



2 Proposal for Extensibility Features in the 3 SAML Schemas

4 Proposal Draft 02, 1 February 2004

5 Document identifier:

6 sstc-maler-schema-extension-02

7 Location:

8 http://www.oasis-open.org/committees/documents.php?wg_abbrev=security

9 Authors:

10 Scott Cantor, individual (cantor.2@osu.edu)

11 Eve Maler, Sun Microsystems (eve.maler@sun.com)

12 Abstract:

13 This document proposes changes to the various extensibility mechanisms used in SAML's XML
14 schemas. It is hoped that some of the proposal's content will contribute to a technical paper on
15 the SAML TC's extensibility design decisions and/or to the SAML core specification's explanation
16 of extensibility.

18 Status:

19 This is the second draft of this proposal, incorporating some feedback and decisions from the 20
20 January 2004 Security Services TC telecon and additional ideas from the XML community.

21 Comments should be sent to the authors or to the security-services@lists.oasis-open.org mailing
22 list.

22 **Table of Contents**

23 1 Introduction.....3

24 1.1 SAML Customizations to Date.....3

25 1.2 SAML's Extensibility Requirements.....3

26 2 Extensibility Mechanism Choices.....6

27 2.1 Derivable Types.....6

28 2.2 Model Groups and Attribute Groups.....7

29 2.3 Element Substitution.....8

30 2.4 Content Models Containing "Any" Wildcards.....8

31 2.5 Elements Bound to "Any" Types.....9

32 2.6 Global Elements and Attributes.....10

33 2.7 URI-Based Identifiers and NamespacesSAML Namespaces.....10

34 3 Recommendations for SAML Extensibility.....12

35 3.1 Recommendations Related to Native Types.....12

36 3.2 Recommendations Related to Open Content.....12

37 3.3 Recommendations Related to Global Elements.....13

38 3.4 Recommendations Related to SAML Namespaces.....14

39 3.5 Recommendations Related to Extension Understandability.....14

40 4 References.....15

41

1 Introduction

The SAML TC has long had a goal of letting customizers extend SAML in a controlled way in order to maximize interoperability. SAML provides extensibility in three areas:

- **Extensibility of structure:** This includes ways to modify (add to or subtract from) SAML's native XML content models.
- **Extensibility of content:** This includes ways to customize the format and interpretation of the content of SAML's XML elements and attributes. (SAML artifacts could also be considered to come under this category.)
- **Extensibility of "protocol":** This includes ways to define new flows, called profiles, of SAML assertion creation, usage, and exchange. Sometimes these profiles also involve extended XML structures and content, as described above.

SAML was initially developed at a time when the art of W3C XML Schema (XSD) [Schema1] "schemography" was extremely new. Since SAML V1.0 first became stable, a number of users have gained experience with customizing SAML. This experience, along with lessons learned about the abilities and limitations of XSD and its conforming processors, are contributing to a new understanding of how SAML can best achieve its extensibility goal.

This proposal attempts to harness this experience to recommend improvements to SAML's extensibility in the areas of structure and content. (Note that schema features that do not affect extensibility are not discussed here, though this could easily become the subject of another position paper.)

1.1 SAML Customizations

The Liberty Alliance schemas [LibProtSchema] are the best-known and most comprehensive customization of SAML. These schemas were developed somewhat in parallel with the original SAML standard and were extended from stable but incomplete drafts of the original specification. This created some divergence as well as some duplication of work as SAML evolved and addressed limitations that Liberty also addressed in the form of extensions; these conflicts have been addressed in part through subsequent revisions of the Liberty schemas, demonstrating that extensions tend to be "evolving" specifications that must adapt as the underlying foundation changes. Understanding these points of contact and minimizing their impact is the essential goal of extensibility.

1.2 SAML's Extensibility Requirements

SAML's general requirements for extensibility can be stated as follows.

Note: The SAML Conformance Program Specification [SAMLConform] §2.4 outlines how extensions and their processors must behave in order to conform with SAML. For example, extensions are never allowed to redefine existing semantics nor alter specified behavior.

1. Allow customizers to modify SAML's XML structures in sufficiently powerful, maintainable, and validatable ways

SAML is generic technology that is designed to be reused, customized, and profiled for a variety of purposes. In general, any customization is likely to be promoted by its creators as a second-order standard, which suggests that customizers' need to create formal schemas is relatively high. This will be particularly true wherever a customizer wishes to make third-order customizations possible.

More controversially, in the past we have had an implicit goal of forcing customizers to define schema customization layers on top of SAML so that we could ensure their correct usage of the native SAML portions. With experience, it seems that this requirement was too nosy and, at times, too impractical.

2. To the extent possible, gracefully tolerate extensions

This is something of an inverse of requirement #1: SAML processors should be able to parse (and

88 validate) a document that has an extension in it even if the schema for the extension is not available.
89 This probably can't be achieved in all situations, but we can use it to color our choices of
90 mechanisms. Note that this requirement might have consequences for the Conformance document.
91 David Orchard's recent article on versioning [Orchard] discusses some ways to achieve this goal
92 more successfully.

93 **3. Make the extensibility mechanisms practical**

94 They should not be so cumbersome or costly that customizers avoid them and use cut-and-paste
95 instead. Support and interoperability of XSD processors and features are key variables here.

96 **4. Allow for unambiguous reuse of foreign taxonomies where SAML is just a carrier for them**

97 In general, SAML tries to avoid doing much more than codifying current usage unless current
98 technologies are wholly inadequate. Thus, if a system of codes, keywords, or values is already
99 available, SAML needs to allow for a way to indicate interpretation of this information. An example is
100 SAML's action namespace framework, which allows usage of existing sets of actions, such as HTTP
101 verbs and UNIX file permissions. Another example is SAML's attribute namespace framework, which
102 has been less successful because it apparently allows for ambiguous semantics.

2 Extensibility Mechanism Choices

The following sections describe the major contenders for extensibility mechanisms, SAML V1.1's usage of each, and brief analyses of their strong and weak points. This is not an exhaustive list of the mechanisms made available by XSD. The analysis focuses on these characteristics:

- **Tools support:** This addresses #1, the practicality goal.
- **Interactions with other mechanisms:** This also addresses #1, the practicality goal.
- **Validation power:** This addresses the goal of allowing customizations to be validated in #2. Some checking can be done at “compile time” (with the schema alone as input), and some is done at “run time” (with the schema and the XML instance as input).
- **Flexibility:** This addresses the goal of allowing sufficiently powerful customizations in #2.
- **Reuse:** This addresses the goal of allowing sufficiently powerful customizations in #2, and also #3, which deals with SAML's ability to reuse in the other direction.

2.1 Derivable Types

These are types, abstract and otherwise, that are designed for derivation. XSD allows a type to serve as a base type (a parent) of a specialized type, à la object-oriented systems, unless the original type is set to “final.” This setting can be controlled globally for a whole schema.

For the complex types that get bound to XML element content models, you can do type extension, which allows adding to the end of a content model, and type restriction, which allows subtracting any optional element from a content model. For the simple types that get bound to simple element and attribute string content, only restriction through “facets,” extension by allowing a list of multiple same-typed values, and extension by unioning two types are allowed.

XSD allows the redefinition (without renaming) of imported types in an importing schema.

All of SAML's defined types are non-final and are explicitly documented as being derivable. In several cases, SAML defines “deep” complex type hierarchies (and matching elements) especially for derivation purposes. In the assertion schema, this includes:

- **ConditionAbstractType > AudienceRestrictionConditionType**
- **StatementAbstractType > SubjectStatementAbstractType > (AuthenticationStatementType, AttributeStatementType, AuthorizationDecisionType)**
- **AttributeDesignatorType > AttributeType**

In the protocol schema, this includes:

- **RequestAbstractType > RequestType**
- **QueryAbstractType > SubjectQueryAbstractType > (AuthenticationQueryType, AttributeQueryType, AuthorizationDecisionQueryType)**
- **ResponseAbstractType > ResponseType**

The types with “Abstract” in their name are, in fact, set to be abstract, which means they must be derived from in order to be used at all. The SAML core specification [SAMLCore] §6 mentions that derivations of SAML's types may be used either with the XSD `xsi:type` attribute on native elements or through substitution of a foreign element bound to the new type.

Tools support

Extension tends to be supported. No problems have been reported with SAML customizations of this sort so far. Support for restriction of complex types may be a concern, though trivial restriction from **anyType** or **anySimpleType** is likely not to pose a problem; XSD processors tend to be sloppy about checking compatibility of the restricted type at compile time. At least one expert is suspicious of the overall interoperability of derivation from complex types [Kawaguchi]. Element substitution of restricted types appears to be a problem for some XSD processors.

148 We don't know of any cases of restriction from the SAML schemas.

149 **Interactions with other mechanisms**

150 Derivation gains significant new abilities in combination with substitution and redefinition, but need not
151 be used with them (and in this way avoids their negatives).

152 Use of the `<x:any>` wildcard at the end of a sequence causes problems when extending the original
153 sequence because of ambiguities between optional elements and wildcard matching.

154 **Validation power**

155 Derivations of SAML types are validatable in XSD processors, though it should be noted that at least one
156 expert is concerned about run-time misinterpretations [Obasanjo1]. Any other mechanism that allows for
157 or requires derivation will gain its validation power. When abstract types are used (as in some cases in
158 SAML), derivation (and validation of it) is forced.

159 **Flexibility**

160 Based on experience to date, the ability to extend types by adding to the end of the content model is
161 sufficient for most customizations, except when the original model terminates in the `<x:any>` wildcard.

162 **Reuse**

163 Customizers' reuse of native constructs: Non-final types are all available to be used in the construction of
164 an entirely new schema that does not use the native elements. It is not possible to forbid this.

165 Native reuse of others' constructs: Our schemas of interest (SAML) could reuse the types defined by
166 others' schemas. (It currently does not; the only "foreign" types it are ones built into the XSD
167 specification.)

168 **2.2 Model Groups and Attribute Groups**

169 Model groups are like programming macros that allow for usage of a previously defined XML content
170 model particle or list of attributes. They can be used to build new content models without the usual
171 extension/restriction constraints, and more importantly for the purposes this paper, an importing schema
172 can redefine a model group (without changing its name) to have different content.

173 SAML does not currently use model groups, so schema customizations can't redefine them.

174 **Tools support**

175 We don't know if tools support is an issue here. At a guess, proper support for redefinition is less
176 universal than support for basic usage of model and attribute groups.

177 **Interactions with other mechanisms**

178 Model and attribute groups don't depend on other mechanisms, though they can be used in combination
179 with type derivation.

180 **Validation power**

181 Model groups form content models, against which XML instances are validated by conforming XSD
182 processors. Attribute groups contribute to the list of attributes on an element, against which instances
183 are likewise validated. Any redefinitions of groups also get validated. There is no way to turn this
184 validation off.

185 **Flexibility**

186 Model and attribute groups allow for a greater degree of flexibility in content model/attribute list
187 modification than type derivation does. For example, you can use a model group for the middle of a
188 content model, which then allows for redefinition of only that portion of the content model; type extension
189 can only add to the end of a content model.

190 **Reuse**

191 Customizers' reuse of native constructs: Model and attribute groups can, we believe, be reused by any
192 schemas that import the schema in which the groups are originally defined.

193 Native reuse of others' constructs: Our schemas of interest (SAML) could reuse model and attribute
194 groups defined by others' schemas. (It currently contains no model or attribute groups, native or
195 otherwise.)

196 **2.3 Element Substitution**

197 In XSD, unless an element has been declared as “blocked,” other elements with a compatible type (the
198 same or a derived type) can appear in place of it. This setting can be controlled globally throughout a
199 schema. Substitution can be thought of as a type-controlled, late-binding version of an `<xs:choice>`
200 group, which allows a mutually exclusive choice among several elements.

201 SAML currently allows all elements to be heads of substitution groups.

202 ***Tools support***

203 Some tools do not seem to fully support this mechanism, especially when combined with extension by
204 restriction.

205 ***Interactions with other mechanisms***

206 Substitution is typically used with derivable types, in order to substitute derived-type elements for base-
207 type elements. It's possible to use substitution without derivation by substituting an element with an
208 identical type, rather than a derived one, for the original element.

209 ***Validation power***

210 The type compability of the substituted element for the head element is validated at compile time by XSD
211 processors. This can't be turned off.

212 ***Flexibility***

213 Some have commented that substitution is more limited than `<xs:choice>` groups [Kawaguchi]
214 because it can only occur with compatible types rather than arbitrary types. However, substitution
215 groups allow for customizers to extend “choices” after the original SAML schema creation, which
216 `<xs:choice>` can't do. Note that type derivation used alone also allows for this possibility with the use
217 of the `xsi:type` attribute.

218 ***Reuse***

219 Does not apply.

220 **2.4 Content Models Containing “Any” Wildcards**

221 These are content models that contain the XSD `<xs:any>` and `<xs:anyAttribute>` particles. They
222 create partially or fully “open” portions of a content model, where a variety of specific elements not
223 foreseen by the original schema may appear.

224 The `<xs:anyAttribute>` wildcard is currently not used anywhere in SAML. The `<xs:any>` wildcard
225 is used in content models as follows:

- 226 • `<Advice>` in the assertion schema contains `<Assertion>`, `<AssertionIDReference>`, and
227 `##other` laxly validated elements
- 228 • `<StatusDetail>` in the protocol schema contains `##any` laxly validated elements

229 ***Tools support***

230 There seems to be reasonable support for these features in tools.

231 **Interactions with other mechanisms**

232 When `<xs:any>` is used with the `##any` namespace setting in a content model mixed with native
233 elements bound to derivable types, and a customizer derives a new type from one of these types and
234 creates XML instances where the element bound to this derived type is supplied in an `<xs:any>` spot
235 and uses `xsi:type`, the content model becomes “ambiguous.” The workaround used in the Liberty
236 Alliance customization was to invent an `<Extension>` element to hold the `<xs:any>` particle
237 exclusively, avoiding mixing it together with native elements in a content model.

238 `<xs:any>` also causes ambiguity if a new type extends a sequence containing `<xs:any>` and adds new
239 elements to the sequence. The new elements are not distinguishable from the original wildcard. This
240 suggests that `<xs:any>` is not effective in conjunction with type derivation and may be best used in
241 types that block (or lack the need for) derivation.

242 The effect of the `<xs:anyAttribute>` wildcard can be achieved implicitly when the **xs:anyType** is
243 bound to an element.

244 An element must be global to satisfy an `<xs:any>` particle.

245 **Validation power**

246 The `<xs:any>` wildcard allows the schemographer to set the desired level of validation when XML
247 content fills the “any” slot: `strict` (there must be a customized schema present that provides a
248 definition for this subtree), `lax` (validate if a customized schema is present covering this subtree), or
249 `none` (no validation will be done on this subtree). Note that use of `xsi:type` supersedes this optionality
250 and forces validation.

251 **Flexibility**

252 The universe of allowed elements in an “any” slot is controllable by namespace (a list of namespaces to
253 allow or bar, or arbitrarily namespaced elements with `##any`, or just elements from foreign namespaces
254 with `##other`). The `##any` setting gives maximum choice to the customizer (but causes problems, as
255 noted above under Interactions).

256 Also, as noted just above under Validation, the flexibility around validation power is greatest in this
257 mechanism.

258 **Reuse**

259 Does not apply.

260 **2.5 Elements Bound to “Any” Types**

261 These are elements that are assigned the **xs:anyType** and **xs:anySimpleType** types. These are types
262 that allow for “open” content but that also may be extended or restricted in order to close down some
263 options.

264 The **xs:anySimpleType** type is not currently used anywhere in SAML. Two elements in the assertion
265 schema are bound to **xs:anyType**:

- 266 • `<SubjectConfirmationData>`
- 267 • `<AttributeValue>`

268 **Tools support**

269 SAML has used this mechanism without any significant tools problems surfacing, though some older
270 parsers do occasionally exhibit problems validating mixed and attribute content

271 **Interactions with other mechanisms**

272 Using **xs:anyType** gives the effect of `<xs:anyAttribute>` for free on that element.

273 More tightly specified content models can be freely defined by extension of such an element, allowing
274 second order extensions to take an open content model and precisely define its use in that context.

275 **Validation power**

276 Since essentially any content is compatible with such an element, an XSD processor will permit anything
277 to appear without complaining. Thus, validation is essentially disabled for the element, unless `xsi:type`
278 or element substitution is used.

279 **Flexibility**

280 An element bound to **xs:anyType** essentially stands in the original schema as a placeholder for
281 whatever the same schema or an extension might define as a specific instance of that element. It thus
282 provides a "conceptual" placeholder in a content model in which any XML can be used to represent the
283 concept.

284 It also enables a combination of validating and non-validating behavior, which is useful when
285 interpretation of the non-validated content is strictly optional or governed by other agreement
286 mechanisms. SAML attribute values are an example of this.

287 **Reuse**

288 Does not apply.

289 **2.6 Global Elements and Attributes**

290 Elements and attributes that are global are allowed to be reused independently in the creation of a new
291 foreign schema. Global elements can appear in `<xs:any>` wildcard content, and can serve as the root
292 element of an XML document.

293 All of SAML's elements are global (and none of its attributes are). SAML has not consciously considered
294 this to be an extensibility mechanism, but rather a general schema style that we hoped would be well
295 supported by tools. However, it would be wise to examine the potential reuse of SAML elements in this
296 light.

297 **Tools support**

298 There seems to be very good tools support for both native global elements and reuse of them in a
299 foreign schema, as this was the first use case XSD needed to solve.

300 **Interactions with other mechanisms**

301 See [Maler] for a comparative analysis of global elements, local elements, named types, and anonymous
302 types.

303 **Validation power**

304 Global elements (as well as local ones) have declarations and are bound to types, and are thus validated
305 against these types. Unless an element is bound to something like **xs:anyType**, the validation can't
306 effectively be turned off.

307 **Flexibility**

308 Global elements are more flexible than local ones, but are also more complex because of the usage (and
309 reuse) choices they offer.

310 **Reuse**

311 The main benefit of global elements is their reuse – appearing in more than one content model of other
312 elements – in the same or importing schemas.

313 **2.7 URI-Based Identifiers and Namespaces**

314 SAML uses URI-based identifiers for interpreting selected SAML element and attribute content correctly.
315 They are indicated through an attribute that contains a URI reference, which is designed to disambiguate
316 which of the many possible meanings for a particular SAML construct is meant. It is also possible in
317 some cases for customizers to assign a validatable type to the XML construct, so that it can be
318 syntactically validated.

319 In some of the cases where URI-based identifiers are used, SAML terms the mechanism "SAML

320 namespaces.” This name is typically used when the structure to be interpreted is inside the element on
321 which the URI appears and when the content is keyword-like.

322 This mechanism differs from the other mechanisms discussed above in that it is a technique specific to
323 the SAML vocabulary and not global to XSD. The SAML assertion schema uses this mechanism in the
324 following cases:

- 325 • Interpreting the name of an **action** to be performed on a resource by referring to a URI-based
326 identifier representing a set of names of actions (“action namespace”)
- 327 • Interpreting the name of an **attribute** by referring to a URI-based identifier representing a set of
328 attribute names (“attribute namespace”)
- 329 • Conveying the **authentication method** that was used for a subject in an authentication statement, or
330 the authentication method being provided to confirm the binding of a subject to the bearer in any kind
331 of assertion, by referring to a URI-based identifier representing an authentication method
- 332 • Interpreting the format of a **name identifier** by referring to a URI-based identifier representing the
333 format

334 ***Tools support***

335 This mechanism seems well supported, though it should be noted that the “semantics” gained from it is
336 outside the knowledge of a generic XSD processor; it is specific to the SAML schemas.

337 ***Interactions with other mechanisms***

338 The way SAML has defined this mechanism, it typically works hand in hand with assignment of a type for
339 syntactic checking of the content, though this validation is not required.

340 ***Validation power***

341 If the namespaced content is assigned a type as discussed above, it can be validated through normal
342 XSD means. Otherwise, any validation must be application-specific. (See [Maler] for a description of an
343 XML standard that uses a different mechanism to encourage syntactic validation much more strongly.)

344 ***Flexibility***

345 Because this mechanism can be entirely divorced from XSD validation, and because anyone can invent
346 a new namespace, it is very flexible. However, it may be that our underspecification of the attribute
347 namespace case may have allowed too much flexibility, causing some divergence in usage.

348 ***Reuse***

349 Namespaces defined by the SAML specifications and by customizers are available to be reused by
350 others.

3 Recommendations for SAML Extensibility

351

352 The following sections contain our recommendations for modifications to the SAML schemas and
353 specifications in order to support our extensibility goals more closely.

3.1 Recommendations Related to Native Types

354

Finality ✓

355

356 Keep all types non-final.

357 **Rationale:** The derivation mechanism is the most well-supported for allowing SAML constructs to be
358 extended, and is known to interact successfully with other mechanisms SAML uses (or should use). The
359 extreme power of model groups for extension has not proven to be needed, may not be fully supported in
360 tools, and is an unfamiliar design pattern for most schemographers.

361 **Action:** None.

Abstraction ✓

362

363 Continue to make selected base types abstract and provide matching elements for them in an
364 `<xs:choice>` group with non-abstract versions. The types that should be subjected to this treatment
365 are the ones that have no useful semantics in SAML all by themselves, but have partial semantics that
366 we feel may be useful to customizers are building blocks for their own SAML-derived work.

367 **Rationale:** This seems to be the only way to make a “deep” type hierarchy available, and experience
368 with existing customizations seems to suggest it’s working successfully.

369 **Issues:** Do we have a problem with the fact that this forces customizers to define derived types off of the
370 abstract ones? We don’t necessarily care if they do this, but the mechanism gives no choice.

371 **Action:** None.

Substitution ✓

372

373 **Option 1:** Block all substitution because its support, particularly in combination with other mechanisms,
374 is iffy. The action in this case would be to set `blockDefault` attribute at the top level of the schema.

375 **Option 2:** Allow it, recognizing its limitations, since customizers have the right to choose it if they wish.
376 No action would be required in this case.

377 **Decision:** The TC decided on 20 January 2004 to accept Option 1.

3.2 Recommendations Related to Open Content

378

xs:anyType vs. <xs:any>

379

380 **Option 1:** Migrate to using `xs:anyType` exclusively in order to stay away from known and potential
381 problems with `<xs:any>`, allow for schema-less processing, and become consistent across the SAML
382 schemas. The action in this case would be to work on the `<Advice>` declaration and `AdviceType`
383 definition to add a new subelement that would be bound to the `xs:anyType` type, and to work on the
384 `<StatusDetail>` element declaration to bind it to `xs:anyType` instead of `StatusDetailType`, which
385 could be removed.

386 **Option 2:** Migrate to using `<xs:any>` exclusively in order to give additional validation-level flexibility to
387 customizers, allow for schema-less processing, and become consistent across the SAML schemas. Note
388 that the substitution-blocking decision made earlier mitigates the major problems with `<xs:any>`;
389 however, it should be used as the lone content of whatever content model it appears in. The action in
390 this case would be to change `<SubjectConfirmationData>` and `<AttributeValue>` to contain
391 `<xs:any namespace="##any" processContents="lax">` and to change `<Advice>` and
392 `AdviceType` to add a new subelement that would contain only `<xs:any>`.

393 **Option 3:** Develop a set of criteria that provide guidance on when to use each mechanism, measure
394 each current usage against the criteria, and synchronize the usage as necessary. NB: This set of criteria

395 does not exist yet and would need to be developed! One of the likely criteria amounts to a guess as to
396 whether the element containing the `<xs:any>` would be likely to need derivation. Metadata is an
397 example. An element that amounts to an "element bag" seems like a poor candidate for extension and
398 using `<xs:any>` seems like a way of allowing new optional "bag contents" from extension schemas to be
399 defined while being ignored by older implementations.

400 **Option 4:** Do nothing and keep the current ad hoc split between the two mechanisms. This requires no
401 action, but also provides no guidance if we add future elements that need flexible/foreign content.

402

403 **`<xs:anyAttribute>` ✓**

404 **Option 1:** Consider adding `<xs:anyAttribute>` to all elements that aren't bound to **xs:anyType**
405 (which implies `<xs:anyAttribute>`), because allowing for open attribute content is a harmless but
406 powerful kind of extensibility that will allow SAML users to pick up and use interesting new global
407 attributes produced in other venues. The action in this case would be to add the `<xs:anyAttribute>`
408 particle to all complex type definitions except for those bound to **xs:anyType**.

409 **Option 2:** Don't do this because new attributes can always be added through type extension. No action
410 would be required in this case.

411 **Decision:** The TC decided on 20 January 2004 to take no action now, but to consider adding
412 `<xs:anyAttribute>` on a case-by-case basis as needed. (Note: David Orchard's article [Orchard]
413 recommending wide application of `<xs:anyAttribute>` has come to the authors' attention since the
414 decision was made.)

415 **3.3 Recommendations Related to Global Elements**

416 ***Restrained Globalness***

417 Keep all elements global, but add prose saying which elements are allowed to be root elements, roots of
418 SOAP payloads, and so on. Keep all attributes local, unless we see fit in the future to define any
419 "common attributes" for use in SAML that we think have utility beyond SAML.

420 **Rationale:** SAML has succeeded with its global elements so far, and its vocabularies are small enough
421 that there seems to be no pressure to have multiple elements with the same name but different content
422 (the main value of local elements). However, without further qualification, global elements are more
423 powerful than we intend most of our elements to be. Adding specification prose to prohibit usage of
424 elements that are "subelements by nature" will avoid odd conformance scenarios. If we were to switch to
425 using local qualified elements to enforce the "root element" constraint in a machine-readable way, we
426 would probably cause any XSLT-based SAML processing (at the least) to have to change, since local
427 element names can be globally non-unique and thus need to be additionally anchored (e.g., simple child
428 ladders would have to be anchored all the way up to a global ancestor).

429 **Action:** This would require adding prose to mention that only certain elements – we propose to start with
430 assertions, requests, responses, name identifiers, and attributes – MAY be reused in importing schemas
431 and appear as root elements in an XML document.

432 **3.4 Recommendations Related to SAML Namespaces**

433 ***Attribute Clarity***

434 **Note:** Be prepared to recognize the "XML attribute" vs. "attribute of a SAML subject"
435 distinction when reading this recommendation!

436 Add a new XML attribute on the `<AttributeDesignator>` and `<Attribute>` elements to capture the
437 source of a SAML attribute.

438 **Rationale:** Rather than continuing to allow users to overload `AttributeNameSpace` with multiple
439 meanings, it's better to settle on exactly what "metadata" is needed to fully disambiguate an attribute
440 name and allow SAML users to provide that metadata.

441 **Action:** This would require adding an attribute to **AttributeDesignatorType**. Note that any action here

442 may have dependencies on the resolutions of the W-28* (attribute-related) work items.

443 ***Semantic vs. Syntactic Clarity***

444 For SAML attributes and actions (and future any uses of the namespace mechanism), document more
445 thoroughly the relationship between a namespace (identifying the “semantic” basis for the value) and the
446 ability to assign a type to the element that holds the value of the attribute or action (identifying the
447 syntactic constraints to which the value should be held during validation).

448 **Rationale:** The spec says little on this point today, and the mechanism is likely to be better understood
449 and more widely and correctly used in future if we explain it better.

450 **Action:** This would require some writing only: no schema changes. Note that any action here may have
451 dependencies on the resolutions of the W-28* (attribute-related) work items.

452 ***Namespace Terminology***

453 Currently there's the namespace camp and the (unnamed) camp, with no solid connection between them.

454 **Option 1:** Unify the descriptions of all the parts of SAML that use the URI-based identifier mechanism,
455 focusing on the “identifier” language rather than the “namespace” language because it's more broad.
456 This would require changing the names of `AttributeNamespace` and `ActionNamespace` and
457 modifying any associated prose.

458 **Option 2:** Unify the descriptions of all the parts of SAML that use the URI-based identifier mechanism,
459 focusing on the “namespace” language rather than the “identifier” language because it's more evocative.
460 This would require modifying prose that discusses URI-based identifiers and calling them “SAML
461 namespaces.”

462 **Option 3:** Do nothing because the distinction hasn't caused any problems to date.

463 **3.5 Recommendations Related to Extension Understandability**

464 ***Must Ignore***

465 David Orchard's article [Orchard] recommends that vocabularies adopt an explicit model stating how to
466 process extensions, with the default recommendation being the “Must Ignore” rule (“Document receivers
467 MUST ignore any XML attributes or elements in a valid XML document that they do not recognize”) in
468 combination with extensibility techniques of the sort described in this document. Another choice might be
469 a fallback model. Do we want to adopt this posture in some fashion? It might require some expression in
470 the metadata, so that entities exchanging SAML can agree on which extensions they produce and
471 expect.

4 References

472

- 473 **[Kawaguchi]** K. Kawaguchi. "W3C XML Schema Made Simple", XML.com, 6 June 2001.
474 <http://www.xml.com/pub/a/2001/06/06/schemasimple.html>.
- 475 **[LibProtSchema]** S. Cantor et al., *Liberty ID-FF Protocols and Schema Specification*, Version 1.2,
476 Liberty Alliance Project, 12 November 2003.
477 <http://www.projectliberty.org/specs/liberty-idff-protocols-schema-v1.2.pdf>.
- 478 **[Maler]** E. Maler. "Schema Design Rules for UBL...and Maybe for You", XML 2002
479 paper, 8-13 December 2002.
480 [http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-01-02/05-01-](http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-01-02/05-01-02.html)
481 [02.html](http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-01-02/05-01-02.html). Note particularly Section 4. Also note that the schema examples in this
482 section share a common error: `<xs:sequence>` is missing from around the
483 content of `<xs:complexType>`.
- 484 **[Obasanjo1]** D. Obasanjo. "XML Schema Design Patterns: Is Complex Type Derivation
485 Unnecessary?", XML.com, 29 October 2003.
486 <http://www.xml.com/pub/a/2003/10/29/derivation.html>.
- 487 **[Orchard]** D. Orchard. "Versioning XML Vocabularies", XML.com, 3 December 2003.
488 <http://www.xml.com/pub/a/2003/12/03/versioning.html>.
- 489 **[SAMLConform]** E. Maler et al. *Conformance Program Specification for the OASIS Security
490 Assertion Markup Language (SAML)*. OASIS, September 2003. Document ID
491 oasis-sstc-saml-conform-1.1. <http://www.oasis-open.org/committees/security/>.
- 492 **[SAMLCore]** E. Maler et al. *Assertions and Protocol for the OASIS Security Assertion Markup
493 Language (SAML)*. OASIS, September 2003. Document ID oasis-sstc-saml-core-
494 1.1. <http://www.oasis-open.org/committees/security/>.
- 495 **[Schema1]** H. S. Thompson et al. *XML Schema Part 1: Structures*. World Wide Web
496 Consortium Recommendation, May 2001. <http://www.w3.org/TR/xmlschema-1/>.

A. Revision History

Rev	Date	By Whom	What
01	19 Jan 2003	Eve Maler, Scott Cantor	Initial draft.
02	28 Jan 2003	Eve Maler, Scott Cantor	Takes into account the feedback and decisions from the 20 January 2004 SSTC telecon, additional discussions between the authors, and an interesting article from David Orchard.

B. Notices

500 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
501 might be claimed to pertain to the implementation or use of the technology described in this document or
502 the extent to which any license under such rights might or might not be available; neither does it
503 represent that it has made any effort to identify any such rights. Information on OASIS's procedures with
504 respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights
505 made available for publication and any assurances of licenses to be made available, or the result of an
506 attempt made to obtain a general license or permission for the use of such proprietary rights by
507 implementors or users of this specification, can be obtained from the OASIS Executive Director.

508 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications,
509 or other proprietary rights which may cover technology that may be required to implement this
510 specification. Please address the information to the OASIS Executive Director.

511 **Copyright © OASIS Open 2004. All Rights Reserved.**

512 This document and translations of it may be copied and furnished to others, and derivative works that
513 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published
514 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright
515 notice and this paragraph are included on all such copies and derivative works. However, this document
516 itself does not be modified in any way, such as by removing the copyright notice or references to OASIS,
517 except as needed for the purpose of developing OASIS specifications, in which case the procedures for
518 copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required
519 to translate it into languages other than English.

520 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
521 or assigns.

522 This document and the information contained herein is provided on an "AS IS" basis and OASIS
523 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
524 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS
525 OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
526 PURPOSE.