in a schema document. This attribute has type `xs:boolean` and its actual value can be either `true` or `false`.

4.4.1.1 Mapping from Schema

{acyclic} of a complex type definition is as specified by the appropriate case among the following:

1. If `sml:acyclic` is present, then {acyclic} has the actual value of this attribute.
2. Otherwise if its {base type definition} is a complex type definition, then {acyclic} has the same value of {acyclic} as its {base type definition}.
3. Otherwise ({base type definition} is a simple type definition), {acyclic} is `false`.

4.4.1.2 Schema Validity Rules

If a complex type definition **D**'s {base type definition} is also a complex type definition and has {acyclic} `true`, then **D** MUST have {acyclic} `true`.

4.4.1.3 Instance Validity Rules

If **CT** is a complex type definition with {acyclic} `true`, then instances of **CT** MUST NOT create cycles in the model. More precisely, the directed graph constructed in the following way MUST be acyclic:

1. The nodes in the graph are all the elements resolved to by SML references of type **CT** or types derived from **CT**
2. If a node **N** is or contains an SML reference **R** of type **CT** or a type derived from **CT**, and **R** resolves to **T** (which must also be a node in the graph), then an arc is drawn from **N** to **T**.

4.4.2 Constraints on SML Reference Targets

Deleted: and every Particle

Deleted: <#>{target element} An Element Declaration component. Optional. ¶

Deleted: is a boolean

Comment: This links to the {acyclic} property defined earlier. The same change is (or should be, if I missed some) made for all mention of {acyclic} and {target *}.

Formatted: HTML Code, Font: (Default) Times New Roman

Deleted: sml:acyclic;

Deleted: , it

Deleted: A cyclic type can be used to derive cyclic or acyclic reference types, but all derived types of an acylic reference type are acyclic. Model validators that conform to this specification MUST enforce the following:¶

Formatted: Font: Bold

Formatted: HTML Code, Font: (Default) Times New Roman

Formatted: Font: Bold

Formatted: HTML Code, Font: (Default) Times New Roman

Deleted: *Validation*

Formatted: HTML Code, Font: (Default) Times New Roman

Deleted: any

Deleted: whose arcs are SML references of a given complexType (and any derived types) and whose nodes are all the elements pointed to by this set of SML references,

Deleted: must

Deleted: .

SML defines three attributes: `sml:targetRequired`, `sml:targetElement`, and `sml:targetType`, for constraining the target of a reference. These three attributes are collectively called `sml:target*` attributes. Model validators that conform to this specification MUST support these attributes on all `xs:element` elements with a `name` attribute.

Deleted: `sml:targetRequired`,

Deleted: and they

Deleted: be supported on global and local element declarations

4.4.2.1 Mapping from Schema

1. {target required} is as specified by the appropriate case among the following:
   - If `sml:targetRequired` is present, then {target required} is the actual value of this attribute.
   - Otherwise if the element declaration has a {substitution group affiliation}, then {target required} is the same as that of the {substitution group affiliation}.
   - Otherwise if the element declaration **ED** is contained (directly, indirectly, or implicitly) in a content model of a complex type **D**, who is a restriction of another complex type **B** and **B** contains an element declaration **EB** with the same name as **ED**, then {target required} of **ED** is the same as that of **EB**.
   - Otherwise {target required} is false.
2. {target element} is as specified by the appropriate case among the following:
   - If `sml:targetElement` is present, then its actual value MUST resolve to a global element declaration **G**, and {target element} is **G**.
   - Otherwise if {substitution group affiliation} is not absent, then {target element} is the same as that of the {substitution group affiliation}.
   - Otherwise if the element declaration **ED** is contained (directly, indirectly, or implicitly) in a content model of a complex type **D**, who is a restriction of another complex type **B** and **B** contains an element declaration **EB** with the same name as **ED**, then {target element} of **ED** is the same as that of **EB**.
   - Otherwise {target element} is absent.
3. {target type} is as specified by the appropriate case among the following:
   - If `sml:targetType` is present, then its actual value MUST resolve to a global type definition **T**, and {target type} is **T**.

Comment: See bug 5063. Only trying to preserve existing bug broken statements. Will remove this bullet if 5063 is resolved with "no inheritance".

Comment: ditto

- Otherwise if {substitution group affiliation} is not absent, then {target type} is the same as that of the {substitution group affiliation}.
- Otherwise if the element declaration **ED** is contained (directly, indirectly, or implicitly) in a content model of a complex type **D**, who is a restriction of another complex type **B** and **B** contains an element declaration **EB** with the same name as **ED**, then {target type} of **ED** is the same as that of **EB**.
- Otherwise {target type} is absent.

Comment: ditto

4.4.2.2 Schema Validity Rules

Model validators that conform to this specification MUST enforce the following:

1. If a global element declaration **S** has a {substitution group affiliation} **G**, then all the following are true:
   - If **G** has {target required} `true` then **S** also has {target required} `true`.
   - If **G** has {target element} **TEG**, then **S** has {target element} **TES** and **TES** is the same as **TEG** or is in the substitution group of **TEG**.
   - If **G** has {target type} **TTG**, then **S** has {target type} **TTS** and **TTS** is validly derived from **TTG**.
2. If 2 element declarations **E1** and **E2** have the same {namespace name} and {name} and they are both contained (directly, indirectly, or implicitly) in a content model of a complex type, then **E1** and **E2** have the same {target required}, {target element}, and {target type}.
3. For a complex type **D** derived by restriction from its {base type definition} **B**, if an element declaration **ED** is included in **D** and an element declaration **EB** is included in **B,** and **ED** and **EB** satisfy the "NameAndTypeOK" constraint (see "Schema Component Constraint: Particle Valid (Restriction) ", section 3.9.6, "Constraints on Particle Schema Components", [XML Schema Structures] for XML Schema's definition of valid restrictions), then all the following are true:
   - If **EB** has {target required} `true` then **ED** also has {target required} `true`.
   - If **EB** has {target element} **TEB**, then **ED** has {target element} **TED** and **TED** is the same as **TEB** or is in the substitution group of **TEB**.
   - If **EB** has {target type} **TTB**, then **ED** has {target type} **TTD** and **TTD** is validly derived from **TTB**.

Note (non-normative): The above condition #2 on the use of `sml:target*` attributes has been defined to reduce the implementation burden on model validators for verifying condition #3, that the use of `sml:target*` attributes is consistent across derivation by restriction. These conditions enable model validators to find the restricted particle for a restricting particle using a simple name match when `sml:target*` attributes are specified for these particles. In the absence of the above conditions, it is extremely difficult for SML validators to verify condition #3. In order to verify it, it is necessary to connect the particles in the derived type with those from the restricted base type. However, this level of support is not provided by most XML Schema frameworks; thus most SML validators would otherwise need to duplicate large parts of XML Schema's compilation logic to verify consistent usage of `sml:target*` attributes across derivation by restriction.

Deleted: s

Deleted: have

Deleted: consistent use of `sml:target*` attributes across a base type and its restricted derived type

Deleted: consistent use of an `sml:target*` attribute on a restricted particle in the base type and its restricting particle in a restricted derived type

4.4.2.3 Instance Validity Rules

If an element declaration **E** has {target required} `true`, then each element instance of **E** that is also an SML reference MUST target some element in the model, i.e. no instance of **E** can be a null or unresolved SML reference.

If an element declaration **E** has {target element} **TE**, then each element instance of **E** that is also a resolved SML reference MUST target an element that is an instance of **TE** or an instance of some global element declaration in the substitution group of **TE**.

If an element declaration **E** has {target type} **TT**, then each element instance of **E** that is also a resolved SML reference MUST target an element whose [type definition] is **TT** or a type derived from **TT**.

4.4.3 Reference Constraints Summary (non-normative)

The effect of the above instance validation rules is summarized in the following table.

Table 4-2. Target Constraints and SML Reference Categories.

	`Acyclic`	`targetRequired`	`targetElement`	`targetType`
Non-reference	Satisfied	Satisfied	Satisfied	Satisfied
Null	Satisfied	Violated	Satisfied	Satisfied
Unresolved	Satisfied	Violated	Satisfied	Satisfied
Resolved	Check	Satisfied	Check	Check

Comment: The above validation rules are actually silent about what happens when a “targetTyep” constraint is checked against a “null” reference. One could say “silent = not violated = satisfied”; one could also say “silent = not specified = implementation dependent”. This issue is not unique to this case. Whenever we say “A MUST be true”, it leaves the question “how about not(A)” open. If we believe this needs to be tightened up, a generic statement should be given in the spec, instead of trying to specify it everywhere.

4.5 Identity Constraints

XML Schema supports the definition of key, unique, and key reference constraints through `xs:key`, `xs:unique`, and `xs:keyref` elements. However, the scope of these constraints is restricted to a single document. SML defines analogs for these constraints, whose scope extends to multiple documents by allowing them to traverse SML references.

Model validators that conform to this specification MUST support the following elements for defining identity constraints across references, as child elements of `xs:element/xs:annotation/xs:appinfo` where the `xs:element` has a `name` attribute:

Name	Description
`sml:key`	Similar to `xs:key` except that the selector and field XPath expression can use the `smlfn:deref` function
`sml:unique`	Similar to `xs:unique` except that the selector and field XPath expression can use the `smlfn:deref` function
`sml:keyref`	Similar to `xs:keyref` except that the selector and field XPath expression can use the `smlfn:deref` function

SML defines a new property for every Element Declaration component:

1. {SML identity-constraints definitions} A set of SML identity constraint definitions components, which have the same set of properties as XML Schema identity constraint definitions.

4.5.1 Syntax and Semantics

Names of all SML identity constraint definitions exist in a single symbol space, which is disjoint from any symbol space of XML Schema components.

4.5.1.1 Mapping from Schema

For each `sml:key`, `sml:unique`, or `sml:keyref` element without the `ref` attribute specified, {SML identity-constraints definitions} contains a component corresponding to this element, as specified in section 3.11 of the XML Schema specification [XML Schema Structures], where `sml:selector` and `sml:field` elements are used in place of `xs:selector` and `xs:field`.

For each `sml:key`, `sml:unique`, or `sml:keyref` element with the `ref` attribute specified, {SML identity-constraints definitions} contains the component resolved to by the actual value of the `ref` attribute, with the following conditions:

1. The `name` attribute MUST NOT be specified.
2. the `sml:selector` and `sml:field` child elements MUST NOT be specified.
3. If the element is `sml:key`, then the value of `ref` attribute MUST resolve to an SML key constraint.
4. If the element is `sml:unique`, then the value of the `ref` attribute MUST resolve to an SML unique constraint.
5. If element is `sml:keyref`, then the value of the `ref` attribute MUST resolve to an SML keyref constraint, and the `refer` attribute MUST not be specified.

In addition to SML identity constraints obtained from the above explicit definitions or references, if an element declaration **S** has a {substitution group affiliation} **G**, then its {SML identity-constraints definitions} also contains members of {SML identity-constraints definitions} of **G**.

If an element declaration **ED** is contained (directly, indirectly, or implicitly) in a content model of a complex type **D**, who is a restriction of another complex type **B** and **B** contains an element declaration **EB** with the same name as **ED**, then {SML identity-constraints definitions} of **ED** also contains members of {SML identity-constraints definitions} of **EB**.

Comment: See bug 5063. Only trying to preserve existing bug broken statements. Will remove this paragraph if 5063 is resolved with "no inheritance".

4.5.1.2 Schema Validity Rules

1. {selector} in SML identity constraints has the same syntax as that defined in the XML identity constraint selector XPath syntax with one exception. The SML identity constraint {selector} XPath allows `smlfn:deref()` functions, nested to any depth, at the beginning of the expression. The XML identity constraint selector Path production is amended to support this requirement as defined below.

```
Path ::= ('.//')? Step ('/'Step)* | DerefExpr
DerefExpr ::= (NCName ':')? 'deref(' ( Step ('/'Step)* |
DerefExpr ) ')' ('/'Step)*
```

2. The `sml:field` XPath expression MUST conform to the amended BNF defined above for the selector XPath expression with the following modification, to allow `smlfn:deref()` functions, nested to any depth, at the beginning of the expression.

```
Path ::= ('.//')?  (Step'/')* ( Step | @NameTest ) |
         DerefExpr ('/' @NameTest)?
```

3. The {SML identity-constraints definitions} of an element declaration MUST NOT contain two identity constraints with the same name.

   Note: This could happen if the `ref` attribute resolves to an identity constraint already contained in the same element declaration's {SML identity-constraints definitions}.

   Comment: This is to reproduce the current rule 3.a: "If the ref attribute is specified for an SML identity constraint element that is specified for an element declaration E, then the value of ref attribute MUST NOT be the name of any other SML identity constraint element specified for E."

4. If a global element declaration **S** has a {substitution group affiliation} **G**, then {SML identity-constraints definitions} of **S** MUST be a superset of that of **G**.

5. If two element declarations **E1** and **E2** have the same {namespace name} and {name} and they are both contained (directly, indirectly, or implicitly) in a content model of a complex type, then **E1** and **E2** MUST have the same set of {SML identity-constraints definitions}.

   Note: This rule is defined to reduce the implementation burden for model validators. It facilitates the matching of restricting and restricted

   Comment: Bug 5091 should be responsible for making it clear that all notes are non-normative.

particles using their names, and avoids the replication of large parts of XML Schema's compilation logic for this purpose.

6. For a complex type **D** derived by restriction from its {base type definition} **B**, if **ED** is included in **D** and **EB** is included in **B** and **ED** and **EB** satisfies the "NameAndTypeOK" constraint (see "Schema Component Constraint: Particle Valid (Restriction) ", section 3.9.6, "Constraints on Particle Schema Components", [XML Schema Structures] for XML Schema's definition of valid restrictions), then {SML identity-constraints definitions} of **ED** MUST be a superset of that of **EB**.

4.5.1.3 Instance Validity Rules

Validation rules for SML identity constraints are the same as specified in section 3.11 of the XML Schema specification [XML Schema Structures], with the addition of support for the `smlfn:deref()` function.

Deleted: 1

4.5.2 University Example

...

sml:targetRequired	Used to specify that a reference's target element is required to be present model.