



# OASIS Service Provisioning Markup Language (SPML) Version 2

## Committee Draft 1.0 2005 September 14

Document identifier: pstc-spml2-cd-01.pdf

Location: <http://www.oasis-open.org/committees/provision/docs/>

Send comments to: [pstc-comment@lists.oasis-open.org](mailto:pstc-comment@lists.oasis-open.org)

Editor:

Gary Cole, Sun Microsystems (Gary.P.Cole@Sun.com)

Contributors:

Jeff Bohren, BMC  
Robert Boucher, CA  
Doron Cohen, BMC  
Gary Cole, Sun Microsystems  
Cal Collingham, CA  
Rami Elron, BMC  
Marco Fanti, Thor Technologies  
Ian Glazer, IBM  
James Hu, HP  
Ron Jacobsen, CA  
Jeff Larson, Sun Microsystems  
Hal Lockhart, BEA  
Prateek Mishra, Oracle Corporation  
Martin Raepple, SAP  
Darran Rolls, Sun Microsystems  
Kent Spaulding, Sun Microsystems  
Gavenraj Sodhi, CA  
Cory Williams, IBM  
Gerry Woods, SOA Software

Abstract:

This specification defines the concepts and operations of an XML-based provisioning request-and-response protocol.

34 Status:

35 This is a candidate Committee Specification that is undergoing a vote of the OASIS  
36 membership in pursuit of OASIS Standard status.

37 If you are on the provision list for committee members, send comments there. If you are not  
38 on that list, subscribe to the [provision-comment@lists.oasis-open.org](mailto:provision-comment@lists.oasis-open.org) list and send  
39 comments there. To subscribe, send an email message to [provision-comment-](mailto:provision-comment-request@lists.oasis-open.org)  
40 [request@lists.oasis-open.org](mailto:provision-comment-request@lists.oasis-open.org) with the word "subscribe" as the body of the message.

41 Copyright (C) OASIS Open 2005. All Rights Reserved.

## 42 Table of contents

43	1	Introduction.....	7
44	1.1	Purpose .....	7
45	1.2	Organization .....	7
46	1.3	Audience.....	7
47	1.4	Notation .....	8
48	1.4.1	Normative sections .....	8
49	1.4.2	Normative terms.....	8
50	1.4.3	Typographical conventions .....	8
51	1.4.4	Namespaces .....	9
52	2	Concepts .....	10
53	2.1	Domain Model .....	10
54	2.1.1	Requestor .....	10
55	2.1.2	Provider.....	11
56	2.1.3	Target.....	11
57	2.1.3.1	Target Schema .....	11
58	2.1.3.2	Supported Schema Entities .....	12
59	2.1.3.3	Capabilities.....	12
60	2.1.4	Provisioning Service Object (PSO).....	13
61	2.2	Core Protocol.....	13
62	2.3	Profile.....	13
63	3	Protocol .....	14
64	3.1	Request/Response Model .....	14
65	3.1.1	Conversational flow.....	16
66	3.1.2	Status and Error codes .....	16
67	3.1.2.1	Status (normative).....	17
68	3.1.2.2	Error (normative).....	17
69	3.1.2.3	Error Message (normative) .....	18
70	3.1.3	Synchronous and asynchronous operations .....	19
71	3.1.3.1	ExecutionMode attribute .....	19
72	3.1.3.2	Async Capability .....	19
73	3.1.3.3	Determining execution mode .....	20
74	3.1.3.4	Results of asynchronous operations (normative) .....	22
75	3.1.4	Individual and batch requests .....	22
76	3.2	Identifiers .....	22
77	3.2.1	Request Identifier (normative) .....	23
78	3.2.2	Target Identifier (normative) .....	23
79	3.2.3	PSO Identifier (normative) .....	24
80	3.3	Selection.....	26

81	3.3.1	QueryClauseType .....	26
82	3.3.2	Logical Operators.....	26
83	3.3.3	SelectionType .....	27
84	3.3.3.1	SelectionType in a Request (normative).....	27
85	3.3.3.2	SelectionType Processing (normative) .....	28
86	3.3.3.3	SelectionType Errors (normative).....	29
87	3.3.4	SearchQueryType.....	29
88	3.3.4.1	SearchQueryType in a Request (normative) .....	30
89	3.3.4.2	SearchQueryType Errors (normative).....	31
90	3.4	CapabilityData .....	32
91	3.4.1	CapabilityDataType.....	32
92	3.4.1.1	CapabilityData in a Request (normative).....	33
93	3.4.1.2	CapabilityData Processing (normative) .....	34
94	3.4.1.3	CapabilityData Errors (normative).....	37
95	3.4.1.4	CapabilityData in a Response (normative) .....	37
96	3.5	Transactional Semantics .....	39
97	3.6	Operations .....	39
98	3.6.1	Core Operations.....	39
99	3.6.1.1	listTargets.....	39
100	3.6.1.2	add .....	50
101	3.6.1.3	lookup.....	56
102	3.6.1.4	modify.....	61
103	3.6.1.5	delete .....	71
104	3.6.2	Async Capability .....	74
105	3.6.2.1	cancel.....	75
106	3.6.2.2	status.....	77
107	3.6.3	Batch Capability .....	83
108	3.6.3.1	batch .....	83
109	3.6.4	Bulk Capability .....	90
110	3.6.4.1	bulkModify.....	90
111	3.6.4.2	bulkDelete .....	92
112	3.6.5	Password Capability .....	95
113	3.6.5.1	setPassword.....	95
114	3.6.5.2	expirePassword .....	97
115	3.6.5.3	resetPassword .....	98

116	3.6.5.4	validatePassword.....	100
117	3.6.6	Reference Capability.....	103
118	3.6.6.1	Reference Definitions.....	105
119	3.6.6.2	References.....	106
120	3.6.6.3	Complex References.....	106
121	3.6.6.4	Reference CapabilityData in a Request (normative).....	112
122	3.6.6.5	Reference CapabilityData Processing (normative).....	113
123	3.6.6.6	Reference CapabilityData Errors (normative).....	115
124	3.6.6.7	Reference CapabilityData in a Response (normative).....	115
125	3.6.7	Search Capability.....	116
126	3.6.7.1	search.....	117
127	3.6.7.2	iterate.....	123
128	3.6.7.3	closeIterator.....	129
129	3.6.8	Suspend Capability.....	133
130	3.6.8.1	suspend.....	133
131	3.6.8.2	resume.....	135
132	3.6.8.3	active.....	137
133	3.6.9	Updates Capability.....	140
134	3.6.9.1	updates.....	141
135	3.6.9.2	iterate.....	147
136	3.6.9.3	closeIterator.....	152
137	3.7	Custom Capabilities.....	157
138	4	Conformance (normative).....	158
139	4.1	Core operations and schema are mandatory.....	158
140	4.2	Standard capabilities are optional.....	158
141	4.3	Custom capabilities must not conflict.....	158
142	4.4	Capability Support is all-or-nothing.....	159
143	4.5	Capability-specific data.....	159
144	5	Security Considerations.....	160
145	5.1	Use of SSL 3.0 or TLS 1.0.....	160
146	5.2	Authentication.....	160
147	5.3	Message Integrity.....	160
148	5.4	Message Confidentiality.....	160
149	Appendix A.	Core XSD.....	161
150	Appendix A.	Async Capability XSD.....	168
151	Appendix B.	Batch Capability XSD.....	170
152	Appendix C.	Bulk Capability XSD.....	172
153	Appendix D.	Password Capability XSD.....	174
154	Appendix E.	Reference Capability XSD.....	176
155	Appendix F.	Search Capability XSD.....	178
156	Appendix G.	Suspend Capability XSD.....	181
157	Appendix H.	Updates Capability XSD.....	183

158	Appendix I. Document References .....	186
159	Appendix J. Acknowledgments .....	188
160	Appendix K. Notices.....	189

---

# 161 1 Introduction

## 162 1.1 Purpose

163 This specification defines the concepts and operations of Version 2 of the Service Provisioning  
164 Markup Language (SPML). SPML is an XML-based provisioning request-and-response protocol.

## 165 1.2 Organization

166 The body of this specification is organized into three major sections: Concepts, Protocol and  
167 Conformance.

- 168 • The [Concepts](#) section introduces the main ideas in SPMLv2. Subsections highlight significant  
169 features that later sections will discuss in more detail.
- 170 • The [Protocol](#) section first presents an overview of protocol features and then discusses the  
171 purpose and behavior of each protocol operation. The core operations are presented in an  
172 order that permits a continuing set of examples. Subsequent sections present optional  
173 operations.

174 Each section that describes an operation includes:

- 175 - The relevant XML Schema
- 176 - A *normative* subsection that describes the *request* for the operation
- 177 - A *normative* subsection that describes the *response* to the operation
- 178 - A *non-normative* sub-section that discusses *examples* of the operation
- 179
- 180 • The [Conformance](#) section describes the aspects of this protocol that a requestor or provider  
181 must support in order to be considered conformant.
- 182 • A [Security and Privacy Considerations](#) section describes risks that an implementer of this  
183 protocol should weigh in deciding how to deploy this protocol in a specific environment.

184 Appendices contain additional information that supports the specification, including references to  
185 other documents.

## 186 1.3 Audience

187 The PSTC intends this specification to meet the needs of several audiences.

188 One group of readers will want to know: **"What is SPML?"**

189 A reader of this type should pay special attention to the [Concepts](#) section.

190 A second group of readers will want to know: **"How would I use SPML?"**

191 A reader of this type should read the [Protocol](#) section  
192 (with special attention to the *examples*).

193 A third group of readers will want to know: **"How must I implement SPML?"**

194 A reader of this type must read the [Protocol](#) section  
195 (with special attention to normative *request* and *response* sub-sections).

196 A reader who is already familiar with SPML 1.0 will want to know: **"What is new in SPMLv2?"**

197 A reader of this type should read the [Concepts](#) section thoroughly.

## 198 1.4 Notation

### 199 1.4.1 Normative sections

200 Normative sections of this specification are labeled as such. The title of a normative section will  
201 contain the word “normative” in parentheses, as in the following title: “**Syntax (normative)**”.

### 202 1.4.2 Normative terms

203 This specification contains schema that conforms to W3C XML Schema and contains normative  
204 text that describes the syntax and semantics of XML-encoded policy statements.

205 The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",  
206 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be  
207 interpreted as described in IETF RFC 2119 [RFC2119]

208 *“they MUST only be used where it is actually required for interoperation or to limit*  
209 *behavior which has potential for causing harm (e.g., limiting retransmissions)”*

210 These keywords are capitalized when used to unambiguously specify requirements of the protocol  
211 or application features and behavior that affect the interoperability and security of implementations.  
212 When these words are not capitalized, they are meant in their natural-language sense.

### 213 1.4.3 Typographical conventions

214 This specification uses the following typographical conventions in text:

Format	Description	Indicates
xmlName	monospace font	The name of an XML <i>attribute, element or type</i> .
“attributeName”	monospace font <i>surrounded by double quotes</i>	An instance of an XML <i>attribute</i> .
`attributeValue`	monospace font <i>surrounded by double quotes</i>	A literal value (of type string).
“attributeName=‘value’”	monospace font name <i>followed by equals sign and value surrounded by single quotes</i>	An instance of an XML <i>attribute value</i> . Read as “a value of (value) specified for an instance of the (attributeName) attribute.”
{XmlTypeName} or {ns:XmlTypeName}	monospace font <i>surrounded by curly braces</i>	The name of an XML <i>type</i> .
<xmlElement> or <ns:xmlElement>	monospace font <i>surrounded by &lt;&gt;</i>	<i>An instance of an XML element.</i>

215 Terms in ***italic boldface*** are intended to have the meaning defined in the Glossary.

216 Listings of SPML schemas appear like this.

217

218

Example code listings appear like this.

219

## 1.4.4 Namespaces

220

Conventional XML namespace prefixes are used throughout the listings in this specification to stand for their respective namespaces as follows, whether or not a namespace declaration is present in the example:

221

222

223

- The prefix `dsml`: stands for the Directory Services Markup Language namespace **[DSML]**.

224

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

225

226

- The prefix `spml`: stands for the SPMLv2 Core XSD namespace **[SPMLv2-CORE]**.

227

228

- The prefix `spmlasync`: stands for the SPMLv2 Async Capability XSD namespace. **[SPMLv2-ASYNC]**.

229

230

- The prefix `spmlbatch`: stands for the SPMLv2 Batch Capability XSD namespace **[SPMLv2-BATCH]**.

231

232

- The prefix `spmlbulk`: stands for the SPMLv2 Bulk Capability XSD namespace **[SPMLv2-BULK]**.

233

234

- The prefix `spmlpass`: stands for the SPMLv2 Password Capability XSD namespace **[SPMLv2-PASS]**.

235

236

- The prefix `spmlref`: stands for the SPMLv2 Reference Capability XSD namespace **[SPMLv2-REF]**.

237

238

- The prefix `spmlsearch`: stands for the SPMLv2 Search Capability XSD namespace **[SPMLv2-SEARCH]**.

239

240

- The prefix `spmlsuspend`: stands for the SPMLv2 Suspend Capability XSD namespace **[SPMLv2-SUSPEND]**.

241

242

- The prefix `spmlupdates`: stands for the SPMLv2 Updates Capability XSD namespace **[SPMLv2-UPDATES]**.

243

## 2 Concepts

244

SPML Version 2 (SPMLv2) builds on the concepts defined in SPML Version 1.

245

The basic roles of [Requesting Authority \(RA\)](#) and [Provisioning Service Provider \(PSP\)](#) are unchanged. The [core protocol](#) continues to define the basis for interoperable management of [Provisioning Service Objects \(PSO\)](#). However, the concept of [Provisioning Service Target \(PST\)](#) takes on [new importance](#) in SPMLv2.

246

247

248

249

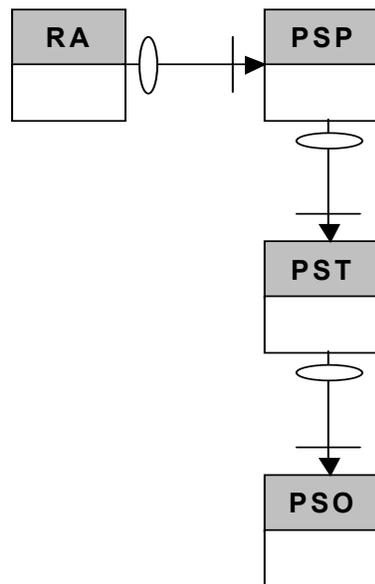
### 2.1 Domain Model

250

The following section describes the main conceptual elements of the SPML domain model. The Entity Relationship Diagram (ERD) in Figure 1 shows the basic relationships between these elements.

251

252



253

254

Figure 1. Domain model elements

255

#### 2.1.1 Requestor

256

A Requesting Authority (RA) or *requestor* is a software component that issues well-formed SPML requests to a [Provisioning Service Provider](#). Examples of requestors include:

257

258

259

- Portal applications that broker the subscription of client requests to system resources
- Service subscription interfaces within an Application Service Provider

260

**Trust relationship.** In an end-to-end integrated provisioning scenario, any component that issues an SPML request is said to be operating as a requestor. This description assumes that the requestor and its provider have established a trust relationship between them. The details of establishing and maintaining this trust relationship are beyond the scope of this specification.

261

262

263

## 264 2.1.2 Provider

265 A Provisioning Service Provider (PSP) or *provider* is a software component that listens for,  
266 processes, and returns the results for well-formed SPML requests from a known [requestor](#). For  
267 example, an installation of an Identity Management system could serve as a provider.

268 **Trust relationship.** In an end-to-end integrated provisioning scenario, any component that  
269 receives and processes an SPML request is said to be operating as a provider. This description  
270 assumes that the provider and its requestor have established a trust relationship between them.  
271 The details of establishing and maintaining this trust relationship are beyond the scope of this  
272 specification.

## 273 2.1.3 Target

274 A Provisioning Service Target (PST) or *target* represents a destination or endpoint that a [provider](#)  
275 makes available for provisioning actions.

276 **A target is not a provider.** A requestor asks a provider to act upon objects that the provider  
277 manages. Each target is a *container* for objects that a provider manages.

278 **A target may not be an actual endpoint.** A target may represent a traditional user account source  
279 (such as a Windows NT domain or a directory service instance), or a target may represent an  
280 abstract collection of endpoints.

281 **Every provider exposes at least one target.** Each target represents a destination or endpoint  
282 (e.g., a system, application or service—or a *set of* systems, applications, and services) to which the  
283 provider can provision (e.g., create or modify accounts).

284 A target is a special, top-level object that:

- 285 • A requestor can [discover from the provider](#)
- 286 • No requestor can add, modify, delete or otherwise act upon
- 287 • May contain any number of [provisioning service objects \(PSO\)](#) upon which a requestor may act
- 288 • May contain a schema that defines the XML structure of the [provisioning service objects \(PSO\)](#)  
289 that the target may contain
- 290 • May define which schema entities the target supports
- 291 • May expose [capabilities](#):
  - 292 - That apply to every supported schema entity
  - 293 - That apply only to specific schema entities

294 The SPMLv2 model does not restrict a provider's targets other than to specify that:

- 295 • A [provider \(PSP\)](#) must uniquely identify each target that it exposes.
- 296 • A provider must uniquely identify each [object \(PSO\)](#) that a target contains.
- 297 • Exactly one target must contain each [object \(PSO\)](#) that the provider manages.

### 298 2.1.3.1 Target Schema

299 The schema for each target defines the XML structure of the [objects \(PSO\)](#) that the target may  
300 contain.

301 SPMLv2 does not specify a required format for the target schema. For example, a target schema  
302 could be XML Schema **[XSD]** or (a target schema could be) SPML1.0 Schema **[SPMLv2-Profile-**  
303 **DSML]**.

304 Each target schema includes a schema namespace. The schema namespace indicates (to any  
305 requestor that recognizes the schema namespace) how to interpret the schema.

306 A provider must present any object (to a requestor) as XML that is valid according to the schema of  
307 the target that contains the object. A requestor must accept and manipulate, as XML that is valid  
308 according to the schema of the target, any object that a target contains.

### 309 **2.1.3.2 Supported Schema Entities**

310 A target may declare that it supports only a subset of the *entities* (e.g., object classes or top-level  
311 elements) in its schema. A target that does not declare such a subset is assumed to support *every*  
312 entity in its schema.

313 A provider must implement the basic SPML operations for any **object** that is an instance of a  
314 supported schema entity (i.e., a schema entity that the target containing the object supports).

### 315 **2.1.3.3 Capabilities**

316 A target may also support a set of capabilities. Each *capability* defines optional operations or  
317 semantics (in addition to the basic operations that the target must support for each supported  
318 schema entity).

319 A capability must be either "standard" or "custom":

320 • The OASIS *PSTC* defines each *standard capability* in an SPML namespace.  
321 See the section titled "[Namespaces](#)".

322 • *Anyone may define a custom capability* in another namespace.

323 A target may support a capability for all of its supported schema entities or (a target may support a  
324 capability) only for specific subset of its supported schema entities. Each capability may specify  
325 any number of supported schema entities to which it applies. A capability that does not specify at  
326 least one supported schema entity *implicitly* declares that the capability applies to every schema  
327 entity that the target supports.

328 **Capability-defined operations.** If a capability defines an operation and if the target supports that  
329 capability for a schema entity of which an object is an instance, then the provider must support that  
330 optional operation for that object. For example, if a target supports the [Password Capability](#) for  
331 User objects (but not for Group objects), then a requestor may ask the provider to perform the  
332 'resetPassword' operation for any User object (but the provider will fail any request to  
333 'resetPassword' for a Group).

334 If a capability defines more than one operation and a target supports that capability (for any set of  
335 schema entities), then the provider must support (for any instance of any of those schema entities  
336 on that target) *every* operation that the capability defines. See the section titled "[Conformance](#)".

337 **Capability-specific data.** A capability may imply that data specific to that capability may be  
338 *associated with* an object. Capability-specific data are *not* part of the schema-defined data of an  
339 object. SPML operations handle capability-specific data separately from schema-defined data.  
340 Any capability that implies capability-specific data must define the structure of that data.  
341 See the section titled "[CapabilityData](#)".

342 Of the capabilities that SPML defines, only one capability actually implies that capability-specific  
343 data may be associated with an object. The Reference Capability implies that an object (that is an  
344 instance of a schema entity for which the provider supports the Reference Capability) may contain  
345 any number of references to other objects. The Reference Capability defines the structure of a  
346 reference element. For more information, see the section titled "[Reference Capability](#)".

## 347 2.1.4 Provisioning Service Object (PSO)

348 A Provisioning Service Object (PSO), sometimes simply called an *object*, represents a data entity  
349 or an information object on a [target](#). For example, a provider would represent as an object each  
350 account that the provider manages.

351 NOTE: Within this document, the term “object” (unless otherwise qualified) refers to a [PSO](#).

352 Every object is contained by exactly one [target](#). Each object has a [unique identifier \(PSO-ID\)](#).

## 353 2.2 Core Protocol

354 SPMLv2 retains the SPML 1.0 concept of a “core protocol”. The SPMLv2 Core XSD defines:

- 355 • *Basic operations* (such as add, lookup, modify and delete)
- 356 • Basic and extensible *data types and elements*
- 357 • The means to expose *individual targets* and *optional operations*

358 The SPMLv2 Core XSD also defines modal mechanisms that allow a requestor to:

- 359 • Specify that a requested operation must be executed asynchronously  
360 (or to specify that a requested operation must be executed synchronously)
- 361 • Recognize that a provider has chosen to execute an operation asynchronously
- 362 • Obtain the status (and any result) of an asynchronous request
- 363 • Stop execution of an asynchronous request

364 Conformant SPMLv2 implementations must support the core protocol, including:

- 365 • The new [listTargets](#) operation
- 366 • The basic operations for [every schema entity that a target supports](#)
- 367 • The modal mechanisms for asynchronous operations

368 (For more information, see the section titled “[Conformance](#)”).

## 369 2.3 Profile

370 SPMLv2 defines two “profiles” in which a requestor and provider may exchange SPML protocol:

- 371 • XML Schema as defined in the “SPMLv2 XSD Profile” [[SPMLv2-Profile-XSD](#)].
- 372 • DSMLv2 as defined in the “SPMLv2 DSMLv2 Profile” [[SPMLv2-Profile-DSML](#)].

373 A requestor and a provider may exchange SPML protocol in any profile to which they agree.

374 SPML 1.0 defined file bindings and SOAP bindings that assumed the SPML 1.0 Schema for DSML  
375 [[SPML-Bind](#)]. The SPMLv2 DSMLv2 Profile provides a *degree of backward compatibility* with  
376 SPML 1.0. The DSMLv2 profile supports a schema model similar to that of SPML 1.0.

377 The DSMLv2 Profile may be more convenient for applications that access mainly targets that are  
378 LDAP or X500 directory services. The XSD Profile may be more convenient for applications that  
379 access mainly targets that are web services.

380

## 3 Protocol

381 **General Aspects.** The general model adopted by this protocol is that a *requestor* (client) asks a  
382 *provider* (server) to perform operations. In the simplest case, each request for an SPML operation  
383 is processed *individually* and is processed *synchronously*. The first sub-section,  
384 “[Request/Response Model](#)”, presents this model and discusses mechanisms that govern  
385 *asynchronous* execution. Sub-sections such as “[Identifiers](#)”, “[Selection](#)”, “[CapabilityData](#)” and  
386 “[Transactional Semantics](#)” also describe aspects of the protocol that apply to every operation.

387 **Core Operations.** In order to encourage adoption of this standard, this specification minimizes the  
388 set of operations that a provider must implement. The [Core Operations](#) section discusses these  
389 *required operations*.

390 **Standard Capabilities.** This specification also defines optional operations. Some operations are  
391 optional (rather than required) because those operations may be more difficult for a provider to  
392 implement for certain kinds of targets. Some operations are optional because those operations may  
393 apply only to specific types of objects on a target. This specification defines a set of standard  
394 capabilities, each of which groups optional operations that are functionally related. The remainder  
395 of the Operations section discusses optional operations (such as [search](#)) that are associated with  
396 SPMLv2's *standard capabilities*.

397 **Custom Capabilities.** The capability mechanism in SPMLv2 is *open* and allows an individual  
398 provider (or any third party) to define additional *custom capabilities*. See the sub-section titled  
399 “[Custom Capabilities](#)”.

### 400 3.1 Request/Response Model

401 The general model adopted by this protocol is that a [requestor](#) (client) asks a [provider](#) (server) to  
402 perform an operation. A requestor asks a provider to perform an operation by sending to the  
403 provider an SPML *request* that describes the operation. The provider examines the request and, if  
404 the provider determines that the request is valid, the provider does whatever is necessary to  
405 implement the requested operation. The provider also returns to the requestor an SPML *response*  
406 that details any status or error that pertains to the request.

```
<complexType name="ExtensibleType">
  <sequence>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"
processContents="lax"/>
  </sequence>
  <anyAttribute namespace="##other" processContents="lax"/>
</complexType>

<simpleType name="ExecutionModeType">
  <restriction base="string">
    <enumeration value="synchronous"/>
    <enumeration value="asynchronous"/>
  </restriction>
</simpleType>

<complexType name="CapabilityDataType">
  <complexContent>
    <extension base="spml:ExtensibleType">
```

```

        <annotation>
            <documentation>Contains elements specific to a
capability.</documentation>
        </annotation>
        <attribute name="mustUnderstand" type="boolean"
use="optional"/>
        <attribute name="capabilityURI" type="anyURI"/>
    </extension>
</complexContent>
</complexType>

<complexType name="RequestType">
    <complexContent>
        <extension base="spml:ExtensibleType">
            <attribute name="requestID" type="xsd:ID" use="optional"/>
            <attribute name="executionMode" type="spml:ExecutionModeType"
use="optional"/>
        </extension>
    </complexContent>
</complexType>

<simpleType name="StatusCodeType">
    <restriction base="string">
        <enumeration value="success"/>
        <enumeration value="failure"/>
        <enumeration value="pending"/>
    </restriction>
</simpleType>

<simpleType name="ErrorCode">
    <restriction base="string">
        <enumeration value="malformedRequest"/>
        <enumeration value="unsupportedOperation"/>
        <enumeration value="unsupportedIdentifierType"/>
        <enumeration value="noSuchIdentifier"/>
        <enumeration value="customError"/>
        <enumeration value="unsupportedExecutionMode"/>
        <enumeration value="invalidContainment"/>
        <enumeration value="unsupportedSelectionType"/>
        <enumeration value="resultSetTooLarge"/>
        <enumeration value="unsupportedProfile"/>
        <enumeration value="invalidIdentifier"/>
        <enumeration value="alreadyExists"/>
        <enumeration value="containerNotEmpty"/>
    </restriction>
</simpleType>

<simpleType name="ReturnDataType">
    <restriction base="string">
        <enumeration value="identifier"/>
        <enumeration value="data"/>
        <enumeration value="everything"/>
    </restriction>
</simpleType>

<complexType name="ResponseType">

```

```

    <complexContent>
      <extension base="spml:ExtensibleType">
        <sequence>
          <element name="errorMessage" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="status" type="spml:StatusCodeType"
use="required"/>
        <attribute name="requestID" type="xsd:ID" use="optional"/>
        <attribute name="error" type="spml:ErrorCode"
use="optional"/>
      </extension>
    </complexContent>
  </complexType>

```

407 The following subsections describe aspects of this request/response model in more detail:

- 408
- 409 • the [exchange of requests and responses](#) between requestor and provider
  - 410 • [synchronous and asynchronous execution](#) of operations
  - 411 • [individual and batch requests](#)

### 411 3.1.1 Conversational flow

412 A requestor asks a provider to do something by issuing an SPML request. A provider responds  
413 exactly once to each request. Therefore, the simplest conversation (i.e., pattern of exchange)  
414 between a requestor and a provider is an orderly alternation of request and response. However, the  
415 SPML protocol does not require this. A requestor may issue any number of concurrent requests to  
416 a single provider. A requestor may issue any number of concurrent requests to multiple providers.

417 **Recommend requestID.** Each SPML request should specify a *reasonably unique* identifier as the  
418 value of "requestID". See the section titled "[Request Identifier \(normative\)](#)". This allows a  
419 requestor to control the identifier for each requested operation and (also allows the requestor) to  
420 match each response to the corresponding request *without relying on the transport protocol* that  
421 underlies the SPML protocol exchange.

### 422 3.1.2 Status and Error codes

423 A provider's response always specifies a "status". This value tells the requestor what the  
424 provider did with (the operation that was described by) the corresponding request.

425 If a provider's response specifies "status='failure'", then the provider's response must also  
426 specify an "error". This value tells the requestor what type of problem prevented the provider  
427 from executing (the operation that was described by) the corresponding request.

428 The "status" and "error" attributes of a response apply to (the operation that is described by)  
429 the corresponding request. This is straightforward for most requests. The status and batch  
430 operations present the only subtleties.

- 431
- 432 • A status request asks for the status of another operation that the provider is *already executing*  
433 *asynchronously*. See the section titled "[Synchronous and asynchronous operations](#)" below. A  
434 status response has status and error attributes that tell the requestor what happened to the  
435 status request itself. However, the response to a successful status operation also contains a  
436 *nested response* that tells what has happened to the operation that the provider is executing  
asynchronously.

- 437 • A batch request contains nested requests (each of which describes an operation). The  
438 response to a batch request contains nested responses (each of which corresponds to a  
439 request that was nested in the batch request). See the section titled "[Individual and batch](#)  
440 [requests](#)" below.

### 441 **3.1.2.1 Status (normative)**

442 A provider's response MUST specify "status" as one of the following values: 'success',  
443 'failure' or 'pending'.

- 444 • A response that specifies "status='success'"  
445 indicates that the provider has completed the requested operation.  
446 In this case, the response contains any result of the operation  
447 and the response MUST NOT specify "error" (see below).
- 448 • A response that specifies "status='failure'"  
449 indicates that the provider could not complete the requested operation.  
450 In this case, the response MUST specify an appropriate value of "error" (see below).
- 451 • A response that specifies "status='pending'"  
452 indicates that the provider will execute the requested operation asynchronously  
453 (see "[Synchronous and asynchronous operations](#)" below).  
454 In this case, the response acknowledges the request and contains the "requestID" value  
455 that identifies the asynchronous operation.

### 456 **3.1.2.2 Error (normative)**

457 A response that specifies "status='failure'" MUST specify an appropriate value of "error".

- 458 • A response that specifies "error='malformedRequest'"  
459 indicates that the provider could not interpret the request.  
460 This includes, but is not limited to, parse errors.
- 461 • A response that specifies "error='unsupportedOperation'"  
462 indicates that the provider does not support the operation that the request specified.
- 463 • A response that specifies "error='unsupportedIdentifierType'"  
464 indicates that the provider does not support the type of identifier specified in the request.
- 465 • A response that specifies "error='noSuchIdentifier'"  
466 indicates that the provider (supports the type of identifier specified in the request,  
467 but the provider) cannot find the object to which an identifier refers.
- 468 • A response that specifies "error='unsupportedExecutionMode'"  
469 indicates that the provider does not support the requested mode of execution.
- 470 • A response that specifies "error='invalidContainment'"  
471 indicates that the provider cannot add the specified object to the specified container.
  - 472 - The request may have specified as container an object that *does not exist*.
  - 473 - The request may have specified as container an object that *is not a valid container*.  
474 The target schema implicitly or explicitly declares each supported schema entity.  
475 An explicit declaration of a supported schema entity specifies  
476 whether an instance of that schema entity may contain other objects.

477 - The request may have specified a container that is *may not contain the specified object*.  
478 The target (or a system or application that underlies the target) may restrict the types of  
479 objects that the provider can add to the specified container. The target (or a system or  
480 application that underlies the target) may restrict the containers to which the provider can  
481 add the specified object.

482 • A response that specifies `"error='resultSetTooLarge'"` indicates that the provider  
483 cannot return (or cannot queue for subsequent iteration—as in the case of an overlarge search  
484 result) the entire result of an operation.

485  
486 In this case, the requestor may be able to refine the request so as to produce a smaller result.  
487 For example, a requestor might break a single search operation into several search requests,  
488 each of which selects a sub-range of the original (overlarge) search result.

489 • A response that specifies `"error='customError'"` indicates that the provider has  
490 encountered an error that none of the standard error code values describes.  
491 In this case, the provider's response SHOULD provide error information in a format that is  
492 available to the requestor. SPMLv2 does not specify the format of a custom error.

493 Several additional values of {ErrorCode} apply only to certain operations. (For example,  
494 `"error='unsupportedProfile'"` applies only to the listTargets operation. Currently,  
495 `"error='invalidIdentifier'"` and `"error='alreadyExists'"` apply only to the add  
496 operation.) The section that discusses each operation also discusses any value of {ErrorCode}  
497 that is specific to that operation.

### 498 **3.1.2.3 Error Message (normative)**

499 A response MAY contain any number of <errorMessage> elements. The XML content of each  
500 <errorMessage> is a string that provides additional information about the status or failure of the  
501 requested operation.

502 • A response that specifies `"status='failure'"` SHOULD contain at least one  
503 <errorMessage> that describes *each condition that caused the failure*.

504 • A response that specifies `"status='success'"` MAY contain any number of  
505 <errorMessage> elements that describe *warning* conditions.

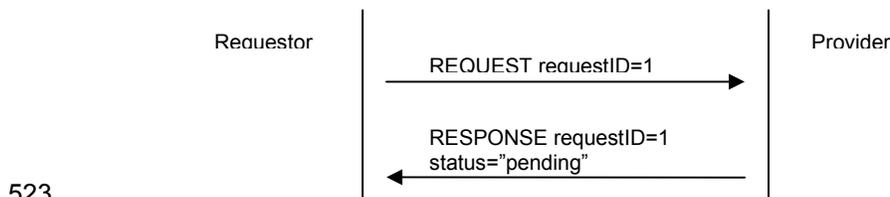
506 • A response that specifies `"status='success'"` SHOULD NOT contain an  
507 <errorMessage> element that describes an *informational* message

508 The content of an <errorMessage> is intended for logging or display to a human administrator  
509 (rather than for programmatic interpretation).

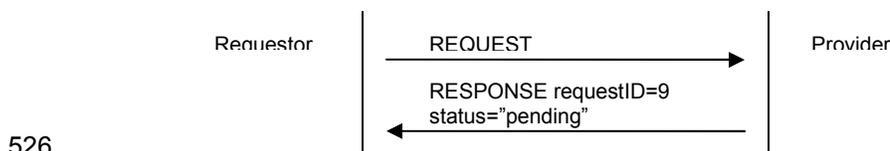
### 510 3.1.3 Synchronous and asynchronous operations

511 A provider may execute a requested operation either *synchronously* or *asynchronously*.

- 512 • **Synchronous: operation before response.** If a provider executes a requested operation  
513 *synchronously*, the provider completes the requested operation before the provider returns a  
514 response to the requestor. The response will include the status and any error or result.
- 515 • **Asynchronous: response before operation.** If a provider executes a requested operation  
516 *asynchronously*, the provider returns to the requestor a response (that indicates that the  
517 operation will be executed asynchronously) before the provider executes the requested  
518 operation. The response will specify "status='pending'" and will specify a "requestID"  
519 value that the requestor must use in order to cancel the asynchronous operation or (in order to)  
520 obtain the status or results of the asynchronous operation.
  - 521 - If a request *specifies* "requestID", then the provider's response to that request will  
522 specify the *same* "requestID" value.



- 524 - If the request *omits* "requestID", then the provider's response to that request will specify  
525 a "requestID" value that is *generated by the provider*.



527 A requestor may specify the execution mode for an operation in its request or (a requestor may  
528 omit the execution mode and thus) allow the provider to decide the execution mode (for the  
529 requested operation). If the requestor specifies an execution mode that the provider cannot support  
530 for the requested operation, then the provider will fail the request.

#### 531 3.1.3.1 ExecutionMode attribute

532 A requestor uses the optional "executionMode" attribute of an SPML request to specify that the  
533 provider must execute the specified operation synchronously or (to specify that the provider must  
534 execute the specified operation) asynchronously. If a requestor omits the "executionMode"  
535 attribute from an SPML request, the provider decides whether to execute the requested operation  
536 synchronously or (to execute the requested operation) asynchronously.

#### 537 3.1.3.2 Async Capability

538 A provider uses the Async Capability that is defined as part of SPMLv2 to tell any requestor that the  
539 provider supports asynchronous execution of requested operations on objects contained by that  
540 target. A target may further refine this declaration to apply *only to specific types of objects* (i.e., for a  
541 specific subset of supported schema entities) on the target.

542 SPMLv2's Async Capability also defines two operations that a requestor may use to manage other  
543 operations that a provider is executing asynchronously:  
544 • A status operation allows a requestor to check the status (and optionally results) of an  
545 operation (or of all operations)  
546 • A cancel operation asks the provider to stop executing an operation.

547 For more information, see the section titled "[Async Capability](#)".

### 548 **3.1.3.3 Determining execution mode**

549 By default, a requestor allows a provider to decide whether to execute a requested operation  
550 synchronously or asynchronously. A requestor that needs the provider to execute a requested  
551 operation in a particular manner must specify this in the request. Each subsection that follows  
552 describes one of the four possibilities:

- 553 • [Requestor specifies synchronous execution](#)
- 554 • [Requestor specifies asynchronous execution](#)
- 555 • [Provider chooses synchronous execution](#)
- 556 • [Provider chooses asynchronous execution](#)

557 The following subsections normatively apply to every SPMLv2 operation unless the normative text  
558 that describes an operation specifies otherwise.

#### 559 **3.1.3.3.1 Requestor specifies synchronous execution (normative)**

560 A requestor *MAY specify* that an operation must execute *synchronously*. A requestor that wants the  
561 provider to execute an operation synchronously *MUST specify*  
562 "executionMode='synchronous'" in the SPML request.

563 If a requestor specifies that an operation must be executed synchronously and the provider cannot  
564 execute the requested operation synchronously, then the provider *MUST* fail the operation. If a  
565 provider fails an operation because the provider cannot execute the operation synchronously, then  
566 the provider's response *MUST specify* "status='failed'" and (the provider's response *MUST*  
567 also specify) "error='unsupportedExecutionMode'".

568 If a requestor specifies that an operation must be executed synchronously and the provider does  
569 not fail the request, then the provider *implicitly agrees* to execute the requested operation  
570 synchronously. The provider *MUST* acknowledge the request with a response that contains any  
571 status and any error or output of the operation. The provider's response *MUST NOT* specify  
572 "status='pending'". The provider's response *MUST specify* either "status='success'" or  
573 "status='failed'".

- 574 • If the provider's response specifies "status='failed'", then the provider's response must  
575 have an "error" attribute.
- 576 • If the provider's response specifies "status='success'", then the provider's response *MUST*  
577 contain any additional results (i.e., output) of the completed operation.

#### 578 **3.1.3.3.2 Requestor specifies asynchronous execution (normative)**

579 A requestor *MAY specify* that an operation must execute *asynchronously*. A requestor that wants  
580 the provider to execute an operation asynchronously *MUST specify*  
581 "executionMode='asynchronous'" in the SPML request.

582 If a requestor specifies that an operation must be executed asynchronously and the provider cannot  
583 execute the requested operation asynchronously, then the provider *MUST* fail the operation. If the

584 provider fails the operation because the provider cannot execute the operation asynchronously,  
585 then the provider's response MUST specify "status='failed'" and (the provider's response  
586 MUST specify) "error='unsupportedExecutionMode'".

587 If a requestor specifies that an operation must be executed asynchronously and the provider does  
588 not fail the request, then the provider *implicitly agrees* to execute the requested operation  
589 asynchronously. The provider MUST acknowledge the request with a synchronous response that  
590 indicates that the operation will execute asynchronously. The provider's response MUST specify  
591 "status='pending'" and (the provider's response MUST specify) "requestID".

592 • If the request specifies a "requestID" value, then the provider's response MUST specify the  
593 same "requestID" value.

594 • If the request omits "requestID", then the provider's response MUST specify a  
595 "requestID" value that uniquely identifies the requested operation within the namespace of  
596 the provider.

597 If the provider's response indicates that the requested operation will execute asynchronously, the  
598 requestor may continue with other processing. If the requestor wishes to obtain the [status and](#)  
599 [results](#) of the requested operation (or to [cancel](#) the requested operation), the requestor MUST use  
600 the "requestID" value that is returned in the provider's response to identify the operation.

601 See also the sections titled "[Async Capability](#)" and "[Results of asynchronous operations](#)  
602 [\(normative\)](#)".

### 603 [3.1.3.3 Provider chooses synchronous execution \(normative\)](#)

604 A requestor MAY allow the provider to decide whether to execute a requested operation  
605 synchronously or asynchronously. A requestor that wants to let the provider decide the type of  
606 execution for an operation MUST omit the "executionMode" attribute of the SPML request.

607 If a requestor lets the provider decide the type of execution for an operation and the provider  
608 *chooses* to execute the requested operation synchronously, then the provider's response MUST  
609 indicate that the requested operation was executed synchronously. The provider's response MUST  
610 NOT specify "status='pending'". The provider's response MUST specify either  
611 "status='success'" or "status='failed'".

612 • If the provider's response specifies "status='failed'", then the provider's response must  
613 have an "error" attribute.

614 • If the provider's response specifies "status='success'", then the provider's response MUST  
615 contain any additional results (i.e., output) of the completed operation.

### 616 [3.1.3.4 Provider chooses asynchronous execution \(normative\)](#)

617 A requestor MAY allow a provider to decide whether to execute a requested operation  
618 synchronously or asynchronously. A requestor that wants to let the provider decide the type of  
619 execution for an operation MUST omit the "executionMode" attribute of the SPML request.

620 If a requestor lets the provider decide the type of execution for an operation and the provider  
621 *chooses* to execute the requested operation *asynchronously*, then the provider's response must  
622 indicate that the requested operation will execute asynchronously. The provider MUST  
623 acknowledge the request with a response that indicates that the operation will execute  
624 asynchronously. The provider's response MUST specify "status='pending'" and (the provider's  
625 response MUST specify) "requestID".

626 • If the request specifies a "requestID" value, then the provider's response MUST specify the  
627 same "requestID" value.

628 • If the request omits "requestID", then the provider's response MUST specify a  
629 "requestID" value that uniquely identifies the requested operation within the namespace of  
630 the provider.

631 If the provider's response indicates that the requested operation will execute asynchronously, the  
632 requestor may continue with other processing. If the requestor wishes to obtain the [status and](#)  
633 [results](#) of the requested operation (or to [cancel](#) the requested operation), the requestor MUST use  
634 the "requestID" value that is returned in the provider's response to identify the operation.

635 See also the sections titled "[Async Capability](#)" and "[Results of asynchronous operations](#)  
636 [\(normative\)](#)".

### 637 **3.1.3.4 Results of asynchronous operations (normative)**

638 A provider that supports asynchronous execution of requested operations MUST maintain the  
639 status and results of each asynchronously executed operation during the period of time that the  
640 operation is executing and for some *reasonable period of time* after the operation completes.  
641 Maintaining this information allows the provider to respond to status requests.

642 A provider that supports asynchronous execution of requested operations SHOULD publish out-of-  
643 band (i.e., make available to requestors in a manner that is not specified by this document) any limit  
644 on the how long after the completion of an asynchronous operation the provider will keep the status  
645 and results of that operation.

### 646 **3.1.4 Individual and batch requests**

647 A requestor generally requests each operation individually. SPMLv2 also defines a capability to  
648 batch requests. If the provider supports this batch capability, a requestor may group any number of  
649 requests (e.g., requests to add, modify or delete) into a single request.

650 **Individual.** The SPMLv2 core protocol allows a requestor to ask a provider to execute an individual  
651 operation. Each request that is part of the SPMLv2 Core XSD asks a provider to perform a single  
652 operation.

653 **Batch.** SPMLv2 defines batch as an optional operation that allows a requestor to combine any  
654 number of requests into a single request. See the section titled "[Batch Capability](#)".

## 655 **3.2 Identifiers**

```
<complexType name="IdentifierType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="ID" type="string" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="PSOIdentifierType">
  <complexContent>
    <extension base="spml:IdentifierType">
      <sequence>
```

```

        <element name="containerID" type="spml:PSOIdentifierType"
minOccurs="0"/>
        </sequence>
        <attribute name="targetID" type="string" use="optional"/>
    </extension>
</complexContent>
</complexType>

```

656 SPMLv2 uses several different types of identifiers.

- 657 • An instance of {xsd:string} identifies a *target*.
- 658     A target identifier must be *unique* within the (namespace of the) provider.
- 659 • An instance of {xsd:ID} identifies a request or an operation.
- 660 • An instance of {PSOIdentifierType} identifies an *object* on a target.
- 661     An instance of {PSOIdentifierType} combines a *target* identifier with an *object* identifier.
- 662     The target identifier MUST be unique within the (namespace of the) provider.
- 663     The object identifier MUST be unique within the (namespace of the) target.

### 664 3.2.1 Request Identifier (normative)

665 **RequestID in a request.** A requestor SHOULD specify a *reasonably unique* value for the  
666 "requestID" attribute in each request. A "requestID" value need not be globally unique. A  
667 "requestID" value needs only to be sufficiently unique to identify each *outstanding* request. (That  
668 is, a requestor SHOULD specify as the value of "requestID" in each SPML request a value that  
669 is sufficiently unique to identify each request *for which the requestor has not yet received the*  
670 *corresponding response*.)

671 A requestor that uses a *transport protocol that is synchronous* (such as SOAP/HTTP) MAY omit  
672 "requestID". The synchronous nature of the transport protocol exchange itself ensures that the  
673 requestor can match the provider's response to the request. (The provider's response will contain  
674 any requestID that is necessary—for example, because the provider executes the requested  
675 operation asynchronously. See the topic named "RequestID in a response" immediately below.)

676 **RequestID in a response.** A provider's response to a request that specifies "requestID" MUST  
677 specify the same "requestID" value.

678 A provider's response to a request that does not specify a value for "requestID" MAY omit the  
679 "requestID" attribute UNLESS the provider executes the requested operation asynchronously.

680 If the provider executes asynchronously (the operation that was described by) a request that  
681 omitted "requestID", then the provider MUST generate a value that uniquely identifies the  
682 operation to the provider and (the provider MUST) specify this value as the value of the  
683 "requestID" attribute in the provider's response. (This allows the requestor to cancel or to obtain  
684 the status of the operation that the provider is executing asynchronously.

685 See the section titled "[Async Capability](#)".)

### 686 3.2.2 Target Identifier (normative)

687 Each of a provider's targets has a string identifier. Within a provider's [listTargets response](#), the  
688 "targetID" attribute of each <target> element specifies this identifier.

689 **TargetID is unique within provider.** Each <target> in a provider's <listTargetsResponse>  
690 MUST specify a value for "targetID" that uniquely identifies the target within the namespace of  
691 the provider.

692 **Wherever targetID occurs** in a request or in a response, the "targetID" must correspond to  
693 one of the provider's targets. (That is, the value of any "targetID" attribute that a request  
694 specifies or (that a request) indirectly contains MUST match the value of the "targetID" attribute  
695 that a <target> element in the provider's <listTargetsResponse> specifies.)

696 If a request contains an invalid "targetID", the provider's response SHOULD specify  
697 "error='noSuchIdentifier'".

### 698 **3.2.3 PSO Identifier (normative)**

699 **PSO Identifier must be unique.** A provider MUST ensure that each object's PSO Identifier is  
700 unique (within the namespace of the provider). Since every instance of {PSOIdentifierType}  
701 also specifies the target that contains the object (see the next topic immediately below), the value  
702 that identifies an object must be unique within the namespace of the target.

703 **TargetID.** Any instance of {PSOIdentifierType} SHOULD specify "targetID".

- 704 • If the provider's <listTargetsResponse> contains only one <target>,  
705 then an instance of {PSOIdentifierType} MAY omit "targetID".
- 706 • If the provider's <listTargetsResponse> contains more than one <target>,  
707 then any instance of {PSOIdentifierType} MUST specify "targetID".  
708 The value of "targetID" MUST identify a valid target. (That is, the value of "targetID"  
709 MUST match the "targetID" of a <target> in the provider's <listTargetsResponse>.  
710 See the section titled "**Target Identifier (normative)**" above.)

711 **containerID.** Any instance of {PSOIdentifierType} MAY contain at most one  
712 <containerID>. Any <containerID> MUST identify an object that exists on the target. (That  
713 is, the content of any <containerID> in an instance of {PSOIdentifierType} MUST match  
714 the <psoID> of an object that exists on a target. In addition, the value of any "targetID"  
715 attribute in the <containerID> element MUST match the value of the "targetID" attribute of  
716 the instance of {PSOIdentifierType} that contains the <containerID>.)

717 **ID.** Any instance of {PSOIdentifierType} MAY specify "ID". This depends on the profile that  
718 the requestor and provider have agreed to use.

- 719 • The DSML Profile and the XML Schema Profile both specify that an instance of  
720 {PSOIdentifierType} MUST specify "ID". The value of "ID" MUST uniquely identify an  
721 object within the namespace of the target that "targetID" specifies.
- 722 • Another profile may specify that an instance of {PSOIdentifierType} MAY omit "ID".

723 **Content depends on profile.** The content of an instance of {PSOIdentifierType} depends on  
724 the profile that a requestor and provider agree to use.

- 725 • Both the DSML profile and the XML Schema Profile specify that an instance of  
726 {PSOIdentifierType} MUST have an "ID" attribute (see the topic immediately above).  
727 Neither the DSML profile nor the XML Schema Profile specifies *XML content* for an instance of  
728 {PSOIdentifierType}.
- 729 • A profile MAY specify XML content for an instance of {PSOIdentifierType}.

730 **Caution: PSO Identifier is mutable.** A provider MAY change the PSO Identifier for an object. For  
731 example, moving an organizational unit (OU) beneath a new parent within a directory service will  
732 change the distinguished name (DN) of the organizational unit. If the provider exposes the  
733 organizational unit as an object and (if the provider exposes) the directory service DN as the  
734 object's PSO Identifier, then this move will change the object's <psoid>.

735 **Recommend immutable PSO Identifier.** A provider SHOULD expose an immutable value (such  
736 as a globally unique identifier or "GUID") as the PSO Identifier for each object. (An immutable PSO  
737 Identifier ensures that a requestor's reference to an object remains valid as long as the object  
738 exists.)

## 739 3.3 Selection

### 740 3.3.1 QueryClauseType

741 SPMLv2 defines a {QueryClauseType} that is used to select objects. Each instance of  
742 {QueryClauseType} represents a selection criterion.

```
<complexType name="QueryClauseType">
  <complexContent>
    <extension base="spml:ExtensibleType">
    </extension>
  </complexContent>
</complexType>
```

743 {QueryClauseType} specifies no element or attribute. This type is a *semantic marker*.

- 744 • Any capability may define elements of (types that extend) QueryClauseType. These query  
745 clause elements allow a requestor to search for objects based on capability-specific data.  
746 (For example, the SPML Reference Capability defines a <hasReference> element  
747 that enables a requestor to query for objects that have a specific reference.  
748 The SPML Suspend Capability also defines an <isActive> element  
749 that enables a requestor to query for objects that are enabled or disabled.)
- 750 • An instance of {SelectionType}, which extends {QueryClauseType}, may *filter a set of*  
751 *objects*. {SelectionType} may also be used to specify a particular element or attribute of an  
752 object. See the section titled “[SelectionType](#)” below.
- 753 • The SPMLv2 Search Capability defines three logical operators that indicate how a provider  
754 should combine selection criteria. Each logical operator is an instance of  
755 {LogicalOperatorType}, which extends {QueryClauseType}.  
756 See the section titled “[Logical Operators](#)” below.

### 757 3.3.2 Logical Operators

758 The SPMLv2 Search Capability defines three *logical operators* that indicate how a provider should  
759 combine selection criteria.

- 760 • The logical operator <and> specifies a *conjunct*  
761 (that is, the <and> is true if and only if *every* selection criterion that the <and> contains is true).
- 762 • The logical operator <or> specifies a *disjunct*  
763 (that is, the <or> is true if *any* selection criterion that the <or> contains is true).
- 764 • The logical operator <not> specifies *negation*  
765 (that is, the <not> is true if and only if the selection criterion that the <not> contains is *false*.)

```
<complexType name="LogicalOperatorType">
  <complexContent>
    <extension base="spml:QueryClauseType">
    </extension>
  </complexContent>
</complexType>

<element name="and" type="spmlsearch:LogicalOperatorType"/>
```

```
<element name="or" type="spmlsearch:LogicalOperatorType"/>
<element name="not" type="spmlsearch:LogicalOperatorType"/>
```

### 766 3.3.3 SelectionType

767 SPMLv2 defines a {SelectionType} that is used in two different ways:

- 768 • An instance of {SelectionType} may *specify an element or attribute of an object*.  
769 For example, the <component> of a <modification> specifies the part of an object that a  
770 [modify](#) operation (or a [bulkModify](#) operation) will change.
- 771 • An instance of {SelectionType} may *filter a set of objects*.  
772 For example, a <query> may contain a <select> that restricts, based on the schema-defined  
773 XML representation of each object, the set of objects that a [search](#) operation returns  
774 (or that a [bulkModify](#) operation changes or that a [bulkDelete](#) operation deletes).

```
<complexType name="SelectionType">
  <complexContent>
    <extension base="spml:QueryClauseType">
      <sequence>
        <element name="namespacePrefixMap"
type="spml:NamespacePrefixMappingType" minOccurs="0"
maxOccurs="unbounded"/>
      </sequence>
      <attribute name="path" type="string" use="required"/>
      <attribute name="namespaceURI" type="string" use="required"/>
    </extension>
  </complexContent>
</complexType>

<element name="select" type="spml:SelectionType"/>
```

775 **SelectionType.** An instance of {SelectionType} has a "path" attribute which value is an  
776 expression. An instance of {SelectionType} also contains a "namespaceURI" attribute that  
777 indicates (to any provider that recognizes the namespace) the language in which the value of the  
778 "path" attribute is expressed.

779 **Namespace Prefix Mappings.** An instance of {SelectionType} may also contain any number  
780 of <namespacePrefixMap> elements (see the [normative section that follows next](#)). Each  
781 <namespacePrefixMap> allows a requestor to specify the URI of an XML namespace that  
782 corresponds to a namespace prefix that occurs (or that may occur) within the value of the "path"  
783 attribute.

#### 784 3.3.3.1 SelectionType in a Request (normative)

785 **namespaceURI.** An instance of {SelectionType} MUST have a "namespaceURI" attribute.  
786 The value of the "namespaceURI" attribute MUST specify the XML namespace of a query  
787 language. (The value of the "path" attribute must be an expression that is valid in this query  
788 language—see below.)

789 **path.** An instance of {SelectionType} MUST have a "path" attribute. The value of the "path"  
790 attribute MUST be an expression that is valid in the query language that the "namespaceURI"  
791 attribute specifies. The "path" value serves different purposes in different contexts.

- 792 • Within a `<modification>` element, the value of the "path" attribute MUST specify a *target*  
793 schema entity (i.e., an element or attribute) of the object that the provider is to modify.
- 794 • Within a `<query>` element, the value of the "path" attribute MUST specify a *filter* that selects  
795 objects based on:  
796 - The presence (or absence) of a specific element or attribute  
797 - The presence (or absence) of a specific value in the content of an element  
798 or (the presence of absence of a specific value) in the value of an attribute

799 The value of the "path" attribute MUST be expressed in terms of elements or attributes that are  
800 valid (according to the schema of the target) for the type of object on which the provider is  
801 requested to operate.

802 **Namespace prefix mappings.** An instance of `{SelectionType}` MAY contain any number of  
803 `<namespacePrefixMap>` elements.

- 804 • Each `<namespacePrefixMap>` MUST have a "prefix" attribute whose value specifies a  
805 namespace prefix (that may occur in the filter expression that is the value of the "path"  
806 attribute).
- 807 • Each `<namespacePrefixMap>` MUST have a "namespace" attribute whose value is the URI  
808 for an XML namespace.

809 A requestor SHOULD use these mappings to define any namespace prefix that the (value of the)  
810 "path" attribute contains.

811 **Depends on profile.** The profile on which a requestor and provider agree may further restrict an  
812 instance of `{SelectionType}`. For example, a particular profile may allow a `<component>` sub-  
813 element within a modification (or a `<select>` sub-element within a query) to specify only *elements*  
814 of a schema entity (and not to specify *attributes* of those elements).

815 Refer to the documentation of each profile for normative specifics.

### 816 3.3.3.2 SelectionType Processing (normative)

817 A provider MUST evaluate an instance of `{SelectionType}` in a manner that is appropriate to  
818 the context in which the instance of `{SelectionType}` occurs:

- 819 • Within a `<modification>` element, a provider must resolve the value of the "path" attribute  
820 to a schema entity (i.e., to an element or attribute) of the object that the provider is to modify.
- 821 • Within a `<query>` element, a provider must evaluate the value of the "path" attribute as a  
822 filter expression that selects objects based on:  
823 - The presence (or absence) of a specific element or attribute  
824 - The presence (or absence) of a specific value in the content of an element  
825 or (the presence of absence of a specific value) in the value of an attribute

826 **Namespace prefix mappings.** A provider SHOULD use any instance of  
827 `<namespacePrefixMap>` that an instance of `{SelectionType}` contains in order to resolve any  
828 namespace prefix that the value of the "path" attribute contains.

829 **Depends on profile.** The profile on which a requestor and provider agree may further restrict (or  
830 may further specify the processing of) an instance of `{SelectionType}`. For example, a  
831 particular profile may allow a `<component>` sub-element within a modification (or a `<select>`  
832 sub-element within a query) to specify only *elements* of a schema entity (and not to specify  
833 *attributes* of those elements).

834 Refer to the documentation of each profile for normative specifics.

### 835 3.3.3.3 SelectionType Errors (normative)

836 A provider's response to a request that contains an instance of {SelectionType}  
837 MUST specify an error if any of the following is true:

- 838 • The provider does not recognize the value of the "namespaceURI" attribute as indicating an  
839 expression language that the provider supports.
- 840 • The provider does not recognize the value of the "path" attribute as an expression that is  
841 valid in the language that the "namespaceURI" attribute specifies.
- 842 • The provider does not recognize the value of a "path" attribute as an expression that refers to  
843 a schema entity (i.e., element or attribute) that is valid according to the schema of the target.
- 844 • The provider does not support the expression that "path" attribute specifies.  
845 (For example, the expression may be too complex or the expression may contain syntax that  
846 the provider does not support.)

847 In all of the cases described above, the provider's response MUST specify either  
848 "error='unsupportedSelectionType'" or "error='customError'".

- 849 • In general, the provider's response SHOULD specify  
850 "error='unsupportedSelectionType' ". The provider's response MAY also contain  
851 instances of <errorMessage> that describe more specifically the problem with the request.
- 852 • However, a provider's response MAY specify "error='customError' "  
853 if the provider's custom error mechanism enables the provider to indicate more specifically  
854 (or to describe more specifically) the problem with the request.

855 **Depends on profile.** The profile on which a requestor and provider agree may further restrict (or  
856 may further specify the errors related to) an instance of {SelectionType}. For example, a  
857 particular profile may allow a <component> sub-element within a modification (or a <select>  
858 sub-element within a query) to specify only *elements* of a schema entity (and not to specify  
859 *attributes* of those elements).

860 Refer to the documentation of each profile for normative specifics.

### 861 3.3.4 SearchQueryType

862 SPMLv2 defines a {SearchQueryType} that is used to select objects on a target.

```
<simpleType name="ScopeType">
  <restriction base="string">
    <enumeration value="pso"/>
    <enumeration value="oneLevel"/>
    <enumeration value="subTree"/>
  </restriction>
</simpleType>

<complexType name="SearchQueryType">
  <complexContent>
    <extension base="spml:QueryClauseType">
      <sequence>
        <annotation>
```

```

      <documentation>Open content is one or more instances of
      QueryClauseType (including SelectionType) or
      LogicalOperator.</documentation>
      </annotation>
      <element name="basePsoID" type="spml:PSOIdentifierType"
minOccurs="0"/>
    </sequence>
    <attribute name="targetID" type="string" use="optional"/>
    <attribute name="scope" type="spmlsearch:ScopeType"
use="optional"/>
  </extension>
</complexContent>
</complexType>

<element name="query" type="spmlsearch:SearchQueryType"/>

```

863 **targetID** specifies the target on which to search for objects.

864 **basePsoID** specifies the starting point for a query. Any <basePsoID> MUST identify an existing  
865 object to use as a *base context* or “root” for the search. That is, a <query> that contains  
866 <basePsoID> may select *only the specified container and objects in that container*.

867 **Scope** indicates whether the query should select the container itself, objects directly contained, or  
868 any object directly or indirectly contained.

869 The “scope” attribute restricts the search operation to one of the following:

- 870 • To the base context itself.
- 871 • To the base context and its direct children.
- 872 • To the base context and any of its descendants.

### 873 3.3.4.1 SearchQueryType in a Request (normative)

874 **targetID**. An instance of {SearchQueryType} MAY specify “targetID”.

- 875 • If the provider's <listTargetsResponse> contains only one <target>,  
876 then a requestor MAY omit the “targetID” attribute of {SearchQueryType}.
- 877 • If the provider's <listTargetsResponse> contains more than one <target>,  
878 then a requestor MUST specify the “targetID” attribute of {SearchQueryType}.

879 **basePsoID**. An instance of {SearchQueryType} MAY contain at most one <basePsoID>.

- 880 • A requestor that wants to search *the entire namespace of a target*  
881 MUST NOT supply <basePsoID>.
- 882 • A requestor that wants to search *beneath a specific object on a target*  
883 MUST supply <basePsoID>. Any <basePsoID> MUST identify an object that exists on the  
884 target. (That is, any <basePsoID> MUST match the <psoID> of an object that already exists  
885 on the target.)

886 **scope**. An instance of {SearchQueryType} MAY have a “scope” attribute. The value of the  
887 “scope” attribute specifies the set of objects against which the provider should evaluate the  
888 <select> element:

- 889 • A requestor that wants the provider to search *only the object* identified by <basePsoID>  
890 MUST specify “scope='pso'”. (NOTE: It is an error to specify “scope='pso'” in An  
891 instance of {SearchQueryType} that does not contain <basePsoID>. The target is not an

- 892 object.)  
 893 See the section titled "[SearchQueryType Errors \(normative\)](#)" below.
- 894 • A requestor that wants the provider to search *only direct descendants* of the target or (that  
 895 wants to search only direct descendants) of the object specified by <basePsoID> MUST  
 896 specify "scope='oneLevel'".
  - 897 • A requestor that wants the provider to search *any direct or indirect descendant* of the target or  
 898 (that wants to search any direct or indirect descendant) of the object specified by  
 899 <basePsoID> MUST specify "scope='subTree'".

900 **Open content.** An instance of {SearchQueryType} MUST contain (as open content) exactly  
 901 one instance of a type that extends {QueryClauseType}.

- 902 • Any capability may define elements of (a type that extends) {QueryClauseType}. These  
 903 elements allow a requestor to select objects based on capability-defined data.  
 904 See the section titled "[QueryClauseType](#)" above.
- 905 • A <select> element is an instance of {SelectionType}, which extends  
 906 {QueryClauseType} to filter objects based on schema-defined content.  
 907 See the section titled "[SelectionType in a Request \(normative\)](#)".
- 908 • Logical Operators such as <and>, <or> and <not> combine individual selection criteria.  
 909 A logical operator MUST contain at least one instance of a type that extends  
 910 {QueryClauseType} or a (logical operator MUST contain at least one) logical operator.  
 911 See the section titled "[Logical Operators](#)" above.

#### 912 **3.3.4.2 SearchQueryType Errors (normative)**

913 The response to a request that contains an instance of {SearchQueryType} (e.g., a <query>  
 914 element) MUST specify an appropriate value of "error" if any of the following is true:

- 915 • The <query> in a <searchRequest> specifies "scope='ps0'" but does not contain  
 916 <basePsoID>. (The target itself is not a PSO.)
- 917 • The "targetID" of the instance of {SearchQueryType} does not specify a valid target.
- 918 • An instance of {SearchQueryType} specifies "targetID" and (the instance of  
 919 {SearchQueryType} also) contains <basePsoID>, but the value of "targetID" in the  
 920 instance of {SearchQueryType} does not match the value of "targetID" in the  
 921 <basePsoID>.
- 922 • An instance of {SearchQueryType} contains a <basePsoID>  
 923 that does not identify an object that exists on a target.  
 924 (That is, the <basePsoID> does not match the <ps0ID> of any object that exists on a target.)
- 925 • The provider cannot evaluate an instance of {QueryClauseType} that the instance of  
 926 {SearchQueryType} contains.
- 927 • The open content of the instance of {SearchQueryType} is too complex for the provider to  
 928 evaluate.
- 929 • The open content of the instance of {SearchQueryType} contains a syntactic error  
 930 (such as an invalid structure of logical operators or query clauses).

- 931 • The provider does not recognize an element of open content that the instance of  
932 {SearchQueryType} contains.

933 Also see the section titled "[SelectionType Errors \(normative\)](#)".

## 934 **3.4 CapabilityData**

935 Any capability may imply that data specific to that capability may be *associated with* an object.  
936 Capability-specific data that is associated with an object is *not* part of the schema-defined data of  
937 an object. SPML operations handle capability-specific data separately from schema-defined data.  
938 Any capability that implies capability-specific data should define the structure of that data. Any  
939 capability that implies capability-specific data may also specify how the core operations should treat  
940 that capability-specific data. See the discussion of "[Capability-specific data](#)" within the section titled  
941 "[Conformance \(normative\)](#)".

942 However, many capabilities will *not* imply any capability-specific data (that may be associated with  
943 an object). Of the standard capabilities that SPMLv2 defines, only the Reference Capability actually  
944 implies that data specific to the Reference Capability may be associated with an object. (The  
945 Suspend Capability supports an <isActive> query clause that allows a requestor to select  
946 objects based on the enablement state of each object, but the <isActive> element is not stored  
947 as <capabilityData> that is associated with an object.)

948 The Reference Capability implies that an object (that is an instance of a schema entity for which the  
949 provider supports the Reference Capability) may contain any number of references to other objects.  
950 The Reference Capability defines the structure of a reference element. The Reference Capability  
951 also specifies how the core operations must treat data specific to the Reference Capability. See the  
952 section titled "[Reference Capability](#)".

### 953 **3.4.1 CapabilityDataType**

954 SPMLv2 defines a {CapabilityDataType} that may occur in a request or in a response. Each  
955 instance of {CapabilityDataType} contains all of the data that is *associated with a particular*  
956 *object and that is specific to a particular capability.*

```
<complexType name="CapabilityDataType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <annotation>
        <documentation>Contains elements specific to a
capability.</documentation>
      </annotation>
      <attribute name="mustUnderstand" type="boolean"
use="optional"/>
      <attribute name="capabilityURI" type="anyURI"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="PSOType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType"/>
        <element name="data" type="spml:ExtensibleType"
minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

```
<element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
</extension>
</complexContent>
</complexType>
```

957 **capabilityURI.** An instance of {CapabilityDataType} has a "capabilityURI" attribute that  
958 identifies a capability. The value of "capabilityURI" must match the value of the  
959 "namespaceURI" attribute of a supported <capability>.

960 **mustUnderstand.** An instance of {CapabilityDataType} may also specify a Boolean value for  
961 "mustUnderstand". This value indicates whether provider must handle the content (of the  
962 instance of {CapabilityDataType}) in a manner that the capability specifies. An instance of  
963 {CapabilityDataType} specifies "mustUnderstand='false'" indicates that default  
964 processing will suffice. (See the next topic below.)

965 The "mustUnderstand" attribute is significant only when a *request* contains the instance of  
966 {CapabilityDataType}.  
967 See the section titled "[CapabilityData in a Request \(normative\)](#)" below.

968 **Default processing.** Each <capabilityData> specifies "capabilityURI" and contains all the  
969 data associated with an object that is specific to that capability.  
970 See the section below titled "[CapabilityData in a Request \(normative\)](#)".

971 By default, a provider treats the set of data specific to each capability as if it were *opaque*. That is,  
972 a provider processes the content of an instance of {CapabilityDataType} *exactly as it is*  
973 without manipulating that content in any way.  
974 See the section titled "[CapabilityData Processing \(normative\)](#)".

975 **Capability-specific processing.** Any capability that implies capability-specific data may specify  
976 how operations should handle the data specific to that capability. Capability-specific handling takes  
977 precedence over the default handling.  
978 See the section titled "[CapabilityData Processing \(normative\)](#)".

### 979 **3.4.1.1 CapabilityData in a Request (normative)**

980 **capabilityURI.** An instance of {CapabilityDataType} MUST specify a value of  
981 "capabilityURI" that identifies a *supported capability*. That is, the (value of the)  
982 "capabilityURI" attribute for an instance of {CapabilityDataType} MUST match the (value  
983 of the) "namespaceURI" attribute of a <capability> the provider supports *for the target* (that  
984 contains the object to be manipulated) and (that the provider supports on that target) *for the*  
985 *schema entity* of which the object to be manipulated is an instance.

986 For normative specifics of supported capabilities,  
987 see the section titled "[listTargetsResponse \(normative\)](#)".

988 **One capabilityData element per capability.** At most one instance of {CapabilityDataType}  
989 within a request MAY refer to a specific capability. That is, a request MUST NOT contain two (and  
990 MUST NOT contain more than two) instances of {CapabilityDataType} that specify the same  
991 value of "capabilityURI".

992 This implies that an instance of {CapabilityDataType} that refers to a certain capability MUST  
993 contain *all the data* within that request *that is specific to that capability* and that is specific to a  
994 particular object.

995 **mustUnderstand.** An instance of {CapabilityDataType} MAY specify "mustUnderstand".  
996 The "mustUnderstand" attribute tells the provider what to do if the provider does not know how  
997 to handle the content of an instance of {CapabilityDataType} in any special manner that the  
998 corresponding capability specifies.

- 999 • A requestor that wants the request to *fail if the provider cannot provide capability-specific*  
1000 *handling* for the set of data specific to a certain capability MUST specify  
1001 "mustUnderstand='true'" on the instance of {CapabilityDataType}  
1002 that contains the data specific to that capability.
- 1003 • A requestor that will *accept default handling* for any data specific to a certain capability MUST  
1004 specify "mustUnderstand='false'" on the instance of {CapabilityDataType} that  
1005 contains the data specific to that capability or (the requestor MUST) omit the  
1006 "mustUnderstand" attribute (from the instance of {CapabilityDataType}  
1007 that contains the data specific to that capability).

1008 The section titled "[CapabilityData Processing \(normative\)](#)" describes the default handling for  
1009 capability-specific data. Any capability for which the default handling is inappropriate MUST specify  
1010 how operations should handle data specific to that capability. The section titled "[Reference](#)  
1011 [CapabilityData Processing \(normative\)](#)" specifies handling of data specific to the Reference  
1012 Capability.

1013 **Capability defines structure.** Any capability that implies capability-specific data SHOULD specify  
1014 the structure of that data. (That is, the capability to which the "capabilityURI" attribute of an  
1015 instance of {CapabilityDataType} refers SHOULD specify the structure of data that the  
1016 instance of {CapabilityDataType} contains.) Furthermore, any capability that implies  
1017 capability-specific data and for which the default processing of capability-specific data is  
1018 inappropriate MUST specify the structure of that capability-specific data and MUST specify how  
1019 operations handle that capability-specific data. See the discussion of "[Capability-specific data](#)"  
1020 within the section titled "[Conformance](#)".

1021 Of the capabilities that SPMLv2 defines, only the Reference Capability implies that capability-  
1022 specific data may be associated with an object. The Reference Capability specifies that an  
1023 instance of {CapabilityDataType} that refers to the Reference Capability  
1024 (e.g., a <capabilityData> element that specifies  
1025 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'"  
1026 MUST contain at least one reference to another object. The Reference Capability defines the  
1027 structure of a <reference> element as {ReferenceType}.) The Reference Capability also  
1028 specifies that each <reference> must match a supported <referenceDefinition>.  
1029 See the section titled "[Reference CapabilityData in a Request \(normative\)](#)".

### 1030 [3.4.1.2 CapabilityData Processing \(normative\)](#)

1031 **capabilityURI.** An instance of {CapabilityDataType} MUST specify a value of  
1032 "capabilityURI" that identifies a *supported capability*. That is, the (value of the)  
1033 "capabilityURI" attribute for an instance of {CapabilityDataType} MUST match the (value  
1034 of the) "namespaceURI" attribute of a <capability> the provider supports *for the target* (that  
1035 contains the object to be manipulated) and (that the provider supports on that target) *for the*  
1036 *schema entity* of which the object to be manipulated is an instance.

1037 For normative specifics of supported capabilities,  
1038 see the section titled "[listTargetsResponse \(normative\)](#)".

1039 **mustUnderstand.** The "mustUnderstand" attribute tells a provider whether the default  
1040 processing of capability-specific data is sufficient for the content of an instance of

1041 {CapabilityDataType}. (The next topic within this section describes the default processing of  
1042 capability-specific data.)

- 1043 • If an instance of {CapabilityDataType} specifies "mustUnderstand='true'", then  
1044 the provider MUST handle the data (that the instance of {CapabilityDataType} contains)  
1045 in the manner that the corresponding capability specifies.  
1046

1047 If the provider cannot handle the data (that the instance of {CapabilityDataType} contains)  
1048 in the manner that the corresponding capability specifies,  
1049 then the provider's response MUST specify "status='failure'".  
1050 See the section titled "[CapabilityData Errors \(normative\)](#)" below.

- 1051 • If an instance of {CapabilityDataType} specifies "mustUnderstand='false'"  
1052 or an instance of {CapabilityDataType} omits "mustUnderstand",  
1053 then a provider MAY handle the data (that the instance of {CapabilityDataType} contains)  
1054 according to the default processing that is described below.

- 1055 - If the provider knows that the corresponding capability (e.g., the Reference Capability)  
1056 specifies special handling, then the provider SHOULD process the data (that the instance  
1057 of {CapabilityDataType} contains) in the manner that the corresponding capability  
1058 specifies.

- 1059 - If the provider knows that the corresponding capability (e.g., the Reference Capability)  
1060 specifies special handling but the provider *cannot provide the special handling* that the  
1061 corresponding capability specifies, then the provider MUST handle the data (that the  
1062 instance of {CapabilityDataType} contains) according to the default processing  
1063 that is described below.

- 1064 - If the provider does not know whether the corresponding capability specifies special  
1065 handling, then the provider MUST handle the data (that the instance of  
1066 {CapabilityDataType} contains) according to the default processing  
1067 that is described below.

1068 **Default processing.** By default, a provider treats the set of data specific to each capability as if it  
1069 were *opaque*. That is, a provider processes the content of an instance of  
1070 {CapabilityDataType} *exactly as it is* --without manipulating that content in any way.

1071 (The provider needs to perform capability-specific processing only if the instance of  
1072 {CapabilityDataType} specifies "mustUnderstand='true'" or if the instance of  
1073 {CapabilityDataType} refers to the Reference Capability. See the topic named  
1074 "mustUnderstand" immediately above within this section.).

- 1075 • If an <addRequest> contains an instance of {CapabilityDataType},  
1076 then the provider MUST associate the instance of {CapabilityDataType} *exactly as it is*  
1077 (i.e., without manipulating its content in any way) with the newly created object.

- 1078 • If a <modification> contains an instance of {CapabilityDataType},  
1079 then the default handling depends on the "modificationMode" of that <modification>  
1080 and also depends on whether an instance of {CapabilityDataType} that specifies the  
1081 same "capabilityURI" is already associated with the object to be modified.

- 1082 - If a <modification> that specifies "modificationMode='add'"  
1083 contains an instance of {CapabilityDataType},  
1084 then the provider MUST *append the content* of the instance of {CapabilityDataType}  
1085 that the <modification> contains *exactly as it is* to (the content of) any instance of  
1086 {CapabilityDataType} that is already associated with the object to be modified

1087 and that specifies the same "capabilityURI".  
1088  
1089 If no instance of {CapabilityDataType} that specifies the same "capabilityURI"  
1090 (as the instance of {CapabilityDataType} that the <modification> contains)  
1091 is already associated with the object to be modified,  
1092 then the provider MUST the associate with the modified object the <capabilityData>  
1093 (that the <modification> contains) *exactly as it is* .

1094 - If a <modification> that specifies "modificationMode='replace'"  
1095 contains an instance of {CapabilityDataType},  
1096 then the provider MUST *replace entirely* any instance of {CapabilityDataType}  
1097 that is already associated with the object to be modified  
1098 and that specifies the same "capabilityURI"  
1099 with the instance of {CapabilityDataType} that the <modification> contains  
1100 *exactly as it is* .  
1101

1102 If no instance of {CapabilityDataType} that specifies the same "capabilityURI"  
1103 (as the instance of {CapabilityDataType} that the <modification> contains)  
1104 is already associated with the object to be modified,  
1105 then the provider MUST the associate with the modified object the <capabilityData>  
1106 (that the <modification> contains) *exactly as it is* .

1107 - If a <modification> that specifies "modificationMode='delete'"  
1108 contains an instance of {CapabilityDataType},  
1109 then the provider MUST *delete entirely* any instance of {CapabilityDataType}  
1110 that is already associated with the object to be modified  
1111 and that specifies the same "capabilityURI"  
1112

1113 If no instance of {CapabilityDataType} that specifies the same "capabilityURI"  
1114 (as the instance of {CapabilityDataType} that the <modification> contains)  
1115 is already associated with the object to be modified, then the provider MUST do nothing.  
1116 In this case, the provider's response MUST NOT specify "status='failure'"  
1117 unless there is some other reason to do so.

1118 **Capability-specific handling.** Any capability that implies capability-specific data and for which the  
1119 default processing of capability-specific data is inappropriate MUST specify how (at least the core)  
1120 operations should process that data. (That is, the capability to which the "capabilityURI"  
1121 attribute of an instance of {CapabilityDataType} refers MUST specify how operations should  
1122 process the data that the instance of {CapabilityDataType} contains if the default processing  
1123 for capability-specific data is inappropriate.)  
1124 See the discussion of "[Capability-specific data](#)" within the section titled "[Conformance](#)".

1125 Of the standard capabilities that SPMLv2 defines, only the Reference Capability implies that  
1126 capability-specific data may be associated with an object. The Reference Capability specifies how  
1127 operations should process the content of an instance of {CapabilityDataType} that specifies  
1128 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'".  
1129 See the section titled "[Reference CapabilityData Processing \(normative\)](#)".

### 1130 3.4.1.3 CapabilityData Errors (normative)

1131 A provider's response to a request that contains an instance of {CapabilityDataType}  
1132 MUST specify an error if any of the following is true:

- 1133 • The instance of {CapabilityDataType} specifies "mustUnderstand='true'"  
1134 and the provider does not recognize the value of the "capabilityURI" attribute  
1135 as identifying a capability that the provider supports *for the target* that contains the object to be  
1136 manipulated *and* that the provider supports *for the schema entity* of which the object to be  
1137 manipulated is an instance.
- 1138 • The instance of {CapabilityDataType} specifies "mustUnderstand='true'"  
1139 and the capability to which its "capabilityURI" refers does not specify the structure of data  
1140 specific to that capability.
- 1141 • The instance of {CapabilityDataType} specifies "mustUnderstand='true'" and the  
1142 capability to which its "capabilityURI" refers does not specify how operations should  
1143 process data specific to that capability.
- 1144 • The request contains two or more instances of {CapabilityDataType} that specify the  
1145 same value of "capabilityURI".

1146 In addition, a provider's response to a request that contains an instance of  
1147 {CapabilityDataType} MAY specify an error if any of the following is true:

- 1148 • The provider does not recognize the value of the "capabilityURI" (that the instance of  
1149 {CapabilityDataType} specifies) as identifying a capability that the provider supports *for*  
1150 *the target* that contains the object to be manipulated *and* that the provider supports *for the*  
1151 *schema entity* of which the object to be manipulated is an instance.  
1152  
1153 Alternatively, the provider MAY perform the default handling as described above  
1154 in the section titled "[CapabilityData Processing \(normative\)](#)".

1155 A provider's response to a request that contains an instance of {CapabilityDataType}  
1156 SHOULD contain an <errorMessage> for each instance of {CapabilityDataType} that the  
1157 provider could not process.

1158 **Capability-specific errors.** Any capability that implies capability-specific data MAY specify  
1159 additional errors related to that data. (That is, the capability to which the "capabilityURI"  
1160 attribute of an instance of {CapabilityDataType} refers MAY specify additional errors related to  
1161 that instance of {CapabilityDataType}.)

1162 Of the capabilities that SPMLv2 defines, only the Reference Capability implies that capability-  
1163 specific data may be associated with an object. The Reference Capability specifies additional  
1164 errors related to any instance of {CapabilityDataType} that refers to the Reference Capability  
1165 See the section titled "[Reference CapabilityData Errors \(normative\)](#)".

### 1166 3.4.1.4 CapabilityData in a Response (normative)

1167 **capabilityURI.** An instance of {CapabilityDataType} MUST specify a value of  
1168 "capabilityURI" that identifies a *supported capability*. That is, the (value of the)  
1169 "capabilityURI" attribute for an instance of {CapabilityDataType} MUST match the (value  
1170 of the) "namespaceURI" attribute of a <capability> the provider supports *for the target* (that  
1171 contains the object to be manipulated) and (that the provider supports on that target) *for the*

1172 *schema entity* of which the object to be manipulated is an instance.  
1173 See the section titled "[listTargetsResponse \(normative\)](#)".

1174 **One per capability.** No more than one instance of {CapabilityDataType} within a response  
1175 may refer to a given capability. That is, a response MUST NOT contain two (and a request MUST  
1176 NOT contain more than two) instances of {CapabilityDataType} that specify the same value of  
1177 "capabilityURI".

1178 This implies that an instance of {CapabilityDataType} that refers to a certain capability MUST  
1179 contain *all the data* within that response *that is specific to that capability* and that is associated with  
1180 a particular object.

1181 **mustUnderstand.** An instance of {CapabilityDataType} within a response MAY specify  
1182 "mustUnderstand". A provider SHOULD preserve any "mustUnderstand" attribute of an  
1183 instance of {CapabilityDataType}. See the discussions of the "mustUnderstand" attribute  
1184 within the sections titled "[CapabilityData in a Request \(normative\)](#)" and "[CapabilityData Processing \(normative\)](#)" above.

1186 **Capability defines structure.** Any capability that implies capability-specific data MUST specify the  
1187 structure of that data. (That is, the capability to which the "capabilityURI" attribute of an  
1188 instance of {CapabilityDataType} refers MUST specify the structure of data that the instance  
1189 of {CapabilityDataType} contains.) See the discussion of "[Custom Capabilities](#)" within the  
1190 section titled "[Conformance](#)".

1191 Of the capabilities that SPMLv2 defines, only the Reference Capability implies that capability-  
1192 specific data may be associated with an object. The Reference Capability specifies that an  
1193 instance of {CapabilityDataType} that refers to the Reference Capability MUST contain at  
1194 least one reference to another object. The Reference Capability defines the structure of a  
1195 <reference> element as {ReferenceType}.) The Reference Capability also specifies that  
1196 each <reference> must match a supported <referenceDefinition>.  
1197 See the section titled "[Reference CapabilityData in a Response \(normative\)](#)".

## 1198 **3.5 Transactional Semantics**

1199 SPMLv2 specifies no transactional semantics. This specification defines no operation that implies  
1200 atomicity. That is, no core operation defines (and no operation that is part of one of SPMLv2's  
1201 standard capabilities defines) a logical unit of work that must be committed or rolled back as a unit.

1202 Provisioning operations are notoriously difficult to undo and redo. For security reasons, many  
1203 systems and applications will not allow certain identity management operations to be fully reversed  
1204 or repeated. (More generally, support for transactional semantics suggests participation in  
1205 externally managed transactions. Such participation is beyond the scope of this specification.)

1206 Any transactional semantics should be defined as a capability (or possibly as more than one  
1207 capability). See the section titled "[Custom Capabilities](#)". A transactional capability would define  
1208 operations that imply atomicity or (would define operations) that allow a requestor to specify  
1209 atomicity.

1210 Any provider that is able to support transactional semantics should then declare its support for such  
1211 a capability as part of the provider's response to the listTargets operation (as the provider would  
1212 declare its support for any other capability).

## 1213 **3.6 Operations**

1214 The first subsection discusses the required [Core Operations](#).

1215 Subsequent subsections discuss any optional operation that is associated with each of the standard  
1216 capabilities:

- 1217 • [Async Capability](#)
- 1218 • [Batch Capability](#)
- 1219 • [Bulk Capability](#)
- 1220 • [Password Capability](#)
- 1221 • [Reference Capability](#)
- 1222 • [Search Capability](#)
- 1223 • [Suspend Capability](#)
- 1224 • [Updates Capability](#)

### 1225 **3.6.1 Core Operations**

1226 Schema syntax for the SPMLv2 core operations is defined in a schema associated with the  
1227 following XML namespace: `urn:oasis:names:tc:SPML:2:0` [**SPMLv2-CORE**]. The Core XSD  
1228 is included as Appendix A to this document.

1229 A conformant provider must implement all the operations defined in the Core XSD. For more  
1230 information, see the section entitled "[Conformance](#)".

1231 The SPMLv2 core operations include:

- 1232 • a *discovery* operation ([listTargets](#)) on the provider
- 1233 • several *basic* operations ([add](#), [lookup](#), [modify](#), [delete](#)) that *apply to objects on a target*

#### 1234 **3.6.1.1 listTargets**

1235 The listTargets operation enables a requestor to determine the set of targets that a provider makes  
1236 available for provisioning and (the listTargets operation also enables a requestor) to determine the  
1237 set of capabilities that the provider supports for each target.

1238 The subset of the Core XSD that is most relevant to the listTargets operation follows.

```
<complexType name="SchemaType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <annotation>
          <documentation>Profile specific schema elements should
be included here</documentation>
        </annotation>
        <element name="supportedSchemaEntity"
type="spml:SchemaEntityRefType" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="ref" type="anyURI" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="SchemaEntityRefType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="targetID" type="string" use="optional"/>
      <attribute name="entityName" type="string" use="optional"/>
      <attribute name="isContainer" type="xsd:boolean"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="CapabilityType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="appliesTo" type="spml:SchemaEntityRefType"
minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="namespaceURI" type="anyURI"/>
      <attribute name="location" type="anyURI" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="CapabilitiesListType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="capability" type="spml:CapabilityType"
minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="TargetType">
  <complexContent>
    <extension base="spml:ExtensibleType">
```

```

        <sequence>
            <element name="schema" type="spml:SchemaType"
maxOccurs="unbounded"/>
            <element name="capabilities"
type="spml:CapabilitiesListType" minOccurs="0"/>
        </sequence>
        <attribute name="targetID" type="string" use="optional"/>
        <attribute name="profile" type="anyURI" use="optional"/>
    </extension>
</complexContent>
</complexType>

<complexType name="ListTargetsRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            </extension>
            <attribute name="profile" type="anyURI" use="optional"/>
        </complexContent>
    </complexType>

<complexType name="ListTargetsResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <sequence>
                <element name="target" type="spml:TargetType"
minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

    <element name="listTargetsRequest"
type="spml:ListTargetsRequestType"/>
    <element name="listTargetsResponse"
type="spml:ListTargetsResponseType"/>

```

1239 **ListTargets must be synchronous.** Because the requestor cannot know (at the time the requestor  
1240 asks to listTargets) whether the provider supports asynchronous execution, the listTargets  
1241 operation must be synchronous.

1242 **ListTargets is not batchable.** Because the requestor cannot know (at the time the requestor asks  
1243 the provider to listTargets) whether the provider supports the batch capability, a requestor must not  
1244 nest a listTargets request in a [batch](#) request.

### 1245 **3.6.1.1.1 listTargetsRequest (normative)**

1246 A requestor **MUST** send a <listTargetsRequest> to a provider in order to ask the provider to  
1247 declare the set of targets that the provider exposes for provisioning operations.

1248 **Execution.** A <listTargetsRequest> **MUST NOT** specify  
1249 "executionMode='asynchronous' ". A <listTargetsRequest> **MUST** specify  
1250 "executionMode='synchronous' " or (a <listTargetsRequest> **MUST**) omit  
1251 "executionMode".

1252 This is because a requestor SHOULD examine each target definition to see whether the target  
1253 supports the Async Capability *before* making a request that specifies  
1254 "executionMode='asynchronous'" (rather than *assuming* that the provider supports  
1255 asynchronous execution of requested operations). Since a requestor typically must perform the  
1256 listTargets operation only once at the beginning of a session, this restriction should not be too  
1257 onerous.

1258 For more information, see the section titled "[Determining execution mode](#)".

1259 **Profile.** a <listTargetsRequest> MAY specify "profile".

1260 Any profile value MUST be a URI (e.g., of an XML namespace) that identifies an SPML profile.

1261 **No required content.** A <listTargetsRequest> requires no sub-element or XML content.

### 1262 [3.6.1.1.2 listTargetsResponse \(normative\)](#)

1263 A provider that receives a <listTargetsRequest> from a requestor that it trusts  
1264 MUST examine the request and (if the request is valid) return to the requestor a list of the targets  
1265 that the provider exposes for provisioning operations.

- 1266 • If a <listTargetsRequest> does not specify a "profile",  
1267 then the <listTargetsResponse> MUST contain every instance of <target>  
1268 that the provider exposes for provisioning operations *regardless of the profile* or profiles  
1269 for (which the provider supports) that target.
- 1270 • If a <listTargetsRequest> specifies a "profile" that the provider supports,  
1271 then the <listTargetsResponse> MUST contain only instances of <target>  
1272 for which the provider supports the specified profile.
- 1273 • If a <listTargetsRequest> specifies a "profile" that the provider *does not support*,  
1274 then the <listTargetsResponse> MUST specify "status='failure'".  
1275 See the topic named "Error" below within this section.

1276 **Execution.** A provider MUST execute a listTargets operation synchronously. This is because a  
1277 provider must allow the requestor to examine each target definition to see whether the target  
1278 supports the Async Capability (and thus whether the provider might choose to execute a requested  
1279 operation asynchronously) *before* the provider chooses to execute a requested operation  
1280 asynchronously. Since a requestor typically must perform the listTargets operation only once at the  
1281 beginning of a session, this restriction should not be too onerous.

1282 If a requestor specifies "executionMode='asynchronous'", a provider MUST fail the  
1283 operation with "error='unsupportedExecutionMode'".

1284 For more information, see the section titled "[Determining execution mode](#)".

1285 **Status.** A <listTargetsResponse> MUST have a "status" attribute that indicates whether  
1286 the provider successfully processed the request. See the section titled "[Status \(normative\)](#)".

1287 **Error.** If the provider cannot return a list of its targets, then the <listTargetsResponse> MUST  
1288 contain an error attribute that characterizes the failure.  
1289 See the general section titled "[Error \(normative\)](#)".

1290 In addition, the <listTargetsResponse> MUST specify an appropriate value of "error" if any  
1291 of the following is true:

1292 • The `<listTargetsRequest>` specifies a "profile" and the provider cannot return at least  
1293 one `<target>` that supports the specified profile. In this case, the  
1294 `<listTargetsResponse>` SHOULD specify "error='unsupportedProfile'".

1295 **Target.** A `<listTargetsResponse>` that specifies "status='success'" MUST contain at  
1296 least one `<target>` element. Each `<target>` SHOULD specify "targetID".

1297 • If the `<listTargetsResponse>` contains only one `<target>`  
1298 then the `<target>` MAY omit "targetID".

1299 • If the `<listTargetsResponse>` contains more than one `<target>`  
1300 then each `<target>` MUST specify "targetID".

1301 Any value of "targetID" MUST identify each target uniquely within the namespace of the  
1302 provider.

1303 **Target profile.** Any `<target>` MAY specify "profile". Any "profile" value MUST be a URI  
1304 (e.g., of an XML namespace) that identifies a specific SPML [profile](#).

1305 If a `<target>` specifies a "profile", then the provider MUST support for that target  
1306 (and for any objects on that target) the behavior that the SPML profile specifies.  
1307 Refer to the documentation of each profile for normative specifics.

1308 **Schema.** A `<target>` MUST contain at least one `<schema>` element. Each `<schema>` element  
1309 MUST contain (or each `<schema>` element MUST refer to) some form of XML Schema that defines  
1310 the structure of XML objects on that target.

1311 **Schema content.** Each `<spml:schema>` element MAY include any number of `<xsd:schema>`  
1312 elements.

1313 • If an `<spml:schema>` element contains no `<xsd:schema>` element,  
1314 then that `<spml:schema>` element MUST have a valid "ref" attribute (see below).

1315 • If an `<spml:schema>` element contains at least one `<xsd:schema>` element,  
1316 then this takes precedence over the value of any "ref" attribute of that `<spml:schema>`.  
1317 In this case, the requestor SHOULD ignore the value of any "ref" attribute.

1318 Each `<xsd:schema>` element (that an `<spml:schema>` element contains)  
1319 MUST include the XML namespace of the schema.

1320 **Schema ref.** Each `<spml:schema>` MAY have a "ref" attribute.  
1321 If an `<spml:schema>` has a "ref" attribute, then:

1322 • The "ref" value MUST be a URI that uniquely *identifies* the schema.  
1323 • The "ref" value MAY be a *location* of a schema document  
1324 (e.g. the physical URL of an XSD file).

1325 A requestor should ignore any "ref" attribute of an `<spml:schema>` element that contains an  
1326 `<xsd:schema>`. (See the topic named "Schema content" immediately above.)

1327 **Supported Schema Entities.** A target MAY declare as part of its `<spml:schema>` the set of  
1328 schema entities for which the target supports the basic SPML operations (i.e., [add](#), [lookup](#), [modify](#)  
1329 and [delete](#)). The target `<spml:schema>` MAY contain any number of  
1330 `<supportedSchemaEntity>` elements. Each `<supportedSchemaEntity>` MUST refer to an  
1331 entity in the target schema. (See the topics named "SupportedSchemaEntity entityName" and  
1332 "SupportedSchemaEntity targetID" below within this section.)

1333 A provider that *explicitly* declares a set of schema entities that a target supports has *implicitly*  
1334 declared that the target supports *only* those schema entities. If a target schema contains at least  
1335 one <supportedSchemaEntity>, then the provider MUST support the basic SPML operations  
1336 for (objects on that target that are instances of) any target schema entity to which a  
1337 <supportedSchemaEntity> refers.

1338 A provider that does not *explicitly* declare as part of a target at least one schema entity that the  
1339 target supports has *implicitly* declared that the target supports *every* schema entity. If a target  
1340 schema contains no <supportedSchemaEntity>, then the provider MUST support the basic  
1341 SPML operations for (objects on that target that are instances of) *any* top-level entity in the target  
1342 schema.

1343 A provider SHOULD explicitly declare the set of schema entities that each target supports. In  
1344 general, the syntactic convenience of omitting the declaration of supported schema entities (and  
1345 thereby implicitly declaring that the provider supports all schema entities) does not justify the  
1346 burden that this imposes on each requestor. When a provider omits the declaration of supported  
1347 schema entities, each requestor must determine the set of schema entities that the target supports.  
1348 This process is especially laborious for a requestor that functions without prior knowledge.

1349 **SupportedSchemaEntity entityName.** Each <supportedSchemaEntity> MUST refer to an  
1350 entity in the schema (of the target that contains the <supportedSchemaEntity>):

- 1351 • In the XSD Profile [**SPMLv2-Profile-XSD**], each <supportedSchemaEntity> MUST specify  
1352 a QName (as the value of its "entityName" attribute).
- 1353 • In the DSMLv2 Profile [**SPMLv2-Profile-DSML**], each <supportedSchemaEntity> MUST  
1354 specify the name of an `objectclass` (as the value of its "entityName" attribute).

1355 **SupportedSchemaEntity targetID.** A <supportedSchemaEntity> SHOULD specify a  
1356 "targetID".

- 1357 • A provider MAY omit "targetID" in any <supportedSchemaEntity>.  
1358 (That is, a provider MAY omit the optional "targetID" attribute of  
1359 {SchemaEntityRefType} in a <supportedSchemaEntity> element.)
- 1360 • Any "targetID" in a <supportedSchemaEntity> MUST refer to the containing target.  
1361 (That is, the value of any "targetID" attribute that a <supportedSchemaEntity> specifies  
1362 MUST match the value of the "targetID" attribute of the <target> element that contains  
1363 the <supportedSchemaEntity> element.)

1364 **SupportedSchemaEntity isContainer.** A <supportedSchemaEntity> MAY have an  
1365 "isContainer" attribute that specifies whether an (object that is an) instance of the supported  
1366 schema entity may contain other objects.

- 1367 • If a <supportedSchemaEntity> specifies "isContainer='true'", then a provider  
1368 MUST allow a requestor to add an object beneath any instance of the schema entity.
- 1369 • If a <supportedSchemaEntity> specifies "isContainer='false'"  
1370 (or if a <supportedSchemaEntity> does not specify "isContainer"), then a provider  
1371 MUST NOT allow a requestor to add an object beneath any instance of the schema entity.

1372 **Capabilities.** A target may also declare a set of capabilities that it supports. Each capability defines  
1373 optional operations or semantics. For general information, see the subsection titled "[Capabilities](#)"  
1374 within the "[Concepts](#)" section.

1375 A <target> element MAY contain at most one <capabilities> element. A <capabilities>  
1376 element MAY contain any number of <capability> elements.

1377 **Capability.** Each <capability> declares support for exactly one capability:

- 1378 • Each <capability> element MUST specify (as the value of its "namespaceURI" attribute)  
1379 an XML namespace that *identifies* the capability.
- 1380 • Each <capability> element MAY specify (as the value of its "location" attribute) the URL  
1381 of an XML schema that defines any structure that is associated with the capability  
1382 (e.g., an SPML request/response pair that defines an operation—see below).
- 1383 **Capability operations.** An XML schema document that a capability "location" attribute  
1384 specifies MAY define operations. An XML schema document for a capability MUST define any  
1385 operation as a paired request and response such that both of the following are true:
- 1386 • The (XSD type of the) request (directly or indirectly) extends {RequestType}  
1387 • The (XSD type of the) response (directly or indirectly) extends {ResponseType}
- 1388 **Capability appliesTo.** A target may support a capability for *all* of the target's supported schema  
1389 entities or only for *a specific subset* of the target's supported schema entities. Each [capability](#)  
1390 element may specify any number of supported schema entities to which it applies. A capability that  
1391 does not specify a supported schema entity to which it applies must apply to every supported  
1392 schema entity.
- 1393 A <capability> element MAY contain any number of <appliesTo> elements.
- 1394 A <capability> element that contains no <appliesTo> element MUST apply to every schema  
1395 entity that the target supports. If the XML schema for the capability defines an operation, the  
1396 provider MUST support the capability-defined operation for (any object that is instance of) any  
1397 schema entity that the target supports. If the capability implies semantic meaning, then the provider  
1398 MUST apply that semantic meaning to (every object that is an instance of) any schema entity that  
1399 the target supports.
- 1400 **Capability appliesTo entityName.** Each <appliesTo> element MUST have an "entityName"  
1401 attribute that refers to a supported schema entity of the containing target. (See the topic named  
1402 "Supported Schema Entities entityName" earlier in this section.)
- 1403 • In the XSD Profile, each <appliesTo> element MUST specify a QName  
1404 (as the value of its "entityName" attribute).
- 1405 • In the DSMLv2 Profile [**SPMLv2-Profile-DSML**], each <appliesTo> element MUST specify  
1406 the name of an `objectclass` (as the value of its "entityName" attribute).
- 1407 An <appliesTo> element MAY have a "targetID" attribute.
- 1408 • A provider MAY omit "targetID" in any <appliesTo>.  
1409 (That is, a provider MAY omit the optional "targetID" attribute of  
1410 {SchemaEntityRefType} in an <appliesTo> element.)
- 1411 • Any "targetID" MUST refer to the containing target.  
1412 (That is, any "targetID" attribute of an <appliesTo> element  
1413 MUST contain the same value as the "targetID" attribute  
1414 of the <target> element that contains the <appliesTo> element.)
- 1415 **Capability content.** SPMLv2 specifies only the optional <appliesTo> element as content for  
1416 most capability elements. However, a declaration of support for the reference capability is special.
- 1417 **Reference Capability content.** A <capability> element that refers to the Reference Capability  
1418 (i.e., any <capability> element that specifies  
1419 "namespaceURI='urn:oasis:names:tc:SPML:2.0:reference'")  
1420 MUST contain (as open content) at least one <referenceDefinition> element.  
1421 (For normative specifics, please see the topic named "Reference Definition" immediately below.

1422 For background and for general information, please see the section titled "[Reference Capability](#)".  
1423 For Reference Capability XSD, please see Appendix F.)

1424 **ReferenceDefinition.** Each <referenceDefinition> element MUST be an instance of  
1425 {spmlref:ReferenceDefinitionType}. Each reference definition names a type of reference,  
1426 specifies a "from" schema entity and specifies a set of "to" schema entities. Any instance of the  
1427 "from" schema entity may refer to any instance of any "to" schema entity using the type of reference  
1428 that the reference definition names.

1429 **ReferenceDefinition typeOfReference.** Each <referenceDefinition> element MUST have a  
1430 "typeOfReference" attribute *that names the type of reference*.

1431 **ReferenceDefinition schemaEntity.** Each <referenceDefinition> element MUST contain  
1432 exactly one <schemaEntity> sub-element that specifies a "*from*" schema entity for that type of  
1433 reference.

- 1434 • The <schemaEntity> MUST have an "entityName" attribute that refers to a supported  
1435 schema entity of the containing target. (See topic named the "Supported Schema Entities"  
1436 earlier in this section.)
- 1437 • The <schemaEntity> MAY have a "targetID" attribute. Any "targetID" that the  
1438 <schemaEntity> specifies MUST refer to the containing target.  
1439 (That is, any "targetID" value that a <schemaEntity> specifies  
1440 MUST match the value of the "targetID" attribute of the <target> element  
1441 that contains the <referenceDefinition>.)

1442 **ReferenceDefinition canReferTo.** Each <referenceDefinition> element MAY contain any  
1443 number of <canReferTo> sub-elements, each of which specifies a valid "*to*" schema entity. A  
1444 <referenceDefinition> element that contains no <canReferTo> element implicitly declares  
1445 that *any instance of any schema entity on any target* is a valid "to" schema entity.

- 1446 • A <canReferTo> element MUST have an "entityName" attribute that refers to a supported  
1447 schema entity. The value of the "entityName" attribute MUST be the name of a top-level  
1448 entity that is valid in the schema.

- 1449 • A <canReferTo> element SHOULD have a "targetID" attribute.

- If the <listTargetsResponse> contains only one <target>,  
1451 then any <canReferTo> element MAY omit "targetID".

- If the <listTargetsResponse> contains more than one <target>,  
1453 then any <canReferTo> element MUST specify "targetID".

- If the <canReferTo> element specifies "targetID",  
1455 then the "entityName" attribute (of the <canReferTo> element)  
1456 MUST refer to a supported schema entity of the specified target  
1457 (i.e., the <target> whose "targetID" value matches  
1458 the "targetID" value that the <canReferTo> element specifies).

- If the <canReferTo> element does not specify "targetID",  
1460 then the "entityName" attribute (of the <canReferTo> element)  
1461 MUST refer to a supported schema entity of the containing target  
1462 (i.e., the <target> that contains the <referenceDefinition>).

1463 **ReferenceDefinition referenceDataType.** Each <referenceDefinition> element MAY  
1464 contain any number of <referenceDataType> sub-elements, each of which specifies a *schema*  
1465 *entity that is a valid structure for reference data*. A <referenceDefinition> element that

- 1466 contains no <referenceDataType> element implicitly declares that an instance of that type of  
1467 reference will never contain reference data.
- 1468 • A <referenceDataType> element MUST have an "entityName" attribute that refers to a  
1469 supported schema entity. The value of the "entityName" attribute MUST be the name of a  
1470 top-level entity that is valid in the schema.
  - 1471 • A <referenceDataType> element SHOULD have a "targetID" attribute.
    - 1472 - If the <listTargetsResponse> contains only one <target>,
   
1473 then any <referenceDataType> element MAY omit "targetID".
    - 1474 - If the <listTargetsResponse> contains more than one <target>,
   
1475 then any <referenceDataType> element MUST specify "targetID".
    - 1476 - If the <referenceDataType> element specifies "targetID",
   
1477 then the "entityName" attribute (of the <canReferTo> element)
   
1478 MUST refer to a supported schema entity of the specified target
   
1479 (i.e., the <target> whose "targetID" value matches
   
1480 the "targetID" value that the <referenceDataType> element specifies).
    - 1481 - If the <referenceDataType> element does not specify "targetID",
   
1482 then the "entityName" attribute (of the <canReferTo> element)
   
1483 MUST refer to a supported schema entity of the containing target
   
1484 (i.e., the <target> that contains the <referenceDefinition>).

### 1485 3.6.1.1.3 *listTargets Examples (non-normative)*

1486 In the following example, a requestor asks a provider to list the [targets](#) that the provider exposes for  
1487 provisioning operations.

```
<listTargetsRequest/>
```

1488 The provider returns a <listTargetsResponse>. The "status" attribute of the  
1489 <listTargetsResponse> element indicates that the listTargets request was successfully  
1490 processed. The <listTargetsResponse> contains two <target> elements. Each <target>  
1491 describes an endpoint that is available for provisioning operations.

1492 The requestor did not specify a profile, but both targets specify the XSD profile [**SPMLv2-Profile-  
1493 XSD**]. The requestor must observe the conventions that the XSD profile specifies in order to  
1494 manipulate an object on either target.

1495 If the requestor had specified the DSML profile, then the response would have contained a different  
1496 set of targets (or would have specified "error='unsupportedProfile'").

```
<listTargetsResponse status="success">
  <target targetID="target1" profile="urn:oasis:names:tc:SPML:2.0:profiles:XSD">
    <schema>
      <xsd:schema targetNamespace="urn:example:schema:target1"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:spml="urn:oasis:names:tc:SPML:2.0" elementFormDefault="qualified">
        <complexType name="Account">
          <sequence>
            <element name="description" type="string" minOccurs="0"/>
          </sequence>
          <attribute name="accountName" type="string" use="required"/>
        </complexType>
      </xsd:schema>
    </schema>
  </target>
</listTargetsResponse>
```

```

    </complexType>
    <complexType name="Group">
      <sequence>
        <element name="description" type="string" minOccurs="0"/>
      </sequence>
      <attribute name="groupName" type="string" use="required"/>
    </complexType>
  </xsd:schema>
  <supportedSchemaEntity entityName="Account"/>
  <supportedSchemaEntity entityName="Group"/>
</schema>
<capabilities>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:bulk"/>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:search"/>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:password">
    <appliesTo entityName="Account"/>
  </capability>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:suspend">
    <appliesTo entityName="Account"/>
  </capability>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:reference">
    <appliesTo entityName="Account"/>
    <referenceDefinition typeOfReference="owner">
      <schemaEntity entityName="Account"/>
      <canReferTo entityName="Person" targetID="target2"/>
    </referenceDefinition>
    <referenceDefinition typeOfReference="memberOf">
      <schemaEntity entityName="Account"/>
      <canReferTo entityName="Group"/>
    </referenceDefinition>
  </capability>
</capabilities>
</target>

  <target targetID="target2" profile="urn:oasis:names:tc:SPML:2.0:profiles:XSD">
    <schema>
  <xsd:schema targetNamespace="urn:example:schema:target2"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:spml="urn:oasis:names:tc:SPML:2.0" elementFormDefault="qualified">
    <complexType name="Person">
      <sequence>
        <element name="dn" type="string"/>
        <element name="email" type="string" minOccurs="0"/>
      </sequence>
      <attribute name="cn" type="string" use="required"/>
      <attribute name="firstName" type="string" use="required"/>
      <attribute name="lastName" type="string" use="required"/>
      <attribute name="fullName" type="string" use="required"/>
    </complexType>
    <complexType name="Organization">
      <sequence>
        <element name="dn" type="string"/>
        <element name="description" type="string" minOccurs="0"/>
      </sequence>

```

```

        <attribute name="cn" type="string" use="required"/>
    </complexType>
    <complexType name="OrganizationalUnit">
        <sequence>
            <element name="dn" type="string"/>
            <element name="description" type="string" minOccurs="0"/>
        </sequence>
        <attribute name="cn" type="string" use="required"/>
    </complexType>
</xsd:schema>
    <supportedSchemaEntity entityName="Person"/>
    <supportedSchemaEntity entityName="Organization" isContainer="true"/>
    <supportedSchemaEntity entityName="OrganizationalUnit" isContainer="true"/>
</schema>
<capabilities>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:bulk"/>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:search"/>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:password">
        <appliesTo entityName="Person"/>
    </capability>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:suspend">
        <appliesTo entityName="Person"/>
    </capability>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:reference">
        <appliesTo entityName="Person"/>
        <referenceDefinition typeOfReference="owns">
            <schemaEntity entityName="Person"/>
            <canReferTo entityName="Account" targetID="target1"/>
        </referenceDefinition>
    </capability>
</capabilities>
</target>
</listTargetsResponse>

```

1497 This example `<listTargetsResponse>` contains two instances of `<target>` that are named  
1498 `target1` and `target2`. Each of these targets contains a simple schema.

1499 The schema for `target1` defines two entities: `Account` and `Group`. The schema for `target1`  
1500 declares each of these entities as a supported schema entity. The provider declares that `target1`  
1501 supports the Bulk capability and Search capability for both `Account` and `Group`. The provider also  
1502 declares that `target1` supports the Password, Suspend, and Reference capabilities for `Account`.

1503 The schema for `target2` defines three entities: `Person`, `Organization` and  
1504 `OrganizationalUnit`. The schema for `target2` declares each of these entities as a supported  
1505 schema entity. The provider declares that `target2` supports the Bulk capability and Search  
1506 capability for all three schema entities. The provider also declares that `target2` supports the  
1507 Password, Suspend, and Reference capabilities for instances of `Person` (but not for instances of  
1508 `Organization` or `OrganizationalUnit`).

1509 **Reference Definitions.** Within `target1`'s declaration of the Reference Capability for `Account`,  
1510 the provider also declares two types of references: `owner` and `memberOf`. The provider declares  
1511 that an instance of `Account` on `target1` may refer to an instance of `Person` on `target2` as its  
1512 `owner`. An instance of `Account` on `target1` may also use a `memberOf` type of reference to refer  
1513 to an instance of `Group` on `target1`.

1514 Within `target2`'s declaration of the Reference Capability for `Person`, the provider declares that a  
1515 `Person` on `target2` may own an `Account` on `target1`. (That is, an instance of `Person` on  
1516 `target2` may use an "owns" type of reference to refer to an instance of `Account` on `target1`.)  
1517 Note that the "owns" type of reference *may be* (but is not necessarily) an inverse of the "owner"  
1518 type of reference. For more information, please see the section titled "Reference Capability".

1519 **NOTE:** Subsequent examples within this section will build on this example, using the target  
1520 definitions returned in this example. Examples will also build upon each other. An object that is  
1521 created in the example of the add operation will be modified or deleted in later examples.

### 1522 3.6.1.2 add

1523 The add operation enables a requestor to create a new object on a target and (optionally) to bind  
1524 the object beneath a specified parent object (thus forming a hierarchy of containment).

1525 The subset of the Core XSD that is most relevant to the add operation follows.

```
<complexType name="CapabilityDataType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <annotation>
        <documentation>Contains elements specific to a
capability.</documentation>
      </annotation>
      <attribute name="mustUnderstand" type="boolean"
use="optional"/>
      <attribute name="capabilityURI" type="anyURI"/>
    </extension>
  </complexContent>
</complexType>

<simpleType name="ReturnDataType">
  <restriction base="string">
    <enumeration value="identifier"/>
    <enumeration value="data"/>
    <enumeration value="everything"/>
  </restriction>
</simpleType>

<complexType name="PSOType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType" />
        <element name="data" type="spml:ExtensibleType"
minOccurs="0" />
        <element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="AddRequestType">
  <complexContent>
```

```

        <extension base="spml:RequestType">
            <sequence>
                <element name="psoID" type="spml:PSOIdentifierType"
minOccurs="0"/>
                <element name="containerID" type="spml:PSOIdentifierType"
minOccurs="0"/>
                <element name="data" type="spml:ExtensibleType"/>
                <element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded" />
            </sequence>
            <attribute name="targetID" type="string" use="optional">
            <attribute name="returnData" type="spml:ReturnDataType"
use="optional" default="everything"/>
        </extension>
    </complexContent>
</complexType>

    <complexType name="AddResponseType">
        <complexContent>
            <extension base="spml:ResponseType">
                <sequence>
                    <element name="pso" type="spml:PSOType" minOccurs="0"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>

    <element name="addRequest" type="spml:AddRequestType"/>
    <element name="addResponse" type="spml:AddResponseType"/>

```

### 1526 [3.6.1.2.1 addRequest \(normative\)](#)

1527 A requestor **MUST** send an `<addRequest>` to a provider in order to (ask the provider to) create a  
1528 new object.

1529 **Execution.** A `<addRequest>` **MAY** specify "executionMode".  
1530 See the section titled "[Determining execution mode](#)".

1531 **TargetID.** An `<addRequest>` **SHOULD** specify "targetID".

- 1532 • If the provider exposes only one target in its `<listTargetsResponse>`,
- 1533 then a requestor **MAY** omit the "targetID" attribute of an `<addRequest>`.
- 1534 • If the provider exposes more than one target in its `<listTargetsResponse>`,
- 1535 then a requestor **MUST** specify the "targetID" attribute of an `<addRequest>`.
- 1536 Any "targetID" value must specify a valid target. (That is, the value of any "targetID" in
- 1537 an `<addRequest>` **MUST** match the "targetID" of a `<target>` that is contained in the
- 1538 provider's `<listTargetsResponse>`.)

1539 **psoID.** An `<addRequest>` **MAY** contain a `<psoID>`. (A requestor supplies `<psoID>` in order to  
1540 specify an identifier for the new object. See the section titled "[PSO Identifier \(normative\)](#)".)

1541 **ContainerID.** An `<addRequest>` **MAY** contain a `<containerID>`. (A requestor supplies  
1542 `<containerID>` in order to specify an existing object under which the new object should be  
1543 bound.)

- 1544 • A requestor that wants to bind a new object *in the top-level namespace of a target*  
1545 MUST NOT supply <containerID>.
- 1546 • A requestor that wants to bind a new object *beneath a specific object on a target*  
1547 MUST supply <containerID>. Any <containerID> must identify an existing object.  
1548 (That is, the content of <containerID> in an <addRequest> must match the <psoid> of an  
1549 object that already exists on the target.)
- 1550 **Data.** An <addRequest> MUST contain a <data> element that supplies initial content for the new  
1551 object. A <data> element MUST contain only elements and attributes defined by the target  
1552 schema as valid for the schema entity of which the object to be added is an instance.
- 1553 **CapabilityData.** An <addRequest> element MAY contain any number of <capabilityData>  
1554 elements. (Each <capabilityData> element contains data specific to a single capability. Each  
1555 <capabilityData> element may contain any number of items of capability-specific data.  
1556 Capability-specific data need not be defined by the target schema as valid for schema entity of  
1557 which the object to be added is an instance.  
1558 See the section titled "[CapabilityData in a Request \(normative\)](#)".
- 1559 **ReturnData.** An <addRequest> MAY have a "returnData" attribute that tells the provider  
1560 which types of data to include in the provider's response.
- 1561 • A requestor that wants the provider to return *nothing* of the added object  
1562 MUST specify "returnData='nothing'".
- 1563 • A requestor that wants the provider to return *only the identifier* of the added object  
1564 MUST specify "returnData='identifier'".
- 1565 • A requestor that wants the provider to return the identifier of the added object  
1566 *plus the XML representation of the object (as defined in the schema of the target)*  
1567 MUST specify "returnData='data'".
- 1568 • A requestor that wants the provider to return the identifier of the added object  
1569 *plus the XML representation of the object (as defined in the schema of the target)*  
1570 *plus any capability-specific data that is associated with the object*  
1571 MAY specify "returnData='everything'" or MAY omit the "returnData" attribute  
1572 (since "returnData='everything'" is the default).

### 1573 [3.6.1.2 addResponse \(normative\)](#)

- 1574 A provider that receives an <addRequest> from a requestor that the provider trusts MUST  
1575 examine the content of the <addRequest>. If the request is valid, the provider MUST create the  
1576 requested object under the specified parent (i.e., target or container object) if it is possible to do so.
- 1577 **PSO Identifier.** The provider MUST create the object with any <psoid> that the <addRequest>  
1578 supplies. If the provider cannot create the object with the specified <psoid> (e.g., because the  
1579 <psoid> is not valid or because an object that already exists has that <psoid>), then the provider  
1580 must fail the request. See the topic named "Error" below within this section.
- 1581 **Data.** The provider MUST create the object with any XML element or attribute contained by the  
1582 <data> element in the <addRequest>.
- 1583 **CapabilityData.** The provider SHOULD associate with the created object the content of each  
1584 <capabilityData> that the <addRequest> contains. The "mustUnderstand" attribute of  
1585 each <capabilityData> indicates whether the provider MUST process the content of the  
1586 <capabilityData> *as the corresponding capability specifies*. See the sections titled  
1587 "[CapabilityData in a Request \(normative\)](#)" and "[CapabilityData Processing \(normative\)](#)".

1588 Also see the section titled "[CapabilityData Errors \(normative\)](#)".

1589 **Execution.** If an `<addRequest>` does not specify a type of execution, a provider MUST choose a  
1590 type of execution for the requested operation.  
1591 See the section titled "[Determining execution mode](#)".

1592 **Response.** The provider must return to the requestor an `<addResponse>`.

1593 **Status.** The `<addResponse>` MUST have a `"status"` attribute that indicates whether the  
1594 provider successfully created the requested object. See the section titled "[Status \(normative\)](#)".

1595 **PSO and ReturnData.** If the provider successfully created the requested object, the  
1596 `<addResponse>` MUST contain an `<pso>` element that contains the (XML representation of the)  
1597 newly created object.

- 1598 • A `<pso>` element MUST contain a `<psoID>` element.  
1599 The `<psoID>` element MUST contain the identifier of the newly created object.  
1600 See the section titled "[PSO Identifier \(normative\)](#)".
- 1601 - If the `<addRequest>` supplies a `<psoID>`, then `<psoID>` of the newly created object  
1602 MUST match the `<psoID>` supplied by the `<addRequest>`.  
1603 (See the topic named "PSO Identifier" above within this section.)
- 1604 - If the `<addRequest>` does not supply `<psoID>`, the provider must generate a `<psoID>`  
1605 that uniquely identifies the newly created object.
- 1606 • A `<pso>` element MAY contain a `<data>` element.
  - 1607 - If the `<addRequest>` specified `"returnData=' identifier' "`  
1608 then the `<pso>` MUST NOT contain a `<data>` element.
  - 1609 - Otherwise, if the `<addRequest>` specified `"returnData=' data' "`  
1610 or (if the `<addRequest>` specified) `"returnData=' everything' "`  
1611 or (if the `<addRequest>`) omitted the `"returnData"` attribute,  
1612 then the `<pso>` MUST contain exactly one `<data>` element that contains the XML  
1613 representation of the object.  
1614 This XML must be valid according to the schema of the target for the schema entity of  
1615 which the newly created object is an instance.
- 1616 • A `<pso>` element MAY contain any number of `<capabilityData>` elements. Each  
1617 `<capabilityData>` element contains a set of *capability-specific data* that is associated with  
1618 the newly created object (for example, a *reference* to another object).  
1619 See the section titled "[CapabilityData in a Response \(normative\)](#)".
  - 1620 - If the `<addRequest>` `"returnData=' identifier' "`  
1621 or (if the `<addRequest>` specified) `"returnData=' data' "`  
1622 then the `<addResponse>` MUST NOT contain a `<capabilityData>` element.
  - 1623 - Otherwise, if the `<addRequest>` specified `"returnData=' everything' "`  
1624 or (if the `<addRequest>`) omitted the `"returnData"` attribute  
1625 then the `<addResponse>` MUST contain a `<capabilityData>` element for each set of  
1626 capability-specific data that is associated with the newly created object.

1627 **Error.** If the provider cannot create the requested object, the `<addResponse>` MUST contain an  
1628 `"error"` attribute that characterizes the failure. See the general section titled "[Error \(normative\)](#)".

1629 In addition, the `<addResponse>` MUST specify an appropriate value of `"error"` if any of the  
1630 following is true:

- 1631 • An <addRequest> specifies "targetID" but the value of "targetID" does not identify a  
1632 target that the provider supports.  
1633 In this case, the <addResponse> SHOULD specify "error='noSuchIdentifier'".
  - 1634 • An <addRequest> specifies "targetID" and (the <addRequest> also) contains  
1635 <containerID> but the value of the "targetID" attribute in the <addRequest> does not  
1636 match the value of the "targetID" attribute in the <containerID>.  
1637 In this case, the <addResponse> SHOULD specify "error='malformedRequest'".
  - 1638 • An <addRequest> contains <containerID> but the content of <containerID> does not  
1639 identify an object that exists. (That is, <containerID> does not match the <psoid> of an  
1640 object that exists.)  
1641 In this case, the <addResponse> SHOULD specify "error='noSuchIdentifier'".
  - 1642 • An <addRequest> contains <containerID> but the <supportedSchemaEntity> (of  
1643 which <containerID> identifies an instance) does not specify "isContainer='true'".  
1644 In this case, the <addResponse> SHOULD specify "error='invalidContainment'".
  - 1645 • An <addRequest> contains <containerID> but the target does not allow the specified  
1646 parent object to contain the object to be created.  
1647 In this case, the <addResponse> SHOULD specify "error='invalidContainment'".
  - 1648 • An <addRequest> supplies <psoid> but the <psoid> element is not valid.  
1649 In this case, the <addResponse> SHOULD specify "error='invalidIdentifier'".
  - 1650 • An <addRequest> supplies <psoid> but an object with that <psoid> already exists.  
1651 In this case, the <addResponse> SHOULD specify "error='alreadyExists'".
  - 1652 • The <data> element is missing an element or attribute that is required (according to the  
1653 schema of the target) for the object to be added.
  - 1654 • A <capabilityData> element specifies "mustUnderstand='true'" and the provider  
1655 cannot associate the content of the <capabilityData> with the object to be created.
- 1656 The provider MAY return an error if:
- 1657 • The <data> element contains data that the provider does not recognize as *valid according to*  
1658 *the target schema* for the type of object to be created.
  - 1659 • The provider does not recognize the content of a <capabilityData> element as specific to  
1660 any capability that the target supports (for the schema entity of which the object to be created is  
1661 an instance).
- 1662 Also see the section titled "[CapabilityData Errors \(normative\)](#)".

### 1663 3.6.1.2.3 *add Examples (non-normative)*

1664 In the following example, a requestor asks a provider to add a new person. The requestor specifies  
1665 the attributes required for the `Person` schema entity (`cn`, `firstName`, `lastName` and `fullName`).  
1666 The requestor also supplies an optional `email` address for the person. This example assumes that  
1667 a container named "ou=Development, org=Example" already exists.

```
<addRequest requestID="127" targetID="target2">
  <containerID ID="ou=Development, org=Example"/>
  <data>
    <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob Briggs">
```

```
<email>joebob@example.com</email>
</Person>
</data>
</addRequest>
```

- 1668 The provider returns an `<addResponse>` element. The "status" attribute of the  
1669 `<addResponse>` element indicates that the add request was successfully processed. The  
1670 `<addResponse>` contains a `<pso>`. The `<pso>` contains a `<psoID>` that identifies the newly  
1671 created object. The `<pso>` also contains a `<data>` element that contains the schema-defined XML  
1672 representation of the newly created object.

```
<addResponse requestID="127" status="success">
  <pso>
    <psoID ID="2244" targetID="target2"/>
    <data>
      <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob
Briggs">
        <email>joebob@example.com</email>
      </Person>
    </data>
  </pso>
</addResponse>
```

- 1673 Next, the requestor asks a provider to add a new account. The requestor specifies a name for the  
1674 account. The requestor also specifies references to a `Group` that resides on `target1` and to a  
1675 `Person` (from the first example in this section) that resides on `target2`.

```
<addRequest requestID="128" targetID="target1">
  <data>
    <Account accountName="joebob"/>
  </data>
  <capabilityData mustUnderstand="true"
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
    <reference typeOfReference="memberOf">
      <toPsoID ID="group1" targetID="target1"/>
    </reference>
    <reference typeOfReference="owner">
      <toPsoID ID="2244" targetID="target2"/>
    </reference>
  </capabilityData>
</addRequest>
```

- 1676 The provider returns an `<addResponse>` element. The "status" attribute of the  
1677 `<addResponse>` element indicates that the add operation was successfully processed. The  
1678 `<addResponse>` contains a `<pso>` that contains a `<psoID>` that identifies the newly created  
1679 object.

```
<addResponse requestID="128" status="success">
  <pso>
    <psoID ID="1431" targetID="target1"/>
    <data>
      <Account accountName="joebob"/>
    </data>
    <capabilityData mustUnderstand="true"
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="memberOf">
        <toPsoID ID="group1" targetID="target1"/>
      </reference>
    </capabilityData>
  </pso>
</addResponse>
```

```

    <reference typeOfReference="owner">
      <toPsoID ID="2244" targetID="target2"/>
    </reference>
  </capabilityData>
</pso>
</addResponse>

```

### 1680 3.6.1.3 lookup

1681 The lookup operation enables a requestor to *obtain the XML that represents an object* on a target.

1682 The lookup operation also obtains any *capability-specific data* that is associated with the object.

1683 The subset of the Core XSD that is most relevant to the lookup operation follows.

```

<complexType name="CapabilityDataType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <annotation>
        <documentation>Contains elements specific to a
capability.</documentation>
      </annotation>
      <attribute name="mustUnderstand" type="boolean"
use="optional"/>
      <attribute name="capabilityURI" type="anyURI"/>
    </extension>
  </complexContent>
</complexType>

<simpleType name="ReturnDataType">
  <restriction base="string">
    <enumeration value="identifier"/>
    <enumeration value="data"/>
    <enumeration value="everything"/>
  </restriction>
</simpleType>

<complexType name="PSOType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType"/>
        <element name="data" type="spml:ExtensibleType"
minOccurs="0"/>
        <element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="LookupRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

```

        <attribute name="returnData" type="spml:ReturnDataType"
use="optional" default="everything"/>
    </extension>
</complexContent>
</complexType>

<complexType name="LookupResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <sequence>
                <element name="pso" type="spml:PSOType" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<element name="lookupRequest" type="spml:LookupRequestType"/>
<element name="lookupResponse" type="spml:LookupResponseType"/>

```

### 1684 **3.6.1.3.1** *lookupRequest (normative)*

1685 A requestor **MUST** send a `<lookupRequest>` to a provider in order to (ask the provider to) return  
1686 (the XML that represents) an existing object.

1687 **Execution.** A `<lookupRequest>` **MAY** specify "executionMode".  
1688 See the section titled "[Determining execution mode](#)".

1689 In general, a requestor **SHOULD NOT** specify "executionMode='asynchronous' ". The  
1690 reason for this is that the result of a lookup should reflect the current state of a target object. If a  
1691 lookup operation is executed asynchronously then other operations are more likely to intervene.

1692 **PsoID.** A `<lookupRequest>` **MUST** contain exactly one `<psoID>` that identifies the object to  
1693 lookup (i.e., the object for which the provider should return the XML representation). The `<psoID>`  
1694 **MUST** identify an object that exists on a target.

1695 **ReturnData.** A `<lookupRequest>` **MAY** have a "returnData" attribute that tells the provider  
1696 which subset of (the XML representation of) a `<pso>` to include in the provider's response.

- 1697 • A requestor that wants the provider to return *nothing* of a requested object  
1698 **MUST** specify "returnData='nothing' ".
- 1699 • A requestor that wants the provider to return *only the identifier* of a requested object  
1700 **MUST** specify "returnData='identifier' ".
- 1701 • A requestor that wants the provider to return the identifier of a requested object  
1702 *plus the XML representation of the object (as defined in the schema of the target)*  
1703 **MUST** specify "returnData='data' ".
- 1704 • A requestor that wants the provider to return the identifier of a requested object  
1705 *plus the XML representation of the object (as defined in the schema of the target)*  
1706 *plus any capability-specific data that is associated with the object*  
1707 **MAY** specify "returnData='everything' " or **MAY** omit the "returnData" attribute  
1708 (since "returnData='everything' " is the default).

### 1709 3.6.1.3.2 *lookupResponse* (normative)

1710 A provider that receives a <lookupRequest> from a requestor that the provider trusts MUST  
1711 examine the content of the <lookupRequest>. If the request is valid, the provider MUST return  
1712 (the XML that represents) the requested object if it is possible to do so.

1713 **Execution.** If an <lookupRequest> does not specify "executionMode", the provider MUST  
1714 choose a type of execution for the requested operation.  
1715 See the section titled "[Determining execution mode](#)".

1716 A provider SHOULD execute a lookup operation synchronously if it is possible to do so. The reason  
1717 for this is that the result of a lookup should reflect the current state of a target object. If a lookup  
1718 operation is executed asynchronously then other operations are more likely to intervene.

1719 **Response.** The provider must return to the requestor a <lookupResponse>.

1720 **Status.** The <lookupResponse> must have a "status" that indicates whether the provider  
1721 successfully returned each requested object. See the section titled "[Status](#) (normative)".

1722 **PSO and ReturnData.** If the provider successfully returned the requested object, the  
1723 <lookupResponse> MUST contain an <ps> element for the requested object. Each <ps>  
1724 contains the subset of (the XML representation of) a requested object that the "returnData"  
1725 attribute of the <lookupRequest> specified. By default, each <ps> contains the entire (XML  
1726 representation of an) object.

1727 • A <ps> element MUST contain a <psID> element.  
1728 The <psID> element MUST contain the identifier of the requested object.  
1729 See the section titled "[PSO Identifier](#) (normative)".

1730 • A <ps> element MAY contain a <data> element.  
1731 - If the <lookupRequest> specified "returnData='identifier'",  
1732 then the <ps> MUST NOT contain a <data> element.  
1733 - Otherwise, if the <lookupRequest> specified "returnData='data'"  
1734 or (if the <lookupRequest> specified) "returnData='everything'"  
1735 or (if the <lookupRequest>) omitted the "returnData" attribute  
1736 then the <data> element MUST contain the XML representation of the object.  
1737 This XML must be valid according to the schema of the target for the schema entity of  
1738 which the newly created object is an instance.

1739 • A <ps> element MAY contain any number of <capabilityData> elements.  
1740 Each <capabilityData> element MUST contain all the data (that are associated with the  
1741 object and) that are specific to the capability that the <capabilityData> specifies as  
1742 "capabilityURI". For example, a <capabilityData> that refers to the Reference  
1743 Capability (i.e., a <capabilityData> that specifies  
1744 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'")  
1745 must contain at least one *reference* to another object.  
1746 See the section titled "[CapabilityData in a Response](#) (normative)".

1747 - If the <lookupRequest> specified "returnData='identifier'"  
1748 or (if the <lookupRequest> specified) "returnData='data'"  
1749 then the <ps> MUST NOT contain a <capabilityData> element.

1750 - Otherwise, if the <lookupRequest> specified "returnData='everything'"  
1751 or (if the <lookupRequest>) omitted the "returnData" attribute,  
1752 then the <ps> MUST contain a <capabilityData> element  
1753 for each set of capability-specific data that is associated with the requested object

1754 (and that is specific to a capability that the target supports for the schema entity  
1755 of which the requested object is an instance).

1756 **Error.** If the provider cannot return the requested object, the `<lookupResponse>` must have an  
1757 "error" attribute that characterizes the failure. See the general section titled "Error (normative)".

1758 In addition, the `<lookupResponse>` MUST specify an appropriate value of "error" if any of the  
1759 following is true:

- 1760 • A `<lookupRequest>` contains no `<psoID>`.
- 1761 • A `<lookupRequest>` contains a `<psoID>` that does not identify an object that exists on a  
1762 target.

1763 The provider MAY return an error if:

- 1764 • A `<psoID>` contains data that the provider does not recognize.

### 1765 **3.6.1.3.3** *lookup Examples (non-normative)*

1766 In the following example, a requestor asks a provider to return the `Person` object from the [add](#)  
1767 [examples](#) above. The requestor specifies the `<psoID>` for the `Person` object.

```
<lookupRequest requestID="125">  
  <psoID ID="2244" targetID="target2"/>  
</lookupRequest>
```

1768 The provider returns a `<lookupResponse>` element. The "status" attribute of the  
1769 `<lookupResponse>` element indicates that the lookup request was successfully processed. The  
1770 `<lookupResponse>` contains a `<pso>` element that contains the requested object.

1771 The `<pso>` element contains a `<psoID>` element that contains the PSO Identifier. The `<pso>` also  
1772 contains a `<data>` element that contains the XML representation of the object (according to the  
1773 schema of the target).

```
<lookupResponse requestID="125" status="success">  
  <pso>  
    <psoID ID="2244" targetID="target2"/>  
    <data>  
      <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob  
Briggs">  
        <email>joebob@example.com</email>  
      </Person>  
    </data>  
  </pso>  
</lookupResponse>
```

1774 Next, the requestor asks a provider to return the `Account` object from the [add examples](#) above.  
1775 The requestor specifies a `<psoID>` for the `Account` object.

```
<lookupRequest requestID="126">  
  <psoID ID="1431" targetID="target1"/>  
</lookupRequest>
```

1776 The provider returns a `<lookupResponse>` element. The "status" attribute of the  
1777 `<lookupResponse>` element indicates that the lookup request was successfully processed. The  
1778 `<lookupResponse>` contains a `<pso>` element that contains the requested object.

1779 The <ps> element contains a <psID> element that uniquely identifies the object. The <ps>  
1780 also contains a <data> element that contains the XML representation of the object (according to  
1781 the schema of the target).

1782 In this example, the <ps> element also contains a <capabilityData> element. The  
1783 <capabilityData> element in turn contains two <reference> elements. The lookup operation  
1784 automatically includes capability-specific data (such as these two reference elements) if the  
1785 schema for the target declares that it supports the reference capability (for the schema entity of  
1786 which the requested object is an instance).

```
<lookupResponse requestID="126" status="success">
  <ps>
    <psID ID="1431" targetID="target1"/>
    <data>
      <Account accountName="joebob"/>
    </data>
    <capabilityData mustUnderstand="true"
  capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="memberOf">
        <toPsID ID="group1" targetID="target1"/>
      </reference>
      <reference typeOfReference="owner">
        <toPsID ID="2244" targetID="target2"/>
      </reference>
    </capabilityData>
  </ps>
</lookupResponse>
```

1787 To illustrate the effect of the "returnData" attribute, let's reissue the previous request and  
1788 specify a value of "returnData" other than the default (which is  
1789 "returnData='everything'"). First, assume that the requestor specifies  
1790 "returnData='identifier'".

```
<lookupRequest requestID="129" returnData="identifier">
  <psID ID="1431" targetID="target1"/>
</lookupRequest>
```

1791 The response specifies "status='success'" which indicates that the lookup operation  
1792 succeeded and that the requested object exists. Since the request specifies  
1793 "return='identifier'", the <ps> in the response contains the <psID> but no <data>.

```
<lookupResponse requestID="129" status="success">
  <ps>
    <psID ID="1431" targetID="target1"/>
  </ps>
</lookupResponse>
```

1794 Next assume that the requestor specifies "returnData='data'".

```
<lookupRequest requestID="130" returnData="data">
  <psID ID="1431" targetID="target1"/>
</lookupRequest>
```

1795 Since the request specifies "return='data'", the <ps> in the response contains the <psID>  
1796 and <data> but no <capabilityData> element. Specifying "return='data'" returns the  
1797 XML representation of the object as defined in the schema for the target but *suppresses capability-*  
1798 *specific data*.

1799 Specifying "return='data'" is advantageous if the requestor is not interested in capability-  
1800 specific data. Omitting capability-specific data may reduce the amount of work that the provider

1801 must do in order to build the <lookupResponse>. Reducing the size of the response should also  
1802 reduce the network traffic that is required in order to transmit the response. Omitting capability-  
1803 specific data may also reduce the amount of XML parsing work that the requestor must perform in  
1804 order to process the response.

```
<lookupResponse requestID="130" status="success">
  <ps0>
    <ps0ID ID="1431" targetID="target1"/>
    <data>
      <Account accountName="joebob"/>
    </data>
  </ps0>
</lookupResponse>
```

#### 1805 **3.6.1.4 modify**

1806 The modify operation enables a requestor to *change an object* on a target. The modify operation  
1807 can change the *schema-defined component* of an object, any *capability-specific data* that is  
1808 associated with the object, or *both*.

1809 **Modify can change PSO Identifier.** One important subtlety is that a modify operation may change  
1810 the identifier of the modified object. For example, assume that a provider exposes the  
1811 Distinguished Name (DN) as the identifier of each object on a target that represents a directory  
1812 service. In this case, modifying the object's Common Name (CN) or moving the object beneath a  
1813 different Organizational Unit (OU) would change the object's DN and therefore its PSO-ID.

1814 A provider should expose an immutable identifier as the PSO-ID of each object. In the case of a  
1815 target that represents a directory service, an immutable identifier could be a Globally Unique  
1816 Identifier (GUID) that is managed by the directory service or it could be any form of unique identifier  
1817 that is managed by the provider.

1818 For normative specifics, please see the section titled "[PSO Identifier \(normative\)](#)".

1819 **Modifying capability-specific data.** Any capability may imply capability-specific data (where the  
1820 target supports that capability for the schema entity of which the object is an instance). However,  
1821 many capabilities do not. Of the standard capabilities that SPMLv2 defines, only the [Reference](#)  
1822 [Capability](#) implies capability-specific data.

1823 The default processing for capability-specific data is to treat the content of each  
1824 <capabilityData> as opaque. See the section titled "[CapabilityData](#)".

1825 The subset of the Core XSD that is most relevant to the modify operation follows.

```
<complexType name="CapabilityDataType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <annotation>
        <documentation>Contains elements specific to a
capability.</documentation>
      </annotation>
      <attribute name="mustUnderstand" type="boolean"
use="optional"/>
      <attribute name="capabilityURI" type="anyURI"/>
    </extension>
  </complexContent>
</complexType>
```

```

<simpleType name="ReturnDataType">
  <restriction base="string">
    <enumeration value="identifier"/>
    <enumeration value="data"/>
    <enumeration value="everything"/>
  </restriction>
</simpleType>

  <complexType name="PSOType">
    <complexContent>
      <extension base="spml:ExtensibleType">
        <sequence>
          <element name="psoID" type="spml:PSOIdentifierType"/>
          <element name="data" type="spml:ExtensibleType"
minOccurs="0"/>
          <element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <simpleType name="ModificationModeType">
    <restriction base="string">
      <enumeration value="add"/>
      <enumeration value="replace"/>
      <enumeration value="delete"/>
    </restriction>
  </simpleType>

  <complexType name="NamespacePrefixMappingType">
    <complexContent>
      <extension base="spml:ExtensibleType">
        <attribute name="prefix" type="string" use="required"/>
        <attribute name="namespace" type="string" use="required"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="QueryClauseType">
    <complexContent>
      <extension base="spml:ExtensibleType">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="SelectionType">
    <complexContent>
      <extension base="spml:QueryClauseType">
        <sequence>
          <element name="namespacePrefixMap"
type="spml:NamespacePrefixMappingType" minOccurs="0"
maxOccurs="unbounded"/>
        </sequence>
        <attribute name="path" type="string" use="required"/>
        <attribute name="namespaceURI" type="string" use="required"/>
      </extension>
    </complexContent>
  </complexType>

```

```

        </extension>
    </complexContent>
</complexType>

<complexType name="ModificationType">
    <complexContent>
        <extension base="spml:ExtensibleType">
            <sequence>
                <element name="component" type="spml:SelectionType"
minOccurs="0"/>
                <element name="data" type="spml:ExtensibleType"
minOccurs="0"/>
                <element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="modificationMode"
type="spml:ModificationModeType" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="ModifyRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="psoID" type="spml:PSOIdentifierType"/>
                <element name="modification" type="spml:ModificationType"
maxOccurs="unbounded"/>
            </sequence>
            <attribute name="returnData" type="spml:ReturnDataType"
use="optional" default="everything"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="ModifyResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <sequence>
                <element name="pso" type="spml:PSOType" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<element name="modifyRequest" type="spml:ModifyRequestType"/>
<element name="modifyResponse" type="spml:ModifyResponseType"/>

```

#### 1826 **3.6.1.4.1** *modifyRequest (normative)*

1827 A requestor **MUST** send a <modifyRequest> to a provider in order to (ask the provider to) modify  
1828 an existing object.

1829 **Execution.** A <modifyRequest> **MAY** specify "executionMode".

1830 See the section titled "[Determining execution mode](#)".

1831 **PsoID.** A <modifyRequest> MUST contain exactly one <psoID>. A <psoID> MUST identify an  
1832 object that exists on a target that is exposed by the provider.

1833 **ReturnData.** A <modifyRequest> MAY have a "returnData" attribute that tells the provider  
1834 which subset of (the XML representation of) each modified <pso> to include in the provider's  
1835 response.

- 1836 • A requestor that wants the provider to return *nothing* of the modified object  
1837 MUST specify "returnData='nothing'".
- 1838 • A requestor that wants the provider to return *only the identifier* of the modified object  
1839 MUST specify "returnData='identifier'".
- 1840 • A requestor that wants the provider to return the identifier of the modified object  
1841 *plus the XML representation of the object (as defined in the schema of the target)*  
1842 MUST specify "returnData='data'".
- 1843 • A requestor that wants the provider to return the identifier of the modified object  
1844 *plus the XML representation of the object (as defined in the schema of the target)*  
1845 *plus any capability-specific data that is associated with the object*  
1846 MAY specify "returnData='everything'" or MAY omit the "returnData" attribute  
1847 (since "returnData='everything'" is the default).

1848 **Modification.** A <modifyRequest> MUST contain at least one <modification>. A  
1849 <modification> describes a set of changes to be applied (to the object that the <psoID>  
1850 identifies). A <modification> MUST have a "modificationMode" that specifies the type of  
1851 change as one of 'add', 'replace' or 'delete'.

1852 A requestor MAY specify a change to a schema-defined element or attribute of the object to be  
1853 modified. A requestor MAY specify any number of changes to capability-specific data associated  
1854 with the object to be modified.

1855 A requestor MUST use a <component> element to specify a schema-defined element or attribute  
1856 of the object to be modified. A requestor MUST use a <capabilityData> element to describe  
1857 each change to a capability-specific data element that is associated with the object to be modified.

1858 A <modification> element MUST contain a <component> element or (the <modification>  
1859 MUST contain) at least one <capabilityData> element. A <modification> element MAY  
1860 contain a <component> element *as well as* one or more <capabilityData> elements.

1861 **Modification component.** The <component> sub-element of a <modification> specifies a  
1862 schema-defined element or attribute of the object that is to be modified. This is an instance of  
1863 {SelectionType}, which occurs in several contexts within SPMLv2.  
1864 See the section titled "[SelectionType in a Request \(normative\)](#)".

1865 **Modification data.** A requestor MUST specify as the content of the <data> sub-element of a  
1866 <modification> any content or *value* that is to be added to, replaced within, or deleted from the  
1867 element or attribute that the <component> (sub-element of the <modification>) specifies.

1868 **Modification capabilityData.** A requestor MAY specify any number of <capabilityData>  
1869 elements within a <modification> element. Each <capabilityData> element specifies  
1870 *capability-specific data* (for example, *references* to other objects) for the object to be modified.  
1871 Because the {CapabilityDataType} is an {ExtensibleType}, a <capabilityData>  
1872 element may validly contain any XML element or attribute. The <capabilityData> element  
1873 SHOULD contain elements that the provider will recognize as specific to a capability that the target  
1874 supports (for the schema entity of which the object to be modified is an instance).  
1875 See the section titled "[CapabilityData in a Request \(normative\)](#)".

1876 **3.6.1.4.2 modifyResponse (normative)**

1877 A provider that receives a <modifyRequest> from a requestor that the provider trusts MUST  
1878 examine the content of the <modifyRequest>. If the request is valid, the provider MUST apply  
1879 each requested <modification> (to the object that is identified by the <psoid> of the  
1880 <modifyRequest>) if it is possible to do so.

1881 For normative specifics related to processing any <capabilityData> within a  
1882 <modification>, please see the section titled "[CapabilityData Processing \(normative\)](#)".

1883 **Execution.** If a <modifyRequest> does not specify "executionMode", the provider MUST  
1884 choose a type of execution for the requested operation.  
1885 See the section titled "[Determining execution mode](#)".

1886 **Response.** The provider must return to the requestor a <modifyResponse>.

1887 **Status.** The <modifyResponse> must have a "status" attribute that indicates whether the  
1888 provider successfully applied the requested modifications to each identified object.  
1889 See the section titled "[Status \(normative\)](#)".

1890 **PSO and ReturnData.** If the provider successfully modified the requested object, the  
1891 <modifyResponse> MUST contain an <psoid> element. The <psoid> contains the subset of (the  
1892 XML representation of) a requested object that the "returnData" attribute of the  
1893 <lookupRequest> specified. By default, the <psoid> contains the entire (XML representation of  
1894 the) modified object.

- 1895 • A <psoid> element MUST contain a <psoid> element.  
1896 The <psoid> element MUST contain the identifier of the requested object.  
1897 See the section titled "[PSO Identifier \(normative\)](#)".
- 1898 • A <psoid> element MAY contain a <data> element.
  - 1899 - If the <modifyRequest> specified "returnData=' identifier' ",  
1900 then the <psoid> MUST NOT contain a <data> element.
  - 1901 - Otherwise, if the <modifyRequest> specified "returnData=' data' "  
1902 or (if the <modifyRequest> specified) "returnData=' everything' "  
1903 or (if the <modifyRequest>) omitted the "returnData" attribute  
1904 then the <data> element MUST contain the XML representation of the object.  
1905 This XML must be valid according to the schema of the target for the schema entity of  
1906 which the newly created object is an instance.
- 1907 • A <psoid> element MAY contain any number of <capabilityData> elements. Each  
1908 <capabilityData> element contains a set of *capability-specific data* that is associated with  
1909 the newly created object (for example, a *reference* to another object).  
1910 See the section titled "[CapabilityData in a Response \(normative\)](#)".
  - 1911 - If the <modifyRequest> specified "returnData=' identifier' "  
1912 or (if the <modifyRequest> specified) "returnData=' data' "  
1913 then the <modifyResponse> MUST NOT contain a <capabilityData> element.
  - 1914 - Otherwise, if the <modifyRequest> specified "returnData=' everything' "  
1915 or (if the <modifyRequest>) omitted the "returnData" attribute,  
1916 then the <modifyResponse> MUST contain a <capabilityData> element for each set  
1917 of capability-specific data that is associated with the requested object  
1918 (and that is specific to a capability that the target supports for the schema entity of which  
1919 the requested object is an instance).

1920 **Error.** If the provider cannot modify the requested object, the `<modifyResponse>` must have an  
1921 "error" attribute that characterizes the failure. See the general section titled "[Error \(normative\)](#)".

1922 In addition, a `<modifyResponse>` MUST specify an appropriate value of "error" if any of the  
1923 following is true:

- 1924 • The `<modifyRequest>` contains a `<modification>` for which there is no corresponding  
1925 `<psoID>`.
- 1926 • A `<modification>` contains neither a `<component>` nor a `<capabilityData>`.
- 1927 • A `<component>` is empty (that is, a `<component>` element has no content).
- 1928 • A `<component>` specifies an element or attribute that is not valid (according to the schema of  
1929 the target) for the type of object to be modified.

1930 The provider MAY return an error if:

- 1931 • A `<component>` contains data that the provider does not recognize as specifying an XML  
1932 element or attribute that is *valid according to the target schema* for the type of object to be  
1933 modified.
- 1934 • A `<capabilityData>` element contains data that the provider does not recognize as specific  
1935 to the capability that its "capabilityURI" attribute identifies.

1936 In addition, see the section titled "[SelectionType Errors \(normative\)](#)" as well as the section titled  
1937 "[CapabilityData Errors \(normative\)](#)".

### 1938 [3.6.1.4.3 modify Examples \(non-normative\)](#)

1939 In the following example, a requestor asks a provider to modify the email address for an existing  
1940 `Person` object.

```
<modifyRequest requestID="123">
  <psoID ID="2244" targetID="target2"/>
  <modification modificationMode="replace">
    <component path="/Person/email" namespaceURI="http://www.w3.org/TR/xpath20" />
    <data>
      <email>joebob@example.com</email>
    </data>
  </modification>
</modifyRequest>
```

1941 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
1942 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
1943 `<pso>` element of the `<modifyResponse>` contains the XML representation of the modified  
1944 object.

```
<modifyResponse requestID="123" status="success">
  <pso>
    <psoID ID="2244" targetID="target2"/>
    <data>
      <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob
Briggs">
        <email>joebob@example.com</email>
      </Person>
    </data>
  </pso>
```

---

```
</modifyResponse>
```

1945 In the following example, a requestor asks a provider to modify the same `Person` object, adding a  
1946 reference to an `Account` that the `Person` owns. (Since the request is to add capability-specific  
1947 data, the `<modification>` element contains no `<component>` sub-element.)

```
<modifyRequest requestID="124">  
  <psolD ID="2244" targetID="target2"/>  
  <modification modificationMode="add">  
    <capabilityData mustUnderstand="true"  
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">  
      <reference typeOfReference="owns" >  
        <toPsolD ID="1431" targetID="target1"/>  
      </reference>  
    </capabilityData>  
  </modification>  
</modifyRequest>
```

1948 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
1949 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
1950 `<pso>` element of the `<modifyResponse>` shows that the provider has added (the  
1951 `<capabilityData>` that is specific to) the "owns" reference.

```
<modifyResponse requestID="124" status="success">  
  <pso>  
    <psolD ID="2244" targetID="target2"/>  
    <data>  
      <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob  
Briggs">  
        <email>joebob@example.com</email>  
      </Person>  
    </data>  
    <capabilityData mustUnderstand="true"  
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">  
      <reference typeOfReference="owns">  
        <toPsolD ID="1431" targetID="target1"/>  
      </reference>  
    </capabilityData>  
  </pso>  
</modifyResponse>
```

1952

1953 **Modifying capabilityData.** Of the standard capabilities defined by SPMLv2, only the [Reference](#)  
1954 [Capability](#) associates capability-specific data with an object. We must therefore imagine a custom  
1955 capability "foo" in order to illustrate the *default processing* of capability data. (We illustrate the  
1956 handling of references further below.)

1957 In this example, the requestor wishes to replace any existing data foo-specific data that is  
1958 associated with a specific `Account` with a new `<foo>` element. The fact that each  
1959 `<capabilityData>` omits the "mustUnderstand" flag indicates that the requestor will accept  
1960 the default processing.

```

<modifyRequest requestID="122">
  <psolD ID="1431" targetID="target1"/>
  <modification modificationMode="replace">
    <capabilityData capabilityURI="urn:oasis:names:tc:SPML:2.0:foo">
      <foo bar="owner"/>
    </capabilityData>
  </modification>
</modifyRequest>

```

- 1961 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
 1962 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
 1963 `<pso>` element of the `<modifyResponse>` shows that any capability data that is specific to the  
 1964 Foo capability has been replaced.

```

<modifyResponse requestID="122" status="success">
  <pso>
    <psolD ID="1431" targetID="target1"/>
    <data>
      <Account accountName="joebob"/>
    </data>
    <capabilityData capabilityURI="urn:oasis:names:tc:SPML:2.0:foo">
      <foo bar="owner"/>
    </capabilityData>
    <capabilityData mustUnderstand="true"
  capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="memberOf">
        <toPsolD ID="group1" targetID="target1"/>
      </reference>
      <reference typeOfReference="owner">
        <toPsolD ID="2245" targetID="target2"/>
      </reference>
    </capabilityData>
  </pso>
</modifyResponse>

```

- 1965 The requestor next adds another `<foo>` element to the set of foo-specific data that is associated  
 1966 with the Account.

```

<modifyRequest requestID="122">
  <psolD ID="1431" targetID="target1"/>
  <modification modificationMode="add">
    <capabilityData capabilityURI="urn:oasis:names:tc:SPML:2.0:foo">
      <foo bar="customer"/>
    </capabilityData>
  </modification>
</modifyRequest>

```

- 1967 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
 1968 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
 1969 `<pso>` element of the `<modifyResponse>` shows that the content of the foo-specific  
 1970 `<capabilityData>` in the `<modification>` has been appended to the previous content of the  
 1971 foo-specific `<capabilityData>` in the `<pso>`.

```

<modifyResponse requestID="122" status="success">
  <pso>
    <psolD ID="1431" targetID="target1"/>
    <data>
      <Account accountName="joebob"/>
    </data>
    <capabilityData capabilityURI="urn:oasis:names:tc:SPML:2.0:foo">
      <foo bar="owner"/>
      <foo bar="customer"/>
    </capabilityData>
    <capabilityData mustUnderstand="true"
  capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="memberOf">
        <toPsolD ID="group1" targetID="target1"/>
      </reference>
      <reference typeOfReference="owner">
        <toPsolD ID="2245" targetID="target2"/>
      </reference>
    </capabilityData>
  </pso>
</modifyResponse>

```

- 1972 Finally, our requestor deletes any foo-specific capability data from the Account. The  
 1973 <capabilityData> element does not need any content. The content of <capabilityData> is  
 1974 irrelevant in the default processing of "modificationMode='delete'".

```

<modifyRequest requestID="122">
  <psolD ID="1431" targetID="target1"/>
  <modification modificationMode="delete">
    <capabilityData capabilityURI="urn:oasis:names:tc:SPML:2.0:foo"/>
  </modification>
</modifyRequest>

```

- 1975 The provider returns a <modifyResponse> element. The "status" attribute of the  
 1976 <modifyResponse> element indicates that the modify request was successfully processed. The  
 1977 <pso> element of the <modifyResponse> shows that the foo-specific <capabilityData> has  
 1978 been removed.

```

<modifyResponse requestID="122" status="success">
  <pso>
    <psolD ID="1431" targetID="target1"/>
    <data>
      <Account accountName="joebob"/>
    </data>
    <capabilityData mustUnderstand="true"
  capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="memberOf">
        <toPsolD ID="group1" targetID="target1"/>
      </reference>
      <reference typeOfReference="owner">
        <toPsolD ID="2245" targetID="target2"/>
      </reference>
    </capabilityData>
  </pso>
</modifyResponse>

```

1979

1980 **Modifying a reference.** The previous topic illustrates the default processing of capability data. The  
1981 Reference Capability specifies enhanced behavior for the modify operation.  
1982 See the section titled "[Reference CapabilityData Processing \(normative\)](#)".

1983 In this example, the requestor wishes to change the owner of an `Account` from "2244" (which is  
1984 the `<psoID>` of "Person:joebob") to "2245" (which is the `<psoID>` of "Person:billybob").

1985 Since SPMLv2 does not specify any mechanism to define the cardinality of a type of reference, a  
1986 requestor should not assume that a provider enforces any specific cardinality for any type of  
1987 reference. For a general discussion of the issues surrounding references, see the section titled  
1988 "[Reference Capability](#)".

1989 Assume that each account should have at most one owner. If the requestor could trust the provider  
1990 to enforce this, and if the requestor could trust that no other requestor has changed the value of  
1991 "owner", the requestor could simply ask the provider to replace the owner value 2244 with 2245.  
1992 However, since our requestor is both cautious and general, the requestor instead nests two  
1993 `<modification>` elements within a single `<modifyRequest>`:  
1994 - one `<modification>` to *delete any current values* of "owner" and  
1995 - one `<modification>` to *add the desired value* of "owner".

1996 The `<modification>` that specifies "modificationMode='delete'" contains a  
1997 `<capabilityData>` that specifies "mustUnderstand='true'". This means that the provider  
1998 must process the content of that `<capabilityData>` as the Reference Capability specifies. (If  
1999 the provider cannot do that, the provider must fail the request.)

2000 The `<capabilityData>` contains a `<reference>` that specifies only  
2001 "typeOfReference='owner'". The `<reference>` contains no `<toPsoID>` and (the  
2002 `<reference>` contains) no `<referenceData>` element. The Reference Capability specifies that  
2003 this *incomplete reference acts as a wildcard*. In this context, this `<reference>` that specifies only  
2004 "typeOfReference" matches every `<reference>` that is associated with the object and that  
2005 specifies "typeOfReference='owner'".

```
<modifyRequest requestID="121">
  <psoID ID="1431" targetID="target1"/>
  <modification modificationMode="delete">
    <capabilityData mustUnderstand="true"
  capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="owner"/>
    </capabilityData>
  </modification>
  <modification modificationMode="add">
    <capabilityData mustUnderstand="true"
  capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="owner" >
        <toPsoID ID="2245" targetID="target2"/>
      </reference>
    </capabilityData>
  </modification>
</modifyRequest>
```

2006 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
2007 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
2008 `<pso>` element of the `<modifyResponse>` shows that the `<reference>` that specifies  
2009 "typeOfReference='owner'" has been changed.

```
<modifyResponse requestID="121" status="success">
  <pso>
```

```

    <psoID ID="1431" targetID="target1"/>
    <data>
      <Account accountName="joebob"/>
    </data>
    <capabilityData mustUnderstand="true"
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="memberOf">
        <toPsoID ID="group1" targetID="target1"/>
      </reference>
      <reference typeOfReference="owner">
        <toPsoID ID="2245" targetID="target2"/>
      </reference>
    </capabilityData>
  </pso>
</modifyResponse>

```

### 2010 **3.6.1.5 delete**

2011 The delete operation enables a requestor to *remove an object* from a target. The delete operation  
 2012 automatically removes any *capability-specific data* that is associated with the object.

2013 The subset of the Core XSD that is most relevant to the delete operation follows.

```

    <complexType name="DeleteRequestType">
      <complexContent>
        <extension base="spml:RequestType">
          <sequence>
            <element name="psoID" type="spml:PSOIdentifierType"/>
          </sequence>
          <attribute name="recursive" type="xsd:boolean" use="optional"
default="false"/>
        </extension>
      </complexContent>
    </complexType>

    <element name="deleteRequest" type="spml:DeleteRequestType"/>
    <element name="deleteResponse" type="spml:ResponseType"/>

```

#### 2014 **3.6.1.5.1 deleteRequest (normative)**

2015 A requestor **MUST** send a `<deleteRequest>` to a provider in order to (ask the provider to)  
 2016 remove an existing object.

2017 **Execution.** A `<deleteRequest>` **MAY** specify "executionMode".  
 2018 See the section titled "[Determining execution mode](#)".

2019 **PsoID.** A `<deleteRequest>` **MUST** contain a `<psoID>` element that identifies the object to  
 2020 delete.

2021 **Recursive.** A `<deleteRequest>` **MAY** have a "recursive" attribute that specifies whether the  
 2022 provider should delete (along with the specified object) any object that the specified object (either  
 2023 directly or indirectly) contains.

- 2024 • A requestor that wants the provider to *delete any object that the specified object contains*  
2025 (along with the specified object) MUST specify "recursive='true'".
- 2026 • A requestor that wants the provider to delete the specified object *only if the specified object*  
2027 *contains no other object* MUST NOT specify "recursive='true' ". Such a requestor MAY  
2028 specify "recursive='false' " or (such a requestor MAY) omit the "recursive" attribute  
2029 (since "recursive='false' " is the default).

### 2030 3.6.1.5.2 deleteResponse (normative)

2031 A provider that receives a <deleteRequest> from a requestor that the provider trusts MUST  
2032 examine the content of the request. If the request is valid, the provider MUST delete the object  
2033 (that is specified by the <psoid> sub-element of the <deleteRequest>) if it is possible to do so.

2034 **Execution.** If an <deleteRequest> does not specify "executionMode", the provider MUST  
2035 choose a type of execution for the requested operation.  
2036 See the section titled "Determining execution mode".

2037 **Recursive.** A provider MUST NOT delete an object that contains another object unless the  
2038 <deleteRequest> specifies "recursive='true' ". If the <deleteRequest> specifies  
2039 "recursive='true' " then the provider MUST delete the specified object along with any object  
2040 that the specified object (directly or indirectly) contains.

2041 **Response.** The provider must return to the requestor a <deleteResponse>.

2042 **Status.** A <deleteResponse> must contain a "status" attribute that indicates whether the  
2043 provider successfully deleted the specified object. See the section titled "Status (normative)".

2044 **Error.** If the provider cannot delete the specified object, the <deleteResponse> must contain an  
2045 "error" attribute that characterizes the failure. See the general section titled "Error (normative)".

2046 In addition, the <deleteResponse> MUST specify an appropriate value of "error" if any of the  
2047 following is true:

- 2048 • The <psoid> sub-element of the <deleteRequest> is empty (that is, the identifier  
2049 element has no content). In this case, the <deleteResponse> SHOULD specify  
2050 "error='noSuchIdentifier'".
- 2051 • The <psoid> sub-element of the <deleteRequest> contains invalid data. In this case the  
2052 provider SHOULD return "error='unsupportedIdentifierType'".
- 2053 • The <psoid> sub-element of the <deleteRequest> does not specify an object that exists.  
2054 In this case the <deleteResponse> MUST specify "error='noSuchIdentifier'".
- 2055 • The <psoid> sub-element of the <deleteRequest> specifies an object that contains another  
2056 object and the <deleteRequest> does not specify "recursive='true' ". In such a case  
2057 the provider should return "error='containerNotEmpty'".

### 2058 3.6.1.5.3 delete Examples (non-normative)

2059 In the following example, a requestor asks a provider to delete an existing Person object.

```
<deleteRequest requestID="120">
  <psoid ID="2244" targetID="target2"/>
</deleteRequest>
```

2060 The provider returns a <deleteResponse> element. The "status" attribute of the  
2061 <deleteResponse> element indicates that the delete request was successfully processed. The  
2062 <deleteResponse> contains no other data.

---

```
<deleteResponse requestID="120" status="success"/>
```

---

2063

2064

2065

## 2066 **3.6.2 Async Capability**

2067 The Async Capability is defined in a schema associated with the following XML namespace:  
2068 `urn:oasis:names:tc:SPML:2:0:async`. The Async Capability XSD is included as Appendix B  
2069 to this document.

2070 A provider that supports asynchronous execution of requested operations for a target SHOULD  
2071 declare that the target supports the Async Capability. A provider that does not support  
2072 asynchronous execution of requested operations for a target MUST NOT declare that the target  
2073 supports the Async Capability.

2074 **IMPORTANT:** The Async Capability does NOT define an operation specific to requesting  
2075 asynchronous execution. A provider that supports the Async Capability (for a schema entity of  
2076 which each object that the requestor desires to manipulate is an instance):

- 2077 1) MUST allow a requestor to specify "`executionMode='asynchronous'`".  
2078 The provider MUST NOT fail such a request with  
2079 "`error='unsupportedExecutionMode'`".  
2080 The provider MUST execute the requested operation asynchronously  
2081 (if the provider executes the requested operation at all).  
2082 See the section titled "[Requestor specifies asynchronous execution \(normative\)](#)".
- 2083 2) MAY choose to execute a requested operation asynchronously  
2084 when the request does not specify the "`executionMode`" attribute.  
2085 See the section titled "[Provider chooses asynchronous execution \(normative\)](#)".

2086 The Async Capability also defines two operations that a requestor may use to manage another  
2087 operation that a provider is executing asynchronously:  
2088 • A status operation allows a requestor to check the status (and possibly results) of an operation.  
2089 • A cancel operation asks the provider to stop executing an operation.

2090 **Status.** When a provider is executing SPML operations asynchronously, the requestor needs a way  
2091 to check the status of requests. The [status](#) operation allows a requestor to determine whether an  
2092 asynchronous operation has succeeded or has failed or is still pending. The [status](#) operation also  
2093 allows a requestor to obtain the output of an asynchronous operation.

2094 **Cancel.** A requestor may also need to cancel an asynchronous operation. The cancel operation  
2095 allows a requestor to ask a provider to [stop executing](#) an asynchronous operation.

2096 **Synchronous.** Both the status and cancel operations must be executed synchronously. Because  
2097 both cancel and status operate on other operations that a provider is executing asynchronously, it  
2098 would be confusing to execute cancel or status asynchronously. For example, what would it mean  
2099 to get the status of a status operation? Describing the expected behavior (or interpreting the result)  
2100 of canceling a cancel operation would be difficult, and the chain (e.g., canceling a request to cancel  
2101 a cancelRequest) could become even longer if status or cancel were supported asynchronously.

2102 **Resource considerations.** A provider must limit the size and duration of its asynchronous  
2103 operation results (or that provider will exhaust available resources). A provider must decide:

- 2104 • *How many resources* the provider will devote to storing the results of operations  
2105 that are executed asynchronously (so that the requestor may obtain the results).
- 2106 • *For how long a time* the provider will store the results of each operation  
2107 that is executed asynchronously.

- 2108 These decisions may be governed by the provider's implementation, by its configuration, or by  
2109 runtime computation.
- 2110 A provider that wishes to *never to store the results of operations* SHOULD NOT declare that it  
2111 supports the Async Capability. (Such a provider may *internally* execute requested operations  
2112 asynchronously, but must respond to each request exactly as if the request had been processed  
2113 synchronously.)
- 2114 A provider that wishes to support the asynchronous execution of requested operations MUST store  
2115 the results of an asynchronous operation *for a reasonable period of time* in order to allow the  
2116 requestor to obtain those results. SPMLv2 does not specify a minimum length of time.
- 2117 As a practical matter, a provider cannot queue the results of asynchronous operations forever. The  
2118 provider must eventually release the resources associated with asynchronous operation results.  
2119 (Put differently, a provider must eventually discard the results of an operation that the provider  
2120 executes asynchronously.) Otherwise, the provider may run out of resources.
- 2121 Providers should carefully manage the resources associated with operation results. For example:
- 2122 • A provider may define a *timeout interval* that specifies the maximum time between status  
2123 requests. If a requestor does not request the status of asynchronous operation within this  
2124 interval, the provider will release the results of the asynchronous operation.  
2125 (Any subsequent request for status on this asynchronous operation will receive a response  
2126 that specifies "error='noSuchRequest'".)
  - 2127 • A provider may also define an overall *result lifetime* that specifies the maximum length of time  
2128 to retain the results of an asynchronous operation. After this amount of time has passed, the  
2129 provider will release the results of the operation.
  - 2130 • A provider may also wish to enforce an *overall limit* on the resources available to store the  
2131 results of asynchronous operations, and may wish to adjust its behavior (or even to refuse  
2132 requests for asynchronous execution) accordingly.
  - 2133 • To prevent denial of service attacks, the provider should not allocate any resource on behalf of  
2134 a requestor until that requestor is properly authenticated.  
2135 See the section titled "[Security and Privacy Considerations](#)".

### 2136 **3.6.2.1 cancel**

2137 The cancel operation enables a requestor to stop the execution of an asynchronous operation. (The  
2138 cancel operation itself must be synchronous.)

2139 The subset of the Async Capability XSD that is most relevant to the cancel operation follows.

```
<complexType name="CancelRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <attribute name="asyncRequestID" type="xsd:string"
use="required"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="CancelResponseType">
  <complexContent>
    <extension base="spml:ResponseType">

```

```
<attribute name="asyncRequestID" type="xsd:string"
use="required"/>
  </extension>
</complexContent>
</complexType>

<element name="cancelRequest" type="spmlasync:CancelRequestType"/>
<element name="cancelResponse" type="spmlasync:CancelResponseType"/>
```

2140 **Cancel must be synchronous.** Because cancel operates on another operation that a provider is  
2141 executing asynchronously, the cancel operation itself must be synchronous. (To do otherwise  
2142 permits unnecessary confusion. What should happen when one cancels a cancel operation?)

2143 **Cancel is not batchable.** Because the cancel operation must be synchronous, a requestor must  
2144 not nest a cancel request in a [batch](#) request.

#### 2145 [3.6.2.1.1](#) *cancelRequest (normative)*

2146 A requestor **MUST** send a `<cancelRequest>` to a provider in order to (ask the provider to) cancel  
2147 a requested operation that the provider is executing asynchronously.

2148 **Execution.** A `<cancelRequest>` **MUST NOT** specify "executionMode='asynchronous'".  
2149 A `<cancelRequest>` **MUST** specify "executionMode='synchronous' "  
2150 or (a `<cancelRequest>` **MUST**) omit the "executionMode" attribute.  
2151 See the section titled "[Determining execution mode](#)".

2152 **AsyncRequestID.** A `<cancelRequest>` **MUST** have an "asyncRequestID" attribute that  
2153 specifies the operation to cancel.

#### 2154 [3.6.2.1.2](#) *cancelResponse (normative)*

2155 A provider that receives a `<cancelRequest>` from a requestor that the provider trusts **MUST**  
2156 examine the content of the request. If the request is valid, the provider **MUST** stop the execution of  
2157 the operation (that the "asyncRequestID" attribute of the `<cancelRequest>` specifies) if it is  
2158 possible for the provider to do so.

2159 • If the provider is already executing the specified operation asynchronously,  
2160 then the provider **MUST** *terminate execution* of the specified operation.

2161 • If the provider plans to execute the specified operation asynchronously  
2162 but has not yet begun to execute the specified operation,  
2163 then the provider **MUST** *prevent execution* of the specified operation.

2164 **Execution.** The provider **MUST** execute the cancel operation synchronously (if the provider  
2165 executes the cancel operation at all). See the section titled "[Determining execution mode](#)".

2166 **Response.** The provider must return to the requestor a `<cancelResponse>`.

2167 **Status.** A `<cancelResponse>` must have a "status" attribute that indicates whether the  
2168 provider successfully processed the request to cancel the specified operation.  
2169 See the section titled "[Status \(normative\)](#)".

2170 Since the provider must execute a cancel operation synchronously, the `<cancelResponse>`  
2171 **MUST NOT** specify "status='pending'". The `<cancelResponse>` **MUST** specify  
2172 "status='success'" or (the `<cancelResponse>` **MUST** specify) "status='failure'".

2173 If the provider successfully canceled the specified operation, the `<cancelResponse>` MUST  
2174 specify `status='success'`. If the provider failed to cancel the specified operation, the  
2175 `<cancelResponse>` MUST specify `status='failure'`.

2176 **Error.** If the provider cannot cancel the specified operation, the `<cancelResponse>` MUST  
2177 contain an `error` attribute that characterizes the failure.  
2178 See the general section titled ["Error \(normative\)"](#).

2179 In addition, the `<cancelResponse>` MUST specify an appropriate value of `error` if any of the  
2180 following is true:

- 2181 • The `asyncRequestID` attribute of the `<cancelRequest>` has no value. In this case, the  
2182 `<cancelResponse>` SHOULD specify `error='invalidIdentifier'`.
- 2183 • The `asyncRequestID` attribute of the `<cancelRequest>` does not specify an operation  
2184 that exists. In this case the provider SHOULD return `error='noSuchRequest'`.

### 2185 [3.6.2.1.3 cancel Examples \(non-normative\)](#)

2186 In order to illustrate the cancel operation, we must first execute an operation asynchronously. In the  
2187 following example, a requestor first asks a provider to delete a `Person` asynchronously.

```
<deleteRequest >  
  <psolD ID="2244" targetID="target2"/>  
</deleteRequest>
```

2188 The provider returns a `<deleteResponse>` element. The `status` attribute of the  
2189 `<deleteResponse>` element indicates that the provider has chosen to execute the delete  
2190 operation asynchronously. The `<deleteResponse>` also returns a `requestID`.

```
<deleteResponse status="pending" requestID="8488"/>
```

2191 Next, the same requestor asks the provider to cancel the delete operation. The requestor specifies  
2192 the value of `requestID` from the `<deleteResponse>` as the value of `asyncRequestID` in  
2193 the `<cancelRequest>`.

```
<cancelRequest requestID="131" asyncRequestID="8488"/>
```

2194 The provider returns a `<cancelResponse>`. The `status` attribute of the `<cancelResponse>`  
2195 indicates that the provider successfully canceled the delete operation.

```
<cancelResponse requestID="131" asyncRequestID="8488" status="success"/>
```

### 2196 [3.6.2.2 status](#)

2197 The status operation enables a requestor to determine whether an asynchronous operation has  
2198 completed successfully or has failed or is still executing. The status operation also (optionally)  
2199 enables a requestor to obtain results of an asynchronous operation. (The status operation itself  
2200 must be synchronous.)

2201 The subset of the Async Capability XSD that is most relevant to the status operation is shown  
2202 below for the convenience of the reader.

```

    <complexType name="StatusRequestType">
      <complexContent>
        <extension base="spml:RequestType">
          <attribute name="asyncRequestID" type="xsd:string"
use="optional"/>
          <attribute name="returnResults" type="xsd:boolean"
use="optional" default="false"/>
        </extension>
      </complexContent>
    </complexType>

    <complexType name="StatusResponseType">
      <complexContent>
        <extension base="spml:ResponseType">
          <attribute name="asyncRequestID" type="xsd:string"
use="optional"/>
        </extension>
      </complexContent>
    </complexType>

    <element name="statusRequest" type="spmlasync:StatusRequestType"/>
    <element name="statusResponse" type="spmlasync:StatusResponseType"/>

```

2203 **Status must be synchronous.** The status operation acts on other operations that a provider is  
 2204 executing asynchronously. The status operation itself therefore must be synchronous. (To do  
 2205 otherwise permits unnecessary confusion. What should be the status of a status operation?)

2206 **Status is not batchable.** Because the status operation must be synchronous, a requestor must not  
 2207 nest a status request in a [batch](#) request.

### 2208 [3.6.2.2.1](#) *statusRequest (normative)*

2209 A requestor **MUST** send a `<statusRequest>` to a provider in order to obtain the status or results  
 2210 of a requested operation that the provider is executing asynchronously.

2211 **Execution.** A `<statusRequest>` **MUST NOT** specify "executionMode='asynchronous'". A  
 2212 `<statusRequest>` **MUST** specify "executionMode='synchronous'" or (a  
 2213 `<statusRequest>` **MUST**) omit "executionMode".  
 2214 See the section titled "[Determining execution mode](#)".

2215 **AsyncRequestID.** A `<statusRequest>` **MAY** have an "asyncRequestID" attribute that  
 2216 specifies one operation for which to return status or results. A `<statusRequest>` that omits  
 2217 "asyncRequestID" implicitly requests the status of *all* operations that the provider has executed  
 2218 asynchronously on behalf of the requestor (and for which operations the provider still retains status  
 2219 and results).

2220 **returnResults.** A `<statusRequest>` **MAY** have a "returnResults" attribute that specifies  
 2221 whether the requestor wants the provider to return any results (or output) of the operation that is  
 2222 executing asynchronously. If a `<statusRequest>` does not specify "returnResults", the  
 2223 requestor has implicitly asked that the provider return only the "status" of the operation that is  
 2224 executing asynchronously.

### 2225 3.6.2.2.2 *statusResponse (normative)*

2226 A provider that receives a `<statusRequest>` from a requestor that the provider trusts MUST  
2227 examine the content of the request. If the request is valid, the provider MUST return the status  
2228 (and, if requested, any result) of the operation (that the `"asyncRequestID"` attribute of the  
2229 `<statusRequest>` specifies) if it is possible for the provider to do so.

2230 **Execution.** The provider MUST execute the status operation synchronously (if the provider  
2231 executes the status operation at all). See the section titled "[Determining execution mode](#)".

2232 **ReturnResults.** A `<statusRequest>` MAY have a `"returnResults"` attribute that indicates  
2233 whether the requestor wants the provider to return in each nested response (in addition to status,  
2234 which is always returned) *any results* of (i.e., output or XML content of the response element for)  
2235 the operation that is executing asynchronously.

2236 • If a `<statusRequest>` specifies `"returnResults='true'"`, then the provider MUST also  
2237 return in the `<statusResponse>` any results (or output) of each operation.

2238 • If a `<statusRequest>` specifies `"returnResults='false'"`, then the provider MUST  
2239 return in the `<statusResponse>` only the `"status"` of the each operation.

2240 • If the `<statusRequest>` does not specify a value for `"returnResults"`, the provider MUST  
2241 assume that the requestor wants only the `"status"` (and the provider MUST NOT return in  
2242 the `<statusResponse>` any result) of the operation that is executing asynchronously.

2243 **Response.** The provider must return to the requestor a `<statusResponse>`.

2244 **Status.** A `<statusResponse>` must have a `"status"` attribute that indicates whether the  
2245 provider successfully obtained the status of the specified operation (and obtained any results of the  
2246 specified operation if the `<statusRequest>` specifies `"returnResults='true'"`).  
2247 See the section titled "[Status \(normative\)](#)".

2248 Since the provider must execute a status operation synchronously, the `<statusResponse>`  
2249 MUST NOT specify `"status='pending'"`. The `<statusResponse>` MUST specify  
2250 `"status='success'"` or (the `<statusResponse>` MUST specify) `"status='failure'"`.

2251 • If the provider successfully obtained the status of the specified operation (and successfully  
2252 obtained any output of the specified operation if the `<statusRequest>` specifies  
2253 `"returnOutput='true'"`), the `<statusResponse>` MUST specify `"status='success'"`.

2254 • If the provider failed to obtain the status of the specified operation (or failed to obtain any output  
2255 of the specified operation if the `<statusRequest>` specifies `"returnOutput='true'"`), the  
2256 `<statusResponse>` MUST specify `"status='failure'"`.

2257 **Nested Responses.** A `<statusResponse>` MAY contain any number of responses. Each  
2258 response is an instance of a type that extends `{ResponseType}`. Each response represents an  
2259 operation that the provider is executing asynchronously.

2260 • A `<statusResponse>` that specifies `"status='failure'"` MUST NOT contain an  
2261 embedded response. Since the status operation failed, the response should not contain data.

2262 • A `<statusResponse>` that specifies `"status='success'"` MAY contain any number of  
2263 responses.

2264 - If the `<statusRequest>` specifies `"asyncRequestID"`,  
2265 then a successful `<statusResponse>` MUST contain *exactly one nested response*  
2266 that represents the operation that `"asyncRequestID"` specifies.

2267 - If the `<statusRequest>` omits `"asyncRequestID"`,  
2268 then a successful `<statusResponse>` MUST contain a *nested response for each*  
2269 *operation* that the provider has executed asynchronously as the result of a request from  
2270 that requestor (and for which operation the provider still retains status and results).

2271 **Nested Response RequestID.** Each nested response MUST have a `"requestID"` attribute that  
2272 identifies the corresponding operation (within the namespace of the provider).

2273 **Nested Response Status.** Each nested response MUST have a `"status"` attribute that  
2274 specifies the current state of the corresponding operation.

2275 • A nested response that represents an operation that failed  
2276 MUST specify `"status='failure'"`.

2277 • A nested response that represents an operation that succeeded  
2278 MUST specify `"status='success'"`.

2279 • A nested response that represents an operation that the provider is still executing  
2280 MUST specify `"status='pending'"`.

2281 **Nested Response and ReturnResults.** If a `<statusRequest>` specifies  
2282 `"returnResults='true'"`, then each response that is nested in the `<statusResponse>`  
2283 MUST contain any output *thus far produced* by the corresponding operation.

2284 • A nested response that specifies `"status='success'"` MUST contain *all* of the output that  
2285 would have been contained in a synchronous response for the operation if the provider had  
2286 executed the specified operation synchronously.

2287 • A nested response that specifies `"status='pending'"` MUST contain *an initial subset* of the  
2288 output that would have been contained in a synchronous response for the operation if the  
2289 provider had executed the specified operation synchronously.

2290 **Error.** If the provider cannot obtain the status of the specified operation, the `<statusResponse>`  
2291 MUST contain an `"error"` attribute that characterizes the failure.  
2292 See the general section titled "[Error \(normative\)](#)".

2293 In addition, a `<statusResponse>` MUST specify an appropriate value of `"error"` if any of the  
2294 following is true:

2295 • The `"asyncRequestID"` attribute of the `<statusRequest>` has no value. In this case, the  
2296 `<statusResponse>` SHOULD specify `"error='invalidIdentifier'"`.

2297 • The `"asyncRequestID"` attribute of the `<statusRequest>` has a value, but does not  
2298 identify an operation for which the provider retains status and results.  
2299 In this case the provider SHOULD return `"error='noSuchRequest'"`.

### 2300 [3.6.2.2.3 status Examples \(non-normative\)](#)

2301 In order to illustrate the status operation, we must first execute an operation asynchronously. In this  
2302 example, a requestor first asks a provider to add a `Person` asynchronously.

```
<addRequest targetID="target2" executionMode="asynchronous">
  <containerID ID="ou=Development, org=Example" />
  <data>
    <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob Briggs">
      <email>joebob@example.com</email>
    </Person>
```

```
</data>
</addRequest>
```

2303 The provider returns an `<addResponse>`. The `"status"` attribute of the `<addResponse>`  
2304 indicates that provider will execute the delete operation asynchronously. The `<addResponse>` also  
2305 has a `"requestID"` attribute (even though the original `<addRequest>` did not specify  
2306 `"requestID"`).

2307 If the original `<addRequest>` had specified a `"requestID"`, then the `<addResponse>` would  
2308 specify the same `"requestID"` value.

```
<addResponse status="pending" requestID="8489"/>
```

2309 The same requestor then asks the provider to obtain the status of the add operation. The requestor  
2310 does not ask the provider to include any output of the add operation.

```
<statusRequest requestID="117" asyncRequestID="8489"/>
```

2311 The provider returns a `<statusResponse>`. The `"status"` attribute of the `<statusResponse>`  
2312 indicates that the provider successfully obtained the status of the add operation.

2313 The `<statusResponse>` also contains a nested `<addResponse>` that represents the add  
2314 operation. The `<addResponse>` specifies `"status='pending'"`, which indicates that the add  
2315 operation has not completed executing.

```
<statusResponse requestID="117" status="success">
  <addResponse status="pending" requestID="8489"/>
</statusResponse>
```

2316 Next, the same requestor asks the provider to obtain the status of the add operation. This time the  
2317 requestor asks the provider to include any results of the add operation.

```
<statusRequest requestID="116" asyncRequestID="8489" returnResults="true"/>
```

2318 The provider again returns a `<statusResponse>`. The `"status"` attribute of the  
2319 `<statusResponse>` again indicates that the provider successfully obtained the status of the add  
2320 operation.

2321 The `<statusResponse>` again contains a nested `<addResponse>` that represents the add  
2322 operation. The `<addResponse>` specifies `"status='pending'"`, which indicates that the add  
2323 operation still has not completed executing.

2324 Because the `statusRequest` specified `"returnOutput='true'"`, the `<addResponse>` contains  
2325 an initial subset of the output that the add operation will eventually produce if the add operation  
2326 successfully completes. The `<pso>` element already contains the `Person` data that was supplied in  
2327 the `<addRequest>` but the `<pso>` element does not yet contain the `<psoID>` element that will be  
2328 generated when the add operation is complete.

```
<statusResponse requestID="116" status="success">
  <addResponse status="pending" requestID="8489">
    <pso>
      <data>
        <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob
Briggs">
          <email>joebob@example.com</email>
        </Person>
      </data>
    </pso>
  </addResponse>
</statusResponse>
```

2329 Finally, the same requestor asks the provider to obtain the status of the add operation. The  
2330 requestor again asks the provider to include any output of the add operation.

```
<statusRequest requestID="115" asyncRequestID="8489" returnResults="true"/>
```

2331 The provider again returns a <statusResponse>. The "status" attribute of the  
2332 <statusResponse> again indicates that the provider successfully obtained the status of the add  
2333 operation.

2334 The <statusResponse> again contains a nested <addResponse> that represents the add  
2335 operation. The <addResponse> specifies "status=' success'", which indicates that the add  
2336 operation completed successfully.

2337 Because the <statusRequest> specified "returnResults=' true'" and because the  
2338 <addResponse> specifies "status=' success'", the <addResponse> now contains all of the  
2339 output of the add operation. The <pso> element contains the <Person> data that was supplied in  
2340 the <addRequest> and the <pso> element also contains the <psoID> element that was missing  
2341 earlier.

```
<statusResponse requestID="115" status="success">  
  <addResponse status="pending" requestID="8489">  
    <pso>  
      <data>  
        <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob  
Briggs">  
          <email>joebob@example.com</email>  
        </Person>  
      </data>  
      <psoID ID="2244" targetID="target2"/>  
    </pso>  
  </addResponse>  
</statusResponse>
```

2342

2343

### 2344 **3.6.3 Batch Capability**

2345 The Batch Capability is defined in a schema associated with the following XML namespace:  
2346 urn:oasis:names:tc:SPML:2:0:batch. The Batch Capability XSD is included as Appendix C  
2347 to this document.

2348 A provider that supports batch execution of requested operations for a target SHOULD declare that  
2349 the target supports the Batch Capability. A provider that does not support batch execution of  
2350 requested operations MUST NOT declare that the target supports the Batch Capability.

2351 The Batch Capability defines one operation: batch.

#### 2352 **3.6.3.1 batch**

2353 The subset of the Batch Capability XSD that is most relevant to the batch operation follows.

```
<simpleType name="ProcessingType">
  <restriction base="string">
    <enumeration value="sequential"/>
    <enumeration value="parallel"/>
  </restriction>
</simpleType>

<simpleType name="OnErrorType">
  <restriction base="string">
    <enumeration value="resume"/>
    <enumeration value="exit"/>
  </restriction>
</simpleType>

<complexType name="BatchRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <annotation>
        <documentation>Elements that extend spml:RequestType
      </documentation>
      </annotation>
      <attribute name="processing" type="spmlbatch:ProcessingType"
use="optional" default="sequential"/>
      <attribute name="onError" type="spmlbatch:OnErrorType"
use="optional" default="exit"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="BatchResponseType">
  <complexContent>
    <extension base="spml:ResponseType">
      <annotation>
        <documentation>Elements that extend spml:ResponseType
      </documentation>
      </annotation>
    </extension>
  </complexContent>
</complexType>
```

```
</complexContent>
</complexType>

<element name="batchRequest" type="spmlbatch:BatchRequestType"/>
<element name="batchResponse" type="spmlbatch:BatchResponseType"/>
```

2354 The batch operation combines any number of individual requests into a single request.

2355 **No transactional semantics.** Using a batch operation to combine individual requests does not  
2356 imply atomicity (i.e., “all-or-nothing” semantics) for the group of batched requests. A requestor must  
2357 not assume that the failure of a nested request will undo a nested request that has already  
2358 completed. (See the section titled “[Transactional Semantics](#)”.)

2359 Note that this does not *preclude* a batch operation having transactional semantics—this is merely  
2360 unspecified. A provider (or some higher-level service) with the ability to undo specific operations  
2361 could support rolling back an entire batch if an operation nested within the batch fails.

2362 **Nested Requests.** The Core XSD defines `{RequestType}` as the base type for any SPML  
2363 request. A requestor may group into a `<batchRequest>` any number of requests that derive from  
2364 `{RequestType}`. However, there are some exceptions. See the topics named “Batch is not  
2365 batchable” and “Some operations are not batchable” immediately below.

2366 **Batch is not batchable.** A requestor must not nest a batch request within another batch request.  
2367 (To support nested batches would impose on each provider a burden of complexity that the benefits  
2368 of nested batches do not justify.)

2369 **Some operations are not batchable.** For various reasons, a requestor must not nest certain  
2370 types of requests within a batch request. For example, a request to [listTargets](#) must not be batched  
2371 (because a requestor cannot know until the requestor examines the response from [listTargets](#)  
2372 whether the provider supports the batch capability). Requests to [search](#) for objects (and requests  
2373 to [iterate](#) the results of a search) must not be batched for reasons of scale. Batching requests to  
2374 [cancel](#) and obtain the [status](#) of asynchronous operations would introduce timing problems.

2375 **Positional correspondence.** The provider’s `<batchResponse>` contains an individual response  
2376 for each individual request that the requestor’s `<batchRequest>` contained. Each individual  
2377 response occupies the same position within the `<batchResponse>` that the corresponding  
2378 individual request occupied within the `<batchRequest>`.

2379 **Processing.** A requestor can specify whether the provider executes the individual requests *one-by-*  
2380 *one in the order that they occur* within a `<batchRequest>`. The “processing” attribute of a  
2381 `<batchRequest>` controls this behavior.

2382 • When a `<batchRequest>` specifies “processing=’sequential’”, the provider must  
2383 execute each requested operation *one at a time and in the exact order* that it occurs within the  
2384 `<batchRequest>`.

2385 • When a `<batchRequest>` specifies “processing=’parallel’”, the provider may execute  
2386 the requested operations within the `<batchRequest>` *in any order*.

2387 **Individual errors.** The “onError” attribute of a `<batchRequest>` specifies whether the provider  
2388 quits at the first error it encounters (in processing individual requests within a `<batchRequest>`) or  
2389 continues despite any number of such errors.

2390 • When a `<batchRequest>` specifies “onError=’exit’”, the provider stops executing  
2391 individual operations within the batch as soon as the provider encounters an error.  
2392 Any operation that produces an error is marked as failed.  
2393 Any operation that the provider does not execute is also marked as failed.

- 2394 • When a <batchRequest> specifies “onError=’ resume’”, the provider handles any error  
2395 that occurs in processing an individual operation within that <batchRequest>.  
2396 No error that occurs in processing an individual operation prevents execution of any other  
2397 individual operation in the batch.  
2398 Any operation that produces an error is marked as failed.

2399 (Note that a requestor can guarantee pre-requisite processing in batch operations by specifying  
2400 both “processing=’sequential’” and “onError=’exit’”.)

2401 **Overall error.** When a requestor issues a <batchRequest> with “onError=’ resume’” and one  
2402 or more of the requests in that batch fails, then the provider will return a <batchResponse> with  
2403 “status=’ failure’” (even if some of the requests in that batch succeed). The requestor must  
2404 examine every individual response within the overall <batchResponse> to determine which  
2405 requests succeeded and which requests failed.

### 2406 3.6.3.1.1 *batchRequest (normative)*

2407 A requestor MUST send a <batchRequest> to a provider in order to (ask the provider to) execute  
2408 multiple requests as a set.

2409 **Nested Requests.** A <batchRequest> MUST contain at least one element that extends  
2410 {RequestType}.

2411 A <batchRequest> MUST NOT contain as a nested request an element that is of any the  
2412 following types:

- 2413 • {spml:ListTargetsRequestType}
- 2414 • {spmlbatch:BatchRequestType}
- 2415 • {spmlsearch:SearchRequestType}
- 2416 • {spmlsearch:IterateRequestType}
- 2417 • {spmlsearch:CloseIteratorRequestType}
- 2418 • {spmlasync:CancelRequestType}
- 2419 • {spmlasync:StatusRequestType}
- 2420 • {spmlupdates:UpdatesRequestType}
- 2421 • {spmlupdates:IterateRequestType}
- 2422 • {spmlupdates:CloseIteratorRequestType}

2423 **Processing.** A <batchRequest> MAY specify “processing”. The value of any “processing”  
2424 attribute MUST be either ‘sequential’ or ‘parallel’.

2425 • A requestor who wants the provider to process the nested requests *concurrently with one*  
2426 *another* MUST specify “processing=’ parallel’”.

2427 • A requestor who wants the provider to process the nested requests one-by-one and in the  
2428 order that they appear MAY specify “processing=’ sequential’”.

2429 • A requestor who does not specify “processing” is *implicitly* asking the provider to process  
2430 the nested requests *sequentially*.

2431 **onError.** A <batchRequest> MAY specify “onError”. The value of any “onError” attribute  
2432 MUST be either ‘exit’ or ‘resume’.

2433 • A requestor who wants the provider to *continue processing* nested requests whenever  
2434 processing one of the nested requests produces in an error MUST specify  
2435 “onError=’ resume’”.

- 2436 • A requestor who wants the provider to *cease processing* nested requests as soon as  
2437 processing any of the nested requests produces an error MAY specify “onError=’ exit’”.
- 2438 • A requestor who does not specify an “onError” attribute *implicitly* asks the provider to cease  
2439 processing nested requests as soon as processing any of the nested requests produces an  
2440 error.

### 2441 3.6.3.1.2 *batchResponse (normative)*

2442 The provider must examine the content of the <batchRequest>. If the request is valid, the  
2443 provider MUST process each nested request (according to the effective “processing” and  
2444 “onError” settings) if the provider possibly can.

2445 **processing.** If a <batchRequest> specifies “processing=’ parallel’”, the provider SHOULD  
2446 begin executing each of the nested requests as soon as possible. (Ideally, the provider would begin  
2447 executing all of the nested requests immediately and concurrently.) If the provider cannot begin  
2448 executing all of the nested requests at the same time, then the provider SHOULD begin executing  
2449 *as many as possible* of the nested requests *as soon as possible*.

2450 If a <batchRequest> specifies (or defaults to) “processing=’ sequential’”, the provider  
2451 MUST execute each of the nested requests one-by-one and in the order that each appears within  
2452 the <batchRequest>. The provider MUST complete execution of each nested request before the  
2453 provider begins to execute the next nested request.

2454 **onError.** The effect (on the provider’s behavior) of the “onError” attribute of a <batchRequest>  
2455 depends on the “processing” attribute of the <batchRequest>.

- 2456 • If a <batchRequest> specifies (or defaults to) “onError=’ exit’” and (the  
2457 <batchRequest> specifies or defaults to) “processing=’ sequential’” then the provider  
2458 MUST NOT execute any (operation that is described by a) nested request that is subsequent to  
2459 the first nested request that produces an error.

2460  
2461 If the provider encounters an error in executing (the operation that is described by) a nested  
2462 request, the provider MUST report the error in the nested response that corresponds to the  
2463 nested request and then (the provider MUST) specify “status=’ failure’” in every nested  
2464 response that corresponds to a subsequent nested request within the same  
2465 <batchRequest>. The provider MUST also specify “status=’ failure’” in the overall  
2466 <batchResponse>.

- 2467 • If a <batchRequest> specifies (or defaults to) “onError=’ exit’” and (the  
2468 <batchRequest> specifies) “processing=’ parallel’” then the provider’s behavior once  
2469 an error occurs (in processing an operation that is described by a nested request) is *not fully*  
2470 *specified*.

2471  
2472 If the provider encounters an error in executing (the operation that is described by) a nested  
2473 request, the provider MUST report the error in the nested response that corresponds to the  
2474 nested request. The provider MUST also specify “status=’ failure’” in the overall  
2475 <batchResponse>. The provider MUST also specify “status=’ failure’” in the nested  
2476 response that corresponds to any operation the provider has not yet begun to execute.  
2477 However, the provider’s behavior with respect to any operation that has already begun to  
2478 execute but that is not yet complete is not fully specified.

2479  
2480 The provider MAY stop executing any (operation that is described by a) nested request that has  
2481 not yet completed or (the provider MAY) choose to complete the execution of any (operation  
2482 that corresponds to a) nested request (within the same <batchRequest> and) for which the

2483 provider has already begun execution. The provider SHOULD NOT begin to execute any  
2484 operation (that corresponds to a nested request within the same <batchRequest> and) for  
2485 which the provider has not yet begun execution.

2486 • If a <batchRequest> specifies “onError=’ resume’ ” and (the <batchRequest> specifies)  
2487 “processing=’ parallel’ ”, then the provider MUST execute every (operation that is  
2488 described by a) nested request within the <batchRequest>. If the provider encounters an  
2489 error in executing any (operation that is described by a) nested request, the provider MUST  
2490 report the error in the nested response that corresponds to the nested request and then (the  
2491 provider MUST) specify “status=’ failure’ ” in the overall <batchResponse>.

2492 • If a <batchRequest> specifies “onError=’ resume’ ” and (the <batchRequest> specifies  
2493 or defaults to) “processing=’ sequential’ ”, then the provider MUST execute every  
2494 (operation that is described by a) nested request within the <batchRequest>. If the provider  
2495 encounters an error in executing any (operation that is described by a) nested request, the  
2496 provider MUST report the error in the nested response that corresponds to the nested request  
2497 and then (the provider MUST) specify “status=’ failure’ ” in the overall  
2498 <batchResponse>.

2499 **Response.** The provider MUST return to the requestor a <batchResponse>.

2500 **Status.** The <batchResponse> must contain a “status” attribute that indicates whether the  
2501 provider successfully processed every nested request.  
2502 See the section titled “[Status \(normative\)](#)”.

2503 • If the provider successfully executed every (operation described by a) nested request,  
2504 then the <batchResponse> MUST specify “status=’ success’ ”.

2505 • If the provider encountered an error in processing (the operation described by) any nested  
2506 request, the <batchResponse> MUST specify “status=’ failure’ ”.

2507 **nested Responses.** The <batchResponse> MUST contain a nested response for each nested  
2508 request that the <batchRequest> contains. Each nested response within the <batchResponse>  
2509 *corresponds positionally* to a nested request within the <batchRequest>. That is, each nested  
2510 response MUST appear in the same position within the <batchResponse> that the nested request  
2511 (to which the nested response corresponds) originally appeared within the corresponding  
2512 <batchRequest>.

2513 The content of each nested response depends on whether the provider actually executed the  
2514 nested operation that corresponds to the nested response.

2515 • Each nested response that corresponds to a nested request *that the provider did not process*  
2516 MUST specify “status=’ failed’ ”. (A provider might not process a nested request, for  
2517 example, if the provider encountered an error processing an earlier nested request and the  
2518 requestor specified both “processing=’ sequential’ ” and “onError=’ exit’ ”.)

2519 • Each nested response that corresponds to a nested request for an operation *that the provider*  
2520 *actually executed* MUST contain the same data that the provider would have returned (in the  
2521 response for the corresponding operation) *if the corresponding operation had been requested*  
2522 *individually* (rather than as part of a batch operation).

2523 **Error.** If something (other than the behavior specified by the “onError” setting with respect to  
2524 errors that occur in processing nested requests) prevents the provider from processing one or more  
2525 of the (operations described by the) nested requests within a <batchRequest>, then the  
2526 <batchResponse> MUST have an “error” attribute that characterizes the failure.  
2527 See the general section titled “[Error \(normative\)](#)”.

2528 **3.6.3.1.3 batch Examples (non-normative)**

2529 In the following example, a requestor asks a provider to perform a series of operations. The  
2530 requestor asks the provider first to add a `Person` object to one target and then to add an `Account`  
2531 object to another target. (These are the first two examples of the add operation.)

```
<batchRequest processing="sequential" onError="exit">
  <addRequest targetID="target2">
    <containerID ID="ou=Development, org=Example"/>
    <data>
      <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob
Briggs">
        <email>joebob@example.com</email>
      </Person>
    </data>
  </addRequest>

  <addRequest targetID="target1">
    <data>
      <Account accountName="joebob"/>
    </data>
    <capabilityData mustUnderstand="true"
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="memberOf">
        <toPsoID ID="group1" targetID="target1"/>
      </reference>
      <reference typeOfReference="owner">
        <toPsoID ID="2244" targetID="target2"/>
      </reference>
    </capabilityData>
  </addRequest>
</batchRequest>
```

2532 The provider returns an `<batchResponse>` element. The "status" of the `<batchResponse>`  
2533 indicates that all of the nested requests were processed successfully. The `<batchResponse>`  
2534 contains an `<addResponse>` for each `<addRequest>` that the `<batchRequest>` contained.  
2535 Each `<addResponse>` contains the same data that it would have contained if the corresponding  
2536 `<addRequest>` had been requested individually.

```
<batchResponse status="success">
  <addResponse status="success">
    <pso>
      <data>
        <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob
Briggs">
          <email>joebob@example.com</email>
        </Person>
      </data>
      <psoID ID="2244" targetID="target2"/>
    </pso>
  </addResponse>

  <addResponse status="success">
    <pso>
      <data>
        <Account accountName="joebob"/>
      </data>
    </pso>
  </addResponse>
</batchResponse>
```

```
</data>
  <psolD ID="1431" targetID="target1"/>
  <capabilityData mustUnderstand="true"
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
    <reference typeOfReference="memberOf">
      <toPsolD ID="group1" targetID="target1"/>
    </reference>
    <reference typeOfReference="owner">
      <toPsolD ID="2244" targetID="target2"/>
    </reference>
  </capabilityData>
</pso>
</addResponse>
</batchResponse>
```

2537

2538

### 2539 **3.6.4 Bulk Capability**

2540 The Bulk Capability is defined in a schema associated with the following XML namespace:  
2541 urn:oasis:names:tc:SPML:2:0:bulk. This document includes the Bulk Capability XSD as  
2542 Appendix D.

2543 The Bulk Capability defines two operations: bulkModify and bulkDelete.

2544 A provider that supports the bulkModify and bulkDelete operations for a target SHOULD declare  
2545 that the target supports the Bulk Capability. A provider that does not support both bulkModify and  
2546 bulkDelete MUST NOT declare that the target supports the Bulk Capability.

#### 2547 **3.6.4.1 bulkModify**

2548 The subset of the Bulk Capability XSD that is most relevant to the bulkModify operation follows.

```
<complexType name="BulkModifyRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element ref="spmlsearch:query"/>
        <element name="modification" type="spml:ModificationType"
maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="bulkModifyRequest"
type="spmlbulk:BulkModifyRequestType"/>
<element name="bulkModifyResponse" type="spml:ResponseType"/>
```

2549 The bulkModify operation applies a specified modification to every object that matches the specified  
2550 query.

2551 • The <modification> is the same type of element that is specified as part of a  
2552 <modifyRequest>.

2553 • The <query> is the same type of element that is specified as part of a <searchRequest>.

2554 **Does not return modified PSO Identifiers.** A bulkModify operation does *not* return a <psoid> for  
2555 each object that it changes, even though a bulkModify operation can change the <psoid> for every  
2556 object that it modifies. By contrast, a modify operation does return the <psoid> of any object that it  
2557 changes.

2558 The difference is that the requestor of a bulkModify operation specifies a *query* that selects objects  
2559 to be modified. The requestor of a modify operation specifies the <psoid> of the object to be  
2560 modified. The modify operation therefore must return the <psoid> to make sure that the requestor  
2561 still has the correct <psoid>.

2562 A bulkModify operation does not return a <psoid> for each object that it changes because:

- 2563 • The requestor does not specify a `<psoid>` as input. (Therefore, a changed `<psoid>` does not  
2564 necessarily interest the requestor).
- 2565 • Returning PSO Identifiers for modified objects would cause the `bulkModify` operation to scale  
2566 poorly (which would defeat the purpose of the `bulkModify` operation).

#### 2567 [3.6.4.1.1 \*bulkModifyRequest \(normative\)\*](#)

2568 A requestor MUST send a `<bulkModifyRequest>` to a provider in order to (ask the provider to)  
2569 make the same set of modifications to every object that matches specified selection criteria.

2570 **Execution.** A `<bulkModifyRequest>` MAY specify "executionMode".  
2571 See the section titled "[Determining execution mode](#)".

2572 **query.** A `<bulkModifyRequest>` MUST contain exactly one `<query>` element.  
2573 A `<query>` describes criteria that (the provider must use to) select objects on a target.  
2574 See the section titled "[SearchQueryType in a Request \(normative\)](#)".

2575 **Modification.** A `<bulkModifyRequest>` MUST contain at least one `<modification>`. Each  
2576 `<modification>` describes a set of changes to be applied (to every object that matches the  
2577 `<query>`). A requestor MUST specify each `<modification>` for a `<bulkModifyRequest>` in  
2578 the same way as for a `<modifyRequest>`.  
2579 See the topic named "Modification" within the section titled "[modifyRequest \(normative\)](#)".

#### 2580 [3.6.4.1.2 \*bulkModifyResponse \(normative\)\*](#)

2581 A provider that receives a `<bulkModifyRequest>` from a requestor that the provider trusts MUST  
2582 examine the content of the `<bulkModifyRequest>`. If the request is valid, the provider MUST  
2583 apply the (set of changes described by each of the) specified `<modification>` elements to every  
2584 object that matches the specified `<query>` (if the provider can possibly do so).  
2585 The section titled "[modifyResponse \(normative\)](#)" describes how the provider should apply each  
2586 `<modification>` to an object.

2587 **Response.** The provider MUST return to the requestor a `<bulkModifyResponse>`.

2588 **Status.** The `<bulkModifyResponse>` must contain a "status" attribute that indicates whether  
2589 the provider successfully applied every specified modification to every object that matched the  
2590 specified query. See the section titled "[Status \(normative\)](#)".

- 2591 • If the provider successfully applied every specified modification to every object that matched  
2592 the specified query, then the `<bulkModifyResponse>` MUST specify "status='success'".
- 2593 • If the provider encountered an error in selecting any object that matched the specified query or  
2594 (if the provider encountered an error) in applying any specified modification to any of the  
2595 selected objects, then the `<bulkModifyResponse>` MUST specify "status='failure'".

2596 **Error.** If the provider was unable to apply the specified modification to every object that matched  
2597 the specified query, then the `<bulkModifyResponse>` MUST have an "error" attribute that  
2598 characterizes the failure. See the general section titled "[Error \(normative\)](#)".

2599 In addition, the section titled "[SearchQueryType Errors \(normative\)](#)" describes errors specific to a  
2600 request that contains a `<query>`.

2601 **3.6.4.1.3** *bulkModify Examples (non-normative)*

2602 In the following example, a requestor asks a provider to change every `Person` with an email  
2603 address matching `'jbbbriggs@example.com'` to have instead an email address of  
2604 `'joebob@example.com'`.

```
<bulkModifyRequest>
  <query scope="subtree" targetID="target2">
    <select path="/Person/email='jbbbriggs@example.com'"
namespaceURI="http://www.w3.org/TR/xpath20" />
  </query>
  <modification modificationMode="replace">
    <component path="/Person/email" namespaceURI="http://www.w3.org/TR/xpath20" />
    <data>
      <email>joebob@example.com</email>
    <data>
  </modification>
</bulkModifyRequest>
```

2605 The provider returns a `<bulkModifyResponse>`. The `"status"` attribute of the  
2606 `<bulkModifyResponse>` indicates that the provider successfully executed the `bulkModify`  
2607 operation.

```
<bulkModifyResponse status="success"/>
```

2608 In the following example, a requestor asks a provider to remove the "owner" of any account that is  
2609 currently owned by "joebob". The requestor uses as a selection criterion the `<hasReference>`  
2610 query clause that the Reference Capability defines.

2611 NOTE: The logic required to modify a reference may depend on the cardinality that is defined for  
2612 that type of reference. See the section titled "[Reference Capability](#)". Also see the topic named  
2613 "Modifying a reference" within the section titled "[modify Examples](#)".

```
<bulkModifyRequest>
  <query scope="subtree" targetID="target2" >
    <hasReference typeOfReference="owner">
      <toPsoID ID="2244" targetID="target2"/>
    </hasReference>
  </query>
  <modification modificationMode="delete">
    <capabilityData mustUnderstand="true"
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
      <reference typeOfReference="owner"/>
    </capabilityData>
  </modification>
</bulkModifyRequest>
```

2614 The provider returns a `<bulkModifyResponse>`. The `"status"` attribute of the  
2615 `<bulkModifyResponse>` indicates that the provider successfully executed the `bulkModify`  
2616 operation.

```
<bulkModifyResponse status="success"/>
```

2617 **3.6.4.2** *bulkDelete*

2618 The subset of the Bulk Capability XSD that is most relevant to the `bulkDelete` operation follows.

```

<complexType name="BulkDeleteRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element ref="spmlsearch:query"/>
      </sequence>
      <attribute name="recursive" type="boolean" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<element name="bulkDeleteRequest"
type="spmlbulk:BulkDeleteRequestType"/>
<element name="bulkDeleteResponse" type="spml:ResponseType"/>

```

2619 The bulkDelete operation deletes every object that matches the specified query.

2620 • The <query> is the same element that is specified as part of a <searchRequest>.

### 2621 [3.6.4.2.1 bulkDeleteRequest \(normative\)](#)

2622 A requestor **MUST** send a <bulkDeleteRequest> to a provider in order to (ask the provider to)  
2623 delete every object that matches specified selection criteria.

2624 **Execution.** A <bulkDeleteRequest> **MAY** specify "executionMode".  
2625 See the section titled "[Determining execution mode](#)".

2626 **query.** A <bulkDeleteRequest> **MUST** contain exactly one <query> element.  
2627 A <query> describes criteria that (the provider must use to) select objects on a target.  
2628 See the section titled "[SearchQueryType in a Request \(normative\)](#)".

2629 **recursive.** A <bulkDeleteRequest> **MAY** have a "recursive" attribute that indicates  
2630 whether the provider should delete the specified object *along with any other object it contains*.  
2631 (Unless the <bulkDeleteRequest> specifies "recursive='true'", a provider will not delete  
2632 an object that contains other objects.)

### 2633 [3.6.4.2.2 bulkDeleteResponse \(normative\)](#)

2634 A provider that receives a <bulkDeleteRequest> from a requestor that the provider trusts must  
2635 examine the content of the <bulkDeleteRequest>. If the request is valid, the provider **MUST**  
2636 delete every object that matches the specified <query> (if the provider can possibly do so).

2637 **recursive.** A provider **MUST NOT** delete any object that contains other objects unless the  
2638 <bulkDeleteRequest> specifies "recursive='true'".

2639 • If the <bulkDeleteRequest> specifies "recursive='true'",  
2640 then the provider **MUST** delete every object that matches the specified query  
2641 *along with any object that a matching object (directly or indirectly) contains*.

2642 • If the <bulkDeleteRequest> specifies "recursive='false'"  
2643 (or if the <bulkDeleteRequest> omits the "recursive" attribute")  
2644 and at least one object that matches the specified query contains another object,  
2645 then the provider **MUST NOT** delete any of the objects that match the specified query.  
2646 In this case, the provider's response must return an error (see below).

2647 **Response.** The provider MUST return to the requestor a `<bulkDeleteResponse>`.

2648 **Status.** The `<bulkDeleteResponse>` must contain a "status" attribute that indicates whether  
2649 the provider successfully deleted every object that matched the specified query.  
2650 See the section titled "[Status \(normative\)](#)".

- 2651 • If the provider successfully deleted every object that matched the specified query, the  
2652 `<bulkDeleteResponse>` MUST specify "status='success'".
- 2653 • If the provider encountered an error in selecting any object that matched the specified query or  
2654 (if the provider encountered an error) in deleting any of the selected objects, the  
2655 `<bulkDeleteResponse>` MUST specify "status='failure'".

2656 **Error.** If the provider was unable to delete every object that matched the specified query, then the  
2657 `<bulkDeleteResponse>` MUST have an "error" attribute that characterizes the failure.  
2658 See the general section titled "[Error \(normative\)](#)".

2659 In addition, the section titled "[SearchQueryType Errors \(normative\)](#)" describes errors specific to a  
2660 request that contains a `<query>`. Also see the section titled "[SelectionType Errors \(normative\)](#)".

2661 If at least one object that matches the specified query contains another object  
2662 and the `<bulkDeleteRequest>` does NOT specify "recursive='true'",  
2663 then the provider's response should specify "error='invalidContainment'".

#### 2664 [3.6.4.2.3 bulkDelete Examples \(non-normative\)](#)

2665 In the following example, a requestor asks a provider to delete every `Person` with an email address  
2666 matching 'joebob@example.com'.

```
<bulkDeleteRequest>
  <query scope="subtree" targetID="target2" >
    <select path="/Person/email='joebob@example.com'"
namespaceURI="http://www.w3.org/TR/xpath20" />
  </query>
</bulkDeleteRequest>
```

2667 The provider returns a `<bulkDeleteResponse>`. The "status" attribute of the  
2668 `<bulkDeleteResponse>` indicates that the provider successfully executed the `bulkDelete`  
2669 operation.

```
<bulkDeleteResponse status="success"/>
```

2670 In the following example, a requestor asks a provider to delete any `Account` that is currently  
2671 owned by "joebob". The requestor uses as a selection criterion the `<hasReference>` query clause  
2672 that the Reference Capability defines.

```
<bulkDeleteRequest>
  <query scope="subtree" targetID="target2" >
    <hasReference typeOfReference="owner">
      <toPsoID ID="2244" targetID="target2"/>
    </hasReference>
  </query>
</bulkDeleteRequest>
```

2673 The provider returns a `<bulkDeleteResponse>`. The "status" attribute of the  
2674 `<bulkDeleteResponse>` indicates that the provider successfully executed the `bulkDelete`  
2675 operation.

```
<bulkDeleteResponse status="success"/>
```

## 2676 **3.6.5 Password Capability**

2677 The Password Capability is defined in a schema that is associated with the following XML  
2678 namespace: `urn:oasis:names:tc:SPML:2:0:password`. This document includes the  
2679 Password Capability XSD as Appendix E.

2680 The Password Capability defines four operations: `setPassword`, `expirePassword`, `resetPassword`  
2681 and `validatePassword`.

- 2682 • The `setPassword` operation *changes to a specified value* the password that is associated with a  
2683 specified object. The `setPassword` operation also allows a requestor to supply the current  
2684 password (in case the target system or application requires it).
- 2685 • The `expirePassword` operation *marks as no longer valid* the password that is associated with a  
2686 specified object. (Most systems or applications will require a user to change an expired  
2687 password on the next login.)
- 2688 • The `resetPassword` operation *changes to an unspecified value* the password that is associated  
2689 with a specified object. The `resetPassword` operation returns the new password.
- 2690 • The `validatePassword` operation *tests whether a specified value would be valid* as the  
2691 password for a specified object. (The `validatePassword` operation allows a requestor to test a  
2692 password value against the password policy for a system or application.)

2693 A provider that supports the `setPassword`, `expirePassword`, `resetPassword` and `validatePassword`  
2694 operations for a target SHOULD declare that the target supports the Password Capability. A  
2695 provider that does not support all of the `setPassword`, `expirePassword`, `resetPassword` and  
2696 `validatePassword` operations MUST NOT declare that the target supports the Password Capability.

### 2697 **3.6.5.1 setPassword**

2698 The `setPassword` operation enables a requestor to *specify a new password* for an object.

2699 The subset of the Password Capability XSD that is most relevant to the `setPassword` operation  
2700 follows.

```
<complexType name="SetPasswordRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType"/>
        <element name="password" type="string"/>
        <element name="currentPassword" type="string"
minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="setPasswordRequest"
type="pass:SetPasswordRequestType"/>
<element name="setPasswordResponse" type="spml:ResponseType"/>
```

2701 **3.6.5.1.1 setPasswordRequest (normative)**

2702 A requestor MUST send a <setPasswordRequest> to a provider in order to (ask the provider to)  
2703 change to a specified value the password that is associated an existing object.

2704 **Execution.** A <setPasswordRequest> MAY specify "executionMode".  
2705 See the section titled "Determining execution mode".

2706 **psoid.** A <setPasswordRequest> MUST contain exactly one <psoid> element. The <psoid>  
2707 MUST identify an object that exists on a target (that is supported by the provider).  
2708 See the section titled "PSO Identifier (normative)".

2709 **password.** A <setPasswordRequest> MUST contain exactly one <password> element. A  
2710 <password> element MUST contain a string value.

2711 **currentPassword.** A <setPasswordRequest> MAY contain at most one <currentPassword>  
2712 element. A <currentPassword> element MUST contain a string value.

2713 **3.6.5.1.2 setPasswordResponse (normative)**

2714 A provider that receives a <setPasswordRequest> from a requestor that the provider trusts  
2715 MUST examine the content of the <setPasswordRequest>. If the request is valid and if the  
2716 specified object exists, then the provider MUST change (to the value that the <password> element  
2717 contains) the password that is associated with the object that is specified by the <psoid>.

2718 **Execution.** If a <setPasswordRequest> does not specify "executionMode", the provider  
2719 MUST choose a type of execution for the requested operation.  
2720 See the section titled "Determining execution mode".

2721 **Response.** The provider must return to the requestor a <setPasswordResponse>. The  
2722 <setPasswordResponse> must have a "status" attribute that indicates whether the provider  
2723 successfully changed (to the value that the <password> element contains) the password that is  
2724 associated with the specified object. See the section titled "Status (normative)".

2725 **Error.** If the provider cannot change (to the value that the <password> element contains) the  
2726 password that is associated with the requested object, the <setPasswordResponse> must  
2727 contain an "error" attribute that characterizes the failure.  
2728 See the general section titled "Error (normative)".

2729 In addition, a <setPasswordResponse> MUST specify an error if any of the following is true:

- 2730
- The <setPasswordRequest> contains a <psoid> for an object that does not exist.
  - The target system or application will not accept (as the new password) the value that a  
2731 <setPasswordRequest> supplies as the content of the <password> element.  
2732
  - The target system or application *requires the current password* in order to change the password  
2733 and a <setPasswordRequest> supplies no content for <currentPassword>.  
2734
  - The target system or application *requires the current password* in order to change the password  
2735 and the target system or application will not accept (as the current password) the value that a  
2736 <setPasswordRequest> supplies as the content of <currentPassword>.  
2737
  - The target system or application *returns an error (or throws an exception)* when the provider  
2738 tries to set the password.  
2739

2740 **3.6.5.1.3** *setPassword Examples (non-normative)*

2741 In the following example, a requestor asks a provider to set the password for a `Person` object.

```
<setPasswordRequest requestID="133">
  <psoID ID="2244" targetID="target2"/>
  <password>y0baby</password>
  <currentPassword>corvette</currentPassword>
</setPasswordRequest>
```

2742 The provider returns a `<setPasswordResponse>` element. The "status" of the  
2743 `<setPasswordResponse>` indicates that the provider successfully changed the password.

```
<setPasswordResponse requestID="133" status="success"/>
```

2744 **3.6.5.2** *expirePassword*

2745 The `expirePassword` operation *marks as invalid the current password* for an object.

2746 The subset of the Password Capability XSD that is most relevant to the `expirePassword` operation  
2747 follows.

```
<complexType name="ExpirePasswordRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType"/>
      </sequence>
      <attribute name="remainingLogins" type="int" use="optional"
default="1"/>
    </extension>
  </complexContent>
</complexType>

<element name="expirePasswordRequest"
type="pass:ExpirePasswordRequestType"/>
<element name="expirePasswordResponse" type="spml:ResponseType"/>
```

2748 **3.6.5.2.1** *expirePasswordRequest (normative)*

2749 A requestor **MUST** send a `<expirePasswordRequest>` to a provider in order to (ask the provider  
2750 to) mark as no longer valid the password that is associated with an existing object.

2751 **Execution.** A `<expirePasswordRequest>` **MAY** specify "executionMode".

2752 See the section titled "[Determining execution mode](#)".

2753 **psoID.** A `<expirePasswordRequest>` **MUST** contain exactly one `<psoID>` element. The

2754 `<psoID>` **MUST** identify an object that exists on a target (that is supported by the provider).

2755 See the section titled "[PSO Identifier \(normative\)](#)".

2756 **remainingLogins.** A `<expirePasswordRequest>` **MAY** have a "remainingLogins" attribute  
2757 that specifies a number of grace logins that the target system or application should permit.

### 2758 3.6.5.2.2 *expirePasswordResponse* (normative)

2759 A provider that receives a `<expirePasswordRequest>` from a requestor that the provider trusts  
2760 MUST examine the content of the `<expirePasswordRequest>`. If the request is valid and if the  
2761 specified object exists, then the provider MUST mark as no longer valid the password that is  
2762 associated with the object that the `<psoID>` specifies.

2763 **Execution.** If an `<expirePasswordRequest>` does not specify "executionMode", the provider  
2764 MUST choose a type of execution for the requested operation.  
2765 See the section titled "[Determining execution mode](#)".

2766 **Response.** The provider must return to the requestor an `<expirePasswordResponse>`. The  
2767 `<expirePasswordResponse>` must have a "status" attribute that indicates whether the  
2768 provider successfully marked as no longer valid the password that is associated with the specified  
2769 object. See the section titled "[Status \(normative\)](#)" for values of this attribute.

2770 **Error.** If the provider cannot mark as invalid the password that is associated with the requested  
2771 object, the `<expirePasswordResponse>` must contain an "error" attribute that characterizes  
2772 the failure. See the general section titled "[Error \(normative\)](#)".

2773 In addition, an `<expirePasswordResponse>` MUST specify an error if any of the following is  
2774 true:

- 2775 • The `<expirePasswordRequest>` contains a `<psoID>` for an object that does not exist.
- 2776 • The target system or application will not accept (as the number of grace logins to permit) the  
2777 value that a `<expirePasswordRequest>` specifies for the "remainingLogins" attribute.
- 2778 • The target system or application *returns an error (or throws an exception)* when the provider  
2779 tries to mark as no longer valid the password that is associated with the specified object.

### 2780 3.6.5.2.3 *expirePassword Examples* (non-normative)

2781 In the following example, a requestor asks a provider to expire the password for a `Person` object.

```
<expirePasswordRequest requestID="134">  
  <psoID ID="2244" targetID="target2"/>  
</expirePasswordRequest>
```

2782 The provider returns an `<expirePasswordResponse>` element. The "status" attribute of the  
2783 `<expirePasswordResponse>` element indicates that the provider successfully expired the  
2784 password.

```
<expirePasswordResponse requestID="134" status="success"/>
```

### 2785 3.6.5.3 *resetPassword*

2786 The `resetPassword` operation enables a requestor to *change (to an unspecified value)* the  
2787 password for an object and to obtain that newly generated password value.

2788 The subset of the Password Capability XSD that is most relevant to the `resetPassword` operation  
2789 follows.

```
<complexType name="ResetPasswordRequestType">  
  <complexContent>  
    <extension base="spml:RequestType">  
      <sequence>
```

```

        <element name="psoID" type="spml:PSOIdentifierType"/>
    </sequence>
</extension>
</complexContent>
</complexType>

<complexType name="ResetPasswordResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <element name="password" type="string" minOccurs="0"/>
        </sequence>
    </extension>
</complexContent>
</complexType>

    <element name="resetPasswordRequest"
type="pass:ResetPasswordRequestType"/>
    <element name="resetPasswordResponse"
type="pass:ResetPasswordResponseType"/>

```

#### 2790 [3.6.5.3.1 resetPasswordRequest \(normative\)](#)

2791 A requestor **MUST** send a `<resetPasswordRequest>` to a provider in order to (ask the provider  
2792 to) change the password that is associated an existing object and to (ask the provider to) return to  
2793 the requestor the new password value.

2794 **Execution.** A `<resetPasswordRequest>` **MAY** specify "executionMode".  
2795 See the section titled "[Determining execution mode](#)".

2796 **psoID.** A `<resetPasswordRequest>` **MUST** contain exactly one `<psoID>` element. The  
2797 `<psoID>` **MUST** identify an object that exists on a target (that is supported by the provider).  
2798 See the section titled "[PSO Identifier \(normative\)](#)".

#### 2799 [3.6.5.3.2 resetPasswordResponse \(normative\)](#)

2800 A provider that receives a `<resetPasswordRequest>` from a requestor that the provider trusts  
2801 **MUST** examine the content of the `<resetPasswordRequest>`. If the request is valid and if the  
2802 specified object exists, then the provider **MUST** change the password that is associated with the  
2803 object that is specified by the `<psoID>` and must return to the requestor the new password value.

2804 **Execution.** If an `<resetPasswordRequest>` does not specify "executionMode", the provider  
2805 **MUST** choose a type of execution for the requested operation.  
2806 See the section titled "[Determining execution mode](#)".

2807 **Response.** The provider must return to the requestor a `<resetPasswordResponse>`. The  
2808 `<resetPasswordResponse>` must have a "status" attribute that indicates whether the provider  
2809 successfully changed the password that is associated with the specified object and successfully  
2810 returned to the requestor the new password value. See the section titled "[Status \(normative\)](#)".

2811 If the provider knows that the provider will not be able to return to the requestor the new password  
2812 value, then the provider **MUST NOT** change the password that is associated with the specified  
2813 object. (To do so would create a state that requires manual administrator intervention, and this  
2814 defeats the purpose of the resetPassword operation.)

2815 **password.** The <resetPasswordResponse> MAY contain a <password> element. If the  
2816 <resetPasswordResponse> contains a <password> element, the <password> element MUST  
2817 contain the newly changed password value that is associated with the specified object.

2818 **Error.** If the provider cannot change the password that is associated with the specified object, or if  
2819 the provider cannot return the new password attribute value to the requestor, then the  
2820 <resetPasswordResponse> MUST specify an "error" that characterizes the failure.  
2821 See the general section titled "Error (normative)".

2822 In addition, a <resetPasswordResponse> MUST specify an error if any of the following is true:

- 2823 • The <resetPasswordRequest> contains a <psoID> for an object that does not exist.
- 2824 • The target system or application will not allow the provider to return to the requestor the new  
2825 password value. (If the provider knows this to be the case, then the provider MUST NOT  
2826 change the password that is associated with the specified object. See above.)
- 2827 • The target system or application *returns an error (or throws an exception)* when the provider  
2828 tries to change the password that is associated with the specified object or (when the provider)  
2829 tries to obtain the new password value.

### 2830 **3.6.5.3 resetPassword Examples (non-normative)**

2831 In the following example, a requestor asks a provider to reset the password for a Person object.

```
<resetPasswordRequest requestID="135">  
  <psoID ID="2244" targetID="target2"/>  
</resetPasswordRequest>
```

2832 The provider returns an <resetPasswordResponse> element. The "status" attribute of the  
2833 <resetPasswordResponse> indicates that the provider successfully reset the password.

```
<resetPasswordResponse requestID="135" status="success">  
  <password>gener8ed</password>  
</resetPasswordResponse>
```

### 2834 **3.6.5.4 validatePassword**

2835 The validatePassword operation enables a requestor to *determine whether a specified value would*  
2836 *be valid* as the password for a specified object.

2837 The subset of the Password Capability XSD that is most relevant to the validatePassword operation  
2838 follows.

```
<complexType name="ValidatePasswordRequestType">  
  <complexContent>  
    <extension base="spml:RequestType">  
      <sequence>  
        <element name="psoID" type="spml:PSOIdentifierType"/>  
        <element name="password" type="xsd:string"/>  
      </sequence>  
    </extension>  
  </complexContent>  
</complexType>  
  
<complexType name="ValidatePasswordResponseType">  
  <complexContent>
```

```

    <extension base="spml:ResponseType">
      <attribute name="valid" type="boolean" use="optional"/>
    </extension>
  </complexContent>
</complexType>

  <element name="validatePasswordRequest"
type="pass:ValidatePasswordRequestType"/>
  <element name="validatePasswordResponse"
type="pass:ValidatePasswordResponseType"/>

```

#### 2839 [3.6.5.4.1](#) *validatePasswordRequest (normative)*

2840 A requestor **MUST** send a `<validatePasswordRequest>` to a provider in order to (ask the  
2841 provider to) test whether a specified value would be valid as the password that is associated with  
2842 an existing object.

2843 **Execution.** A `<validatePasswordRequest>` **MAY** specify "executionMode".  
2844 See the section titled "[Determining execution mode](#)".

2845 **psoid.** A `<validatePasswordRequest>` **MUST** contain exactly one `<psoid>` element. The  
2846 `<psoid>` **MUST** identify an object that exists on a target (that is supported by the provider).  
2847 See the section titled "[PSO Identifier \(normative\)](#)".

2848 **password.** A `<validatePasswordRequest>` **MUST** contain exactly one `<password>` element.  
2849 The `<password>` element **MUST** contain a string value.

#### 2850 [3.6.5.4.2](#) *validatePasswordResponse (normative)*

2851 A provider that receives a `<validatePasswordRequest>` from a requestor that the provider  
2852 trusts **MUST** examine the content of the `<validatePasswordRequest>`. If the request is valid  
2853 and if the specified object exists, then the provider **MUST** test whether the specified value would be  
2854 valid as the password that is associated with the object that the `<psoid>` identifies.

2855 **Execution.** If an `<validatePasswordRequest>` does not specify "executionMode", the  
2856 provider **MUST** choose a type of execution for the requested operation.  
2857 See the section titled "[Determining execution mode](#)".

2858 **Response.** The provider must return to the requestor a `<validatePasswordResponse>`. The  
2859 `<validatePasswordResponse>` **MUST** have a "status" attribute that indicates whether the  
2860 provider successfully tested whether the supplied value would be valid as the password that is  
2861 associated with the specified object. See the section titled "[Status \(normative\)](#)".

2862 **valid.** The `<validatePasswordResponse>` **MUST** have a "valid" attribute that indicates  
2863 whether the `<password>` (content that was specified in the `<validatePasswordRequest>`)  
2864 would be valid as the password that is associated with the specified object.

2865 **Error.** If the provider cannot determine whether the specified value would be valid as the password  
2866 that is associated with the specified object, then the `<validatePasswordResponse>` **MUST**  
2867 specify an "error" value that characterizes the failure.  
2868 See the general section titled "[Error \(normative\)](#)".

2869 In addition, a `<validatePasswordResponse>` **MUST** specify an appropriate value of "error" if  
2870 any of the following is true:

- 2871 • The <validatePasswordRequest> contains a <psoid> for an object that does not exist.
- 2872 • The target system or application *returns an error (or throws an exception)* when the provider
- 2873 tries to determine whether the specified value would be valid as the password that is
- 2874 associated with the specified object.

#### 2875 [3.6.5.4.3 validatePassword Examples \(non-normative\)](#)

2876 In the following example, a requestor asks a provider to validate a value as a password for a

2877 Person object.

```
<validatePasswordRequest requestID="136">
  <psoid ID="2244" targetID="target2"/>
  <password>y0baby</password>
</validatePasswordRequest>
```

2878 The provider returns an <validatePasswordResponse> element. The "status" attribute of

2879 the <validatePasswordResponse> indicates that the provider successfully tested whether the

2880 <password> value specified in the request would be valid as the password that is associated with

2881 the specified object. The <validatePasswordResponse> specifies "valid='true'", which

2882 indicates that the specified value *would be valid* as the password that is associated with the

2883 specified object.

```
<validatePasswordResponse requestID="136" status="success" valid="true"/>
```

2884

### 2885 **3.6.6 Reference Capability**

2886 The Reference Capability is defined in a schema that is associated with the following XML  
2887 namespace: urn:oasis:names:tc:SPML:2:0:reference. This document includes the  
2888 Reference Capability XSD as Appendix F.

```
<complexType name="ReferenceType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="toPsoID" type="spml:PSOIdentifierType"
minOccurs="0"/>
        <element name="referenceData" type="spml:ExtensibleType"
minOccurs="0"/>
      </sequence>
      <attribute name="typeOfReference" type="string"
use="required"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="ReferenceDefinitionType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="schemaEntity"
type="spml:SchemaEntityRefType"/>
        <element name="canReferTo" type="spml:SchemaEntityRefType"
minOccurs="0" maxOccurs="unbounded"/>
        <element name="referenceDataType"
type="spml:SchemaEntityRefType" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="typeOfReference" type="string" use="required"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="HasReferenceType">
  <complexContent>
    <extension base="spml:QueryClauseType">
      <sequence>
        <element name="toPsoID" type="spml:PSOIdentifierType"
minOccurs="0" />
        <element name="referenceData" type="spml:ExtensibleType"
minOccurs="0" />
      </sequence>
      <attribute name="typeOfReference" type="string"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

<element name="hasReference" type="spmlref:HasReferenceType"/>
```

```
<element name="reference" type="spmlref:ReferenceType"/>
<element name="referenceDefinition"
type="spmlref:ReferenceDefinitionType"/>
```

2889 The Reference Capability defines no operation. Instead, the Reference Capability allows a provider  
2890 to declare, as part of each target, which types of objects support *references* to which other types of  
2891 objects. The XML representations of *references flow through the core operations as capability-*  
2892 *specific data*.

2893 • In order to *create an object with references*, a requestor specifies capability-specific data to the  
2894 add operation.

2895 • In order to *add, remove or replace references* to an object, a requestor specifies capability-  
2896 specific data to the modify operation.

2897 • In order to *obtain references* for an object, a requestor examines capability-specific data  
2898 returned as output by the add, lookup and search operations.

2899 **Motivation.** Defining a standard capability for references is important for several reasons.

2900 • Managing references to other objects can be an important part of managing objects.

2901 • Object references to other objects present a *scalability* problem.

2902 • Object references to other objects present an *integrity* problem.

2903 Provisioning systems must often list, create, and delete connections between objects  
2904 in order to manage the objects themselves. In some cases, a provisioning system  
2905 must manage data that is part a specific connection (e.g., in order to specify  
2906 the expiration of a user's membership in a group) – see the topic named “Reference Data” below.  
2907 Because connections to other objects can be very important, it is important to be able to represent  
2908 such connections *generically* (rather than as something specific to each target schema).

2909 The reference capability enables a requestor to manage an object's references independent of the  
2910 object's schema. This is particularly important in the cases where a provider allows references to  
2911 span targets. For example, a provisioning system must often maintain knowledge about which  
2912 people own which accounts. In such cases, an `Account` object (that is contained by one target)  
2913 may refer to a `Person` object (that is contained by another target) as its owner.

2914 Scale is another significant aspect of references. The *number of connections* between objects may  
2915 be an order of magnitude greater than the number of objects themselves. Unconditionally including  
2916 reference information in the XML representation of each object could greatly increase the size of  
2917 each object's XML representation. Imagine, for example, that each `Account` may refer to multiple  
2918 `Groups` (or that a `Group` might refer to each of its members).

2919 Defining reference as an optional capability (and allowing references to be omitted from each  
2920 object's schema) does two things. First, this allows a requestor to exclude an object's references  
2921 from the XML representation of each object (since a requestor can control which capability-specific  
2922 data are included). Second, this allows providers to manage references separately from schema-  
2923 defined attributes (which may help a provider cope with the scale of connections).

2924 The ability to manage references separately from schema-defined data may also help providers to  
2925 maintain the integrity of references. In the systems and applications that underlie many  
2926 provisioning target, deleting an object A may not delete another object B's reference to object A.  
2927 Allowing a provider to manage references separately allows the provider to control such behavior  
2928 (and perhaps even to prevent the deletion of object A when another object B still refers to object A).

### 2929 **3.6.6.1 Reference Definitions**

2930 **Reference Definitions.** A provider declares each type of reference that a particular target supports  
2931 (or declares each type of reference *that a particular supported schema entity* on a target supports)  
2932 as an instance of {ReferenceDefinitionType}.

2933 A provider's <listTargetsResponse> contains a list of targets that the provider exposes for  
2934 provisioning operations. Part of each target declaration is the set of capabilities that the target  
2935 supports. Each capability refers (by means of its "namespaceURI" attribute) to a specific  
2936 capability. Any <capability> element that refers to the Reference Capability may contain (as  
2937 open content) any number of <referenceDefinition> elements.

2938 Each reference definition names a specific type of reference and also specifies the following:

- 2939 • *which schema entity* (on the <target> that contains the <capability> that contains the  
2940 <referenceDefinition>) *can refer...*
- 2941 • *...to which schema entity* or schema entities (on which targets).

2942 For normative specifics, see the topic named "Reference Capability content" within the section titled  
2943 "[listTargetsResponse \(normative\)](#)".

2944 **Overlap.** Any number of reference definitions may declare different "from- and to-" entity pairs for  
2945 the same type of reference. For example, a reference definition may declare that an Account may  
2946 refer to a Person as its "owner". Another reference definition may declare that an  
2947 OrganizationalUnit may refer to a Person as its "owner". SPMLv2 specifies the mechanism-  
2948 *-but does not define the semantics--*of reference.

2949 **Direction.** Each reference definition specifies the *direction* of reference. A reference is always  
2950 from an object (that is an instance of the schema entity that <schemaEntity> specifies) to  
2951 another object (that is an instance of a schema entity that <canReferTo> specifies).

2952 **No Inverse.** A standard SPMLv2 reference definition specifies nothing about an inverse  
2953 relationship. For example, a reference definition that says an Account may refer to a Person as  
2954 its "owner" does NOT imply that a Person may refer to Account.

2955 Nothing prevents a provider from declaring (by means of a reference definition) that Person may  
2956 refer to Account in a type of reference called "owns", but nothing (at the level of this specification)  
2957 associates these two types of references to say that "owns" is the inverse of "owner".

2958 **No Cardinality.** A reference definition specifies no restrictions on the number of objects to which an  
2959 object may refer (by means of that defined type of reference). Thus, for example, an Account may  
2960 refer to multiple instances of Person as its "owner". This may be logically incorrect, or this may  
2961 not be the desired behavior, but SPMLv2 does not require a provider to support restrictions on the  
2962 cardinality of a particular type of reference.

2963 In general, a requestor must assume that each defined type of reference is optional and many-to-  
2964 many. This is particularly relevant when a requestor wishes to modify references. A requestor  
2965 SHOULD NOT assume that a reference that the requestor wishes to modify is the object's only  
2966 reference of that type. A requestor also SHOULD NOT assume that a reference from one object to  
2967 another object that the requestor wishes to modify is the *only* reference between the two objects.  
2968 The only restriction that SPMLv2 imposes is that an object A may have no more than one reference  
2969 of the same type to another object B. See the topic named "No duplicates" in the section titled  
2970 "[References](#)".

2971 **ReferenceDataType.** A reference definition may be *complex*, which means that an instance of that  
2972 type of reference may have reference data associated with it.  
2973 See the section titled "[Complex References](#)" below.

2974 The definition of a type of reference that is complex must contain a `<referenceDataType>` for  
2975 each possible structure of reference data. Each `<referenceDataType>` element refers to a  
2976 specific entity in a target schema. A `<referenceData>` element (within any instance of that type  
2977 of reference) may contain one element of any of these types (to which a `<referenceDataType>`  
2978 refers).

2979 A reference definition that contains no `<referenceDataType>` sub-element indicates that the  
2980 type of reference it defines *does not support reference data*.

2981 For a normative description, see the topic named "ReferenceDefinition referenceDataType" within  
2982 the section titled "[listTargetsResponse \(normative\)](#)".

### 2983 **3.6.6.2 References**

2984 **Must contain toPsoID.** Any `<reference>` MUST specify its "toObject". That is, any instance of  
2985 `{ReferenceType}` MUST contain a valid `<toPsoID>`. The only exception is a `<reference>`  
2986 that is used as a wildcard within a `<modification>` that specifies  
2987 "modificationMode='delete' ". In this case (and only in this case), the `<reference>` MUST  
2988 specify a valid "typeOfReference" but (the `<reference>`) MAY omit `<toPsoID>`.  
2989 See the section titled "[Reference CapabilityData Processing \(normative\)](#)".

2990 **No duplicates.** Within the set of references that is associated with an object, at most one  
2991 `<reference>` of a specific "typeOfReference" may refer to a particular object. That is, an  
2992 instance of `{CapabilityDataType}` MUST NOT contain two (and MUST NOT contain more than  
2993 two) instances of `<reference>` that specify the same value of "typeOfReference" and that  
2994 contain `<toPsoID>` elements that identify the same object. See the section titled "[Reference  
2995 CapabilityData in a Request \(normative\)](#)".

2996 **Reference Data.** SPMLv2 allows each reference (i.e., each instance of `{ReferenceType}`) to  
2997 contain additional reference data. Most references between objects require no additional data, but  
2998 allowing references to contain additional data supports cases in which a reference from one object  
2999 to another may carry additional information "on the arrow" of the relationship. For example, a  
3000 RACF user's membership in a particular RACF group carries with it the additional information of  
3001 whether that user has the ADMINISTRATOR or SPECIAL privilege within that group. Several other  
3002 forms of group membership carry with them additional information about the member's expiration.  
3003 See the section titled "[Complex References](#)" below.

3004 **Search.** A requestor can *search for objects based on reference values* using the  
3005 `<hasReference>` query clause. The `{HasReferenceType}` extends `{QueryClauseType}`,  
3006 which indicates that an instance of `{HasReferenceType}` can be used to select objects. A  
3007 `<hasReference>` clause matches an object if and only if the object has a reference that *matches*  
3008 *every specified component* (i.e., element or attribute) of the `<hasReference>` element.  
3009 See the section titled "[search Examples](#)".

### 3010 **3.6.6.3 Complex References**

3011 The vast majority of reference types are simple: that is, one object's reference to another object  
3012 carries no additional information. However certain types of references may support additional  
3013 information that is specific to a particular reference. For example, when a user is assigned to one  
3014 or more Entrust GetAccess Roles, each role assignment has a start date and an end date. We  
3015 describe a *reference that contains additional data* (where that data is specific to the reference) as a  
3016 "complex" reference.

3017 **Example: RACF Group Membership** is another example of a complex type of reference. Each  
3018 RACF group membership carries with it additional data about whether the user has the SPECIAL,  
3019 AUDITOR, or OPERATIONS privileges in that group.

- 3020 • Group-SPECIAL gives a group administrator control over all profiles within the group
- 3021 • Group-AUDITOR allows a user to monitor the use of the group's resources
- 3022 • Group-OPERATIONS allows a user to perform maintenance operations  
3023 on the group's resources

3024 For purposes of this example, let us represent these three group-specific privileges as attributes of  
3025 an XML type called "RacfGroupMembershipType". Suppose that the XML Schema for such a type  
3026 looks like the following:

```
<complexType name="RacfGroupMembershipType">
  <complexContent>
    <attribute name="special" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="auditor" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="operations" type="xsd:boolean" use="optional" default="false"/>
  </complexContent>
</complexType>

<element name="racfGroupMembership" type="RacfGroupMembershipType"/>
```

3027

3028 The following subsections describe several different ways to model RACF Group Membership. The  
3029 fictional `<xsd:schema>` is the same in all of the examples. In each subsection, however, the  
3030 provider's `<target>` definition varies with the approach.

### 3031 **3.6.6.3.1 Using Reference Data**

3032 The simplest way to model a complex reference such as RACF Group membership is to represent  
3033 the additional information as arbitrary *reference data*. The `<referenceData>` element within a  
3034 `<reference>` may contain any data.

3035 The following example shows how a provider's `listTargetsResponse` might reflect this approach.  
3036 The sample schema for the "RACF" target is very simple (for the sake of brevity). The provider  
3037 defines a type of reference called "memberOfGroup". Within a `<reference>` of this type, the  
3038 `<referenceData>` element must contain exactly one `<racfGroupMembership>` element (and  
3039 should contain nothing else).

```
<listTargetsResponse status="success">
  <target targetID="RacfGroupMembership-ReferenceData">
    <schema>
      <xsd:schema targetNamespace="urn:example:schema:RACF"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:spml="urn:oasis:names:tc:SPML:2:0" elementFormDefault="qualified">
        <complexType name="RacfUserProfileType">
          <attribute name="userid" type="string" use="required"/>
        </complexType>
        <complexType name="RacfGroupProfileType">
          <attribute name="groupName" type="string" use="required"/>
        </complexType>
        <complexType name="RacfGroupMembershipType">
```

```

    <attribute name="special" type="boolean" use="optional" default="false"/>
    <attribute name="auditor" type="boolean" use="optional" default="false"/>
    <attribute name="operations" type="boolean" use="optional" default="false"/>
  </complexType>
  <element name="racfUserProfile" type="RacfUserProfileType">
  <element name="racfGroupProfile" type="RacfGroupProfileType">
  <element name="racfGroupMembership" type="RacfGroupMembershipType">
</xsd:schema>
  <supportedSchemaEntity entityName="racfUserProfile"/>
  <supportedSchemaEntity entityName="racfGroupProfile"/>
</schema>
<capabilities>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:bulk"/>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:search"/>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:password">
    <appliesTo entityName="racfUserProfile"/>
  </capability>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:suspend">
    <appliesTo entityName="racfUserProfile"/>
    <appliesTo entityName="racfGroupProfile"/>
  </capability>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:reference">
    <appliesTo entityName="racfUserProfile"/>
    <referenceDefinition typeOfReference="memberOfGroup"/>
      <schemaEntity entityName="racfUserProfile"/>
      <canReferTo entityName="racfGroupProfile"/>
      <referenceDataType entityName="racfGroupMembership"/>
      <annotation>
        <documentation> ReferenceData for a "memberOfGroup" reference
        must contain exactly one racfGroupMembership element.</documentation>
      </annotation>
    </referenceDefinition>
  </capability>
</capabilities>
</target>
</listTargetsResponse>

```

3040 **Manipulating Reference Data.** The only way to manipulate the reference data associated with a  
 3041 complex reference is by using the [modify](#) operation that is part of the Core XSD. A requestor may  
 3042 add, replace or delete any capability-specific data that is associated with an object.

3043 **Capabilities Do Not Apply.** SPML specifies no way to apply a capability-specific operation to a  
 3044 reference. Thus, for example, one can neither suspend nor resume a reference. This is because a  
 3045 *reference is not a provisioning service object*. A reference is instead *capability-specific data that is*  
 3046 *associated with an object*.

3047 You can think of an object's references (or any set of capability-specific data that is associated with  
 3048 an object) as an "extra" attribute (or as an "extra" sub-element) of the object. The provider supports  
 3049 each "extra" (attribute or sub-element) data *independent of the schema* of the target that contains  
 3050 the object. The provider keeps all <capabilityData> separate from the regular schema-defined  
 3051 <data> within each <ps>.

### 3052 [3.6.6.3.2 Relationship Objects](#)

3053 The fact that capabilities cannot apply to references does not prevent a provider from offering this  
 3054 kind of rich function. There is an elegant way to represent a complex relationship that allows a

3055 requestor to operate directly on the relationship itself. A provider may model a complex relationship  
3056 between two objects as a third object that refers to each of the first two objects.

3057 This approach is analogous to a “linking record” in relational database design. In the “linking  
3058 record” approach, the designer “normalizes” reference relationships into a separate table. Each  
3059 row in a third table connects a row from one table to a row in another table. This approach allows  
3060 each relationship to carry additional information that is specific to that relationship. Data specific to  
3061 each reference are stored in the columns of the third table. Even when relationships do not need to  
3062 carry additional information, database designers often use this approach when two objects may be  
3063 connected by more than one instance of the same type of relationship, or when relationships are  
3064 frequently added or deleted and referential integrity must be maintained.

3065 Rather than have an object A refer to an object B directly, a third object C refers to both object A  
3066 and object B. Since object C represents the relationship itself, object C refers to object A as its  
3067 “fromObject” and object C refers to object B as its “toObject”.

3068 A provider that wants to treat each instance of a (specific type of) relationship as an object does so  
3069 by defining in the schema for a target a schema entity to contain the additional information (that is  
3070 specific to that type of relationship). The provider then declares two types of references that apply  
3071 to that schema entity: a “fromObject” type of reference and a “toObject” type of reference. The  
3072 provider may also declare that certain capabilities apply to that schema entity. This model allows a  
3073 requestor to operate conveniently on each instance of a complex relationship.

3074 For example, suppose that a provider models as a schema entity a type of relationship that has an  
3075 effective date and has an expiration date. As a convenience to requestors, the provider might  
3076 declare that this schema entity (that is, the “linking” entity) supports the Suspend Capability. The  
3077 ‘suspend’ and ‘resume’ operations could manipulate the expiration date and the effective date  
3078 *without the requestor having to understand the structure of that schema entity*. This convenience  
3079 could be very valuable where the attribute values or element content that are manipulated have  
3080 complex syntax, special semantics or implicit relationships with other elements or attributes.

3081 The following example shows how a provider’s listTargetsResponse might reflect this approach.  
3082 The sample schema for the “RACF” target is again simple (for the sake of brevity).

```
<listTargetsResponse status="success">
  <target targetID="RacfGroupMembership-IndependentRelationshipObject">
    <schema>
      <xsd:schema targetNamespace="urn:example:schema:RACF"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:spml="urn:oasis:names:tc:SPML:2:0" elementFormDefault="qualified">
        <complexType name="RacfUserProfileType">
          <attribute name="userid" type="string" use="required"/>
        </complexType>
        <complexType name="RacfGroupProfileType">
          <attribute name="groupName" type="string" use="required"/>
        </complexType>
        <complexType name="RacfGroupMembershipType">
          <attribute name="special" type="boolean" use="optional" default="false"/>
          <attribute name="auditor" type="boolean" use="optional" default="false"/>
          <attribute name="operations" type="boolean" use="optional" default="false"/>
        </complexType>
        <element name="racfUserProfile" type="RacfUserProfileType">
        <element name="racfGroupProfile" type="RacfGroupProfileType">
        <element name="racfGroupMembership" type="RacfGroupMembershipType">
      </xsd:schema>
    </supportedSchemaEntity entityName="racfUserProfile"/>
  </target>
</listTargetsResponse>
```

```

    <supportedSchemaEntity entityName="racfGroupProfile"/>
    <supportedSchemaEntity entityName="racfGroupMembership">
      <annotation>
        <documentation> Each instance of racfGroupMembership refers to one
        racfUserProfile and refers to one racfGroupProfile.</documentation>
      </annotation>
    </supportedSchemaEntity>
  </schema>
  <capabilities>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:bulk"/>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:search"/>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:password">
      <appliesTo entityName="RacfUserProfile"/>
    </capability>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:suspend">
      <appliesTo entityName="racfUserProfile"/>
      <appliesTo entityName="racfGroupProfile"/>
    </capability>
    <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:reference">
      <appliesTo entityName="racfGroupMembership"/>
      <referenceDefinition typeOfReference="fromUser"/>
        <schemaEntity entityName="racfGroupMembership"/>
        <canReferTo entityName="racfUserProfile"/>
      </referenceDefinition>
      <referenceDefinition typeOfReference="toGroup"/>
        <schemaEntity entityName="racfGroupMembership"/>
        <canReferTo entityName="racfGroupProfile"/>
      </referenceDefinition>
    </capability>
  </capabilities>
</target>
</listTargetsResponse>

```

3083 **Variations.** Naturally, many variations of this approach are possible. For example, an instance of  
 3084 RacfUserProfile could refer to an instance of RacfGroupMembership (rather than having an  
 3085 instance of RacfGroupMembership refer to both RacfUserProfile and an instance of  
 3086 RacfGroupProfile). However, such a variation would not permit an instance of RacfUserProfile to  
 3087 refer to more than one group (and could result in an orphaned relationship objects unless the  
 3088 provider carefully guards against this).

### 3089 3.6.6.3.3 *Bound Relationship Objects*

3090 One particularly robust variation of independent relationship objects is to *bind each relationship*  
 3091 *object beneath one of the objects it connects*. For example, one could bind each instance of  
 3092 RacfGroupMembership beneath the instance of RacfUserProfile that would otherwise be the  
 3093 "fromUser". That way, deleting an instance of RacfUserProfile also deletes all of its  
 3094 RacfGroupMemberships. This modeling approach makes clear that the relationship belongs with  
 3095 the "fromObject" and helps to prevent orphaned relationship objects.

3096 The next example illustrates bound relationship objects.

```

<listTargetsResponse status="success">
  <target targetID="RacfGroupMembership-BoundRelationshipObject">
    <schema>
      <xsd:schema targetNamespace="urn:example:schema:RACF"
        xmlns="http://www.w3.org/2001/XMLSchema"

```

```

xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:spml="urn:oasis:names:tc:SPML:2:0" elementFormDefault="qualified">
  <complexType name="RacfUserProfileType">
    <attribute name="userid" type="string" use="required"/>
  </complexType>
  <complexType name="RacfGroupProfileType">
    <attribute name="groupName" type="string" use="required"/>
  </complexType>
  <complexType name="RacfGroupMembershipType">
    <attribute name="special" type="boolean" use="optional" default="false"/>
    <attribute name="auditor" type="boolean" use="optional" default="false"/>
    <attribute name="operations" type="boolean" use="optional" default="false"/>
  </complexType>
  <element name="racfUserProfile" type="RacfUserProfileType">
  <element name="racfGroupProfile" type="RacfGroupProfileType">
  <element name="racfGroupMembership" type="RacfGroupMembershipType">
</xsd:schema>
  <supportedSchemaEntity entityName="racfUserProfile" isContainer="true"/>
    <annotation>
      <documentation> Any number of racfGroupMembership objects may be
bound beneath a racfUserProfile object.</documentation>
    </annotation>
  </supportedSchemaEntity>
</supportedSchemaEntity>
  <supportedSchemaEntity entityName="racfGroupProfile"/>
  <supportedSchemaEntity entityName="racfGroupMembership">
    <annotation>
      <documentation> Each racfGroupMembership is bound beneath a
racfUserProfile and refers to one racfGroupProfile.</documentation>
    </annotation>
  </supportedSchemaEntity>
</schema>
<capabilities>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:bulk"/>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:search"/>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:password">
    <appliesTo entityName="racfUserProfile"/>
  </capability>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:suspend">
    <appliesTo entityName="racfUserProfile"/>
    <appliesTo entityName="racfGroupProfile"/>
  </capability>
  <capability namespaceURI="urn:oasis:names:tc:SPML:2.0:reference">
    <appliesTo entityName="racfGroupMembership"/>
    <referenceDefinition typeOfReference="toGroup"/>
      <schemaEntity entityName="racfGroupMembership"/>
      <canReferTo entityName="racfGroupProfile"/>
    </referenceDefinition>
  </capability>
</capabilities>
</target>
</listTargetsResponse>

```

3097 **3.6.6.4 Reference CapabilityData in a Request (normative)**

3098 The general rules that govern an instance of {CapabilityDataType} in a request also apply to  
3099 an instance of {CapabilityDataType} that refers to the Reference Capability.  
3100 See the section titled "[CapabilityData in a Request \(normative\)](#)".

3101 **capabilityURI.** An instance of {CapabilityDataType}  
3102 that contains data that are specific to the Reference Capability MUST specify  
3103 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'".

3104 **mustUnderstand.** An instance of {CapabilityDataType} that refers to the Reference  
3105 Capability SHOULD specify "mustUnderstand='true'".

3106 **Capability defines structure.** An instance of {CapabilityDataType} that refers to the  
3107 Reference Capability MUST contain at least one <reference> element. An instance of  
3108 {CapabilityDataType} that refers to the Reference Capability SHOULD NOT contain any  
3109 element that is not a <reference> element.

3110 **No duplicates.** Within the set of references that is associated with an object, at most one  
3111 <reference> of a specific "typeOfReference" may refer to a specific object. That is, an  
3112 instance of {CapabilityDataType} MUST NOT contain two (and MUST NOT contain more than  
3113 two) instances of <reference> that specify the same value of "typeOfReference" and that  
3114 contain <toPsoID> elements that identify the same object.

3115 **Validate each reference.** Any <reference> that an instance of {CapabilityDataType}  
3116 contains must be an instance of {spmlref:ReferenceType}. In addition, a provider MUST  
3117 examine the following aspects of each <reference>:

- 3118 - The *"from" object*. (The object that contains--or that is intended to contain--the reference.)
- 3119 - The *"to" object*. (The object that the <toPsoID> of the reference identifies.)
- 3120 - The *"from" schema entity*. (The schema entity of which the "from" object is an instance.)
- 3121 - The *"to" schema entity* (The schema entity of which the "to" object is an instance.)
- 3122 - The *typeOfReference*
- 3123 - Any *referenceData*

3124 The standard aspects of SPML that specify supported schema entities and capabilities imply the  
3125 following:

- 3126 - The "to" object MUST exist (on a target that the provider exposes).
- 3127 - The target that contains the "from" object MUST support the "from" schema entity.
- 3128 - The target that contains the "to" object MUST support the "to" schema entity.
- 3129 - The target that contains the "from" object MUST support the Reference Capability.
- 3130 - The target that contains the "from" object MUST declare that
- 3131 the Reference Capability applies to the "from" schema entity.

3132 See the section titled "[listTargetsResponse \(normative\)](#)".

3133 **Check Reference Definition.** In addition, a provider must validate the "typeOfReference" that  
3134 each <reference> specifies (as well as the "from" schema entity and the "to" schema entity)  
3135 against the set of valid reference definitions..

3136 The <capability> that declares that the target (that contains the "from" object)  
3137 supports the Reference Capability for the "from" schema entity  
3138 MUST contain a <referenceDefinition> for which all of the following are true:

- 3139 - The <referenceDefinition> specifies the same "typeOfReference"  
3140 that the <reference> specifies

- 3141 - The <referenceDefinition> contains a <schemaEntity> element  
3142 that specifies the "from" schema entity
- 3143 - The <referenceDefinition> contains a <canReferTo> element  
3144 that specifies the "to" schema entity.

3145 See the section titled "[Reference Definitions](#)" above.

### 3146 **3.6.6.5 Reference CapabilityData Processing (normative)**

3147 The general rules that govern processing of an instance of {CapabilityDataType} in a request  
3148 also apply to an instance of {CapabilityDataType} that refers to the Reference Capability. See  
3149 the section titled "[CapabilityData Processing \(normative\)](#)".

3150 **capabilityURI.** An instance of {CapabilityDataType} that refers to the Reference Capability  
3151 MUST specify "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference' ". The  
3152 target (that contains the object to be manipulated) MUST support the Reference Capability for the  
3153 schema entity of which the object to be manipulated is an instance.

3154 **mustUnderstand.** An instance of {CapabilityDataType} that refers to the Reference  
3155 Capability SHOULD specify "mustUnderstand='true' ". A provider that supports the Reference  
3156 Capability MUST handle the content as this capability specifies (regardless of the value of  
3157 "mustUnderstand"). See the topic named "mustUnderstand" within the section titled  
3158 "[CapabilityData Processing \(normative\)](#)".

3159 **Open content.** An instance of {CapabilityDataType} that refers to the Reference Capability  
3160 MUST contain at least one <reference>. An instance of {CapabilityDataType} that refers to  
3161 the Reference Capability SHOULD NOT contain any element that is not a <reference>.

3162 **Validation.** A provider MUST examine the content of any instance of {CapabilityDataType}  
3163 that refers to the Reference Capability (regardless of the type of request that contains the instance  
3164 of {CapabilityDataType}) and ensure that it contains only valid instances of <reference>.  
3165 See the section titled "[Reference CapabilityData in a Request \(normative\)](#)".

3166 If the content (of the instance of {CapabilityDataType} that refers to the Reference Capability)  
3167 is not valid, then the provider's response MUST specify "status='failure' ".  
3168 See the section titled "[Request CapabilityData Errors \(normative\)](#)".

3169 **Process individual references.** In addition to the validation described above, the content of an  
3170 instance of {CapabilityDataType} that refers to the Reference Capability is not treated as  
3171 opaque, but instead as a set of individual references. The handling of each <reference>  
3172 depends on the type of element that contains the instance of {CapabilityDataType}).

- 3173 • If an <addRequest> contains an instance of {CapabilityDataType} that refers to the  
3174 Reference Capability, then the provider MUST associate the instance of  
3175 {CapabilityDataType} (and each <reference> that it contains)  
3176 with the newly created object.
  - 3177 • If a <modification> contains an instance of {CapabilityDataType} that refers to the  
3178 Reference Capability, then the handling of each <reference> (that the instance of  
3179 {CapabilityDataType} contains) depends on the "modificationMode" of that  
3180 <modification> and also depends on whether a matching <reference> is already  
3181 associated with the object to be modified.
- 3182 - If the <modification> specifies "modificationMode='add' ",  
3183 then the provider MUST *add each new reference* for which no matching <reference> is

3184 already associated with the object.  
3185 That is, the provider MUST associate with the object to be modified each <reference>  
3186 (that the instance of {CapabilityDataType} within the <modification> contains)  
3187 for which no <reference> that is already associated with the object  
3188 specifies the same value for "typeOfReference" (that the <reference> from the  
3189 <modification> specifies) and contains a <toPsoID> that identifies the same object  
3190 (that the <toPsoID> of the <reference> from the <modification> identifies).  
3191  
3192 The provider MUST *replace each matching reference* that is already associated with the  
3193 object with the <reference> from the <modification>.  
3194 That is, if a <reference> that is already associated with the object specifies the same  
3195 value for "typeOfReference" (that the <reference> from the <modification>  
3196 specifies) and if the <reference> that is already associated with the object contains a  
3197 <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from  
3198 the <modification> identifies), then the provider MUST *remove* the <reference> that  
3199 is already associated with the object and (the provider MUST) *add* the <reference> from  
3200 the <modification>.  
3201 This has the net effect of replacing any optional <referenceData> (as well as replacing  
3202 any open content) of the matching <reference>.

- 3203 - If the <modification> specifies "modificationMode='replace'",  
3204 then the provider MUST *add each new reference* for which no matching <reference> is  
3205 already associated with the object.  
3206 That is, the provider MUST associate with the object to be modified each <reference>  
3207 (that the instance of {CapabilityDataType} within the <modification> contains)  
3208 for which no <reference> that is already associated with the object  
3209 specifies the same value for "typeOfReference" (that the <reference> from the  
3210 <modification> specifies) and contains a <toPsoID> that identifies the same object  
3211 (that the <toPsoID> of the <reference> from the <modification> identifies).  
3212  
3213 The provider MUST *replace each matching reference* that is already associated with the  
3214 object with the <reference> from the <modification>.  
3215 That is, if a <reference> that is already associated with the object specifies the same  
3216 value for "typeOfReference" (that the <reference> from the <modification>  
3217 specifies) and if the <reference> that is already associated with the object contains a  
3218 <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from  
3219 the <modification> identifies), then the provider MUST *remove* the <reference> that  
3220 is already associated with the object and (the provider MUST) *add* the <reference> from  
3221 the <modification>.  
3222 This has the net effect of replacing any optional <referenceData> (as well as replacing  
3223 any open content) of the matching <reference>.
- 3224 - If the <modification> specifies "modificationMode='delete'",  
3225 then the provider MUST *remove each matching reference*.  
3226 A reference that omits <toPsoID> is *treated as a wildcard*.  
3227  
3228 If the <reference> from the <modification> contains a <toPsoID> element,  
3229 then the provider MUST remove (from the set of references that are associated with the  
3230 object) any <reference> that specifies the same value for "typeOfReference" (that  
3231 the <reference> from the <modification> specifies) and that contains a <toPsoID>  
3232 that identifies the same object (that the <toPsoID> of the <reference> from the  
3233 <modification> identifies).  
3234

3235 If the <reference> from the <modification> contains no <toPsoID> element,  
3236 then the provider MUST remove (from the set of references that are associated with the  
3237 object) any <reference> that specifies the same value for "typeOfReference" (that  
3238 the <reference> from the <modification> specifies).

3239  
3240 If no instance of <reference> that is associated with the object to be modified matches  
3241 the <reference> from the <modification>, then the provider MUST do nothing for that  
3242 <reference>. In this case, the provider's response MUST NOT specify  
3243 "status='failure'" unless there is some other reason to do so.

#### 3244 **3.6.6.6 Reference CapabilityData Errors (normative)**

3245 The general rules that govern errors related to an instance of {CapabilityDataType} in a  
3246 request also apply to an instance of {CapabilityDataType} that refers to the Reference  
3247 Capability. See the section titled "[CapabilityData Errors \(normative\)](#)".

3248 A provider's response to a request that contains an instance of {CapabilityDataType} that  
3249 refers to the Reference Capability (e.g., a <capabilityData> element that specifies  
3250 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'")  
3251 MUST specify an error if any of the following is true:

- 3252 • The instance of {CapabilityDataType} that refers to the Reference Capability  
3253 does not contain at least one <reference> element.
- 3254 • The instance of {CapabilityDataType} that refers to the Reference Capability  
3255 contains a <reference> element that is not a valid instance of {ReferenceType}.
- 3256 • The instance of {CapabilityDataType} that refers to the Reference Capability  
3257 contains a <reference> element for which no instance of Reference Definition declares that  
3258 (an instance of) the "from" schema entity may refer to (an instance of) the "to" schema entity  
3259 with the typeOfReference that the <reference> specifies.  
3260 See the section titled "[Reference Definitions](#)" above.

3261 A provider's response to a request that contains an instance of {CapabilityDataType} that  
3262 refers to the Reference Capability MAY specify an error if any of the following is true:

- 3263 • The instance of {CapabilityDataType} that refers to the Reference Capability  
3264 contains data other than valid <reference> elements.

3265 A provider's response (to a request that contains an instance of {CapabilityDataType} that  
3266 refers to the Reference Capability) SHOULD contain an <errorMessage> for each <reference>  
3267 element that was not valid.

#### 3268 **3.6.6.7 Reference CapabilityData in a Response (normative)**

3269 The general rules that govern an instance of {CapabilityDataType} in a response also apply to  
3270 an instance of {CapabilityDataType} that refers to the Reference Capability.  
3271 See the section titled "[CapabilityData in a Response \(normative\)](#)".

3272 The specific rules that apply to an instance of {CapabilityDataType} that refers to the  
3273 Reference Capability *in a response* also apply to an instance of {CapabilityDataType} (that  
3274 refers to the Reference Capability) *in a request*. (However, if the provider has applied the rules in  
3275 processing each request, the provider should not need to apply those rules again in formatting a  
3276 response.) See the section titled "[Reference CapabilityData in a Request \(normative\)](#)".

### 3277 3.6.7 Search Capability

3278 The Search Capability is defined in a schema associated with the following XML namespace:  
3279 `urn:oasis:names:tc:SPML:2:0:search`. This document includes the Search Capability XSD  
3280 as Appendix G.

3281 The Search Capability defines three operations: `search`, `iterate` and `closeterator`. The search and  
3282 iterate operations together allow a requestor to obtain *in a scalable manner* the XML representation  
3283 of every object that matches specified selection criteria. The search operation returns in its  
3284 response a first set of matching objects. Each subsequent iterate operation returns more matching  
3285 objects. The closeterator operation allows a requestor to tell a provider that it does not intend to  
3286 finish iterating a search result (and that the provider may therefore release the associated  
3287 resources).

3288 A provider that supports the search and iterate operations for a target SHOULD declare that the  
3289 target supports the Search Capability. A provider that does not support both search and iterate  
3290 MUST NOT declare that the target supports the Search Capability.

3291 **Resource considerations.** A provider must limit the size and duration of its search results (or that  
3292 provider will exhaust available resources). A provider must decide:

- 3293 • How large of a search result the provider will *select* on behalf of a requestor.
- 3294 • How large of a search result the provider will *queue* on behalf of a requestor  
3295 (so that the requestor may iterate the search results).
- 3296 • For *how long a time* the provider will queue a search result on behalf of a requestor.

3297 These decisions may be governed by the provider's implementation, by its configuration, or by  
3298 runtime computation.

3299 A provider that wishes to *never to queue search results* may return every matching object (up to the  
3300 provider's limit and up to any limit specified by the requestor) in the search response. Such a  
3301 provider would never return an iterator, and would not need to support the iterate operation. The  
3302 disadvantage is that, without an iterate operation, a provider's search capability either is limited to  
3303 small results or produces large search responses.

3304 A provider that wishes to support the iterate operation must store (or somehow queue) the objects  
3305 selected by a search operation until the requestor has a chance to iterate those results. (That is, a  
3306 provider must somehow queue the objects that matched the criteria of a search operation and that  
3307 were not returned in the search response.)

3308 If all goes well, the requestor will continue to iterate the search result until the provider has sent all  
3309 of the objects to the requestor. The requestor may also use the closeterator operation to tell the  
3310 provider that the requestor is no longer interested in the search result. In either case, the provider  
3311 may free any resource that is still associated with the search result. However, it is possible that the  
3312 requestor may not iterate the search result in a timely manner—or that the requestor may *never*  
3313 iterate the search result completely. Such a requestor may also neglect to close the iterator.

3314 A provider cannot queue search results indefinitely. The provider must eventually release the  
3315 resources that are associated with a search result. (Put differently, any iterator that a provider  
3316 returns to a requestor must eventually expire.) Otherwise, the provider may run out of resources.

3317 Providers should carefully manage the resources associated with search results. For example:

- 3318 • A provider may define a *timeout interval* that specifies the maximum time between iterate  
3319 requests. If a requestor does not request an iterate operation within this interval, the provider

- 3320 will release the resources associated with the search result. This invalidates any iterator that  
3321 represents this search result.
- 3322 • A provider may also define an overall *result lifetime* that specifies the maximum length of time  
3323 to retain a search result. After this amount of time has passed, the provider will release the  
3324 search result.
- 3325 • A provider may also wish to enforce an *overall limit* on the resources available to queue search  
3326 results, and may wish to adjust its behavior (or even to refuse search requests) accordingly.
- 3327 • To prevent denial of service attacks, the provider should not allocate any resource on behalf of  
3328 a requestor until that requestor is properly authenticated.  
3329 See the section titled "[Security and Privacy Considerations](#)".

### 3330 **3.6.7.1 search**

3331 The search operation obtains every object that matches a specified query.

3332 The subset of the Search Capability XSD that is most relevant to the search operation follows.

```

<simpleType name="ScopeType">
  <restriction base="string">
    <enumeration value="pso"/>
    <enumeration value="oneLevel"/>
    <enumeration value="subTree"/>
  </restriction>
</simpleType>

<complexType name="SearchQueryType">
  <complexContent>
    <extension base="spml:QueryClauseType">
      <sequence>
        <annotation>
          <documentation>Open content is one or more instances of
QueryClauseType (including SelectionType) or
LogicalOperator.</documentation>
        </annotation>
        <element name="basePsoID" type="spml:PSOIdentifierType"/>
      </sequence>
      <attribute name="targetID" type="string" use="optional"/>
      <attribute name="scope" type="spmlsearch:ScopeType"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="ResultsIteratorType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="ID" type="xsd:ID"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="SearchRequestType">

```

```

    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="query" type="spmlsearch:SearchQueryType"
minOccurs="0"/>
          <element name="includeDataForCapability" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="returnData" type="spml:ReturnDataType"
use="optional" default="everything"/>
        <attribute name="maxSelect" type="xsd:int" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="SearchResponseType">
    <complexContent>
      <extension base="spml:ResponseType">
        <sequence>
          <element name="pso" type="spml:PSOType" minOccurs="0"
maxOccurs="unbounded"/>
          <element name="iterator"
type="spmlsearch:ResultsIteratorType" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <element name="query" type="spmlsearch:SearchQueryType"/>
  <element name="searchRequest" type="spmlsearch:SearchRequestType"/>
  <element name="searchResponse" type="spmlsearch:SearchResponseType"/>

```

3333 The <query> is the same type of element that is specified as part of a <bulkModifyRequest> or  
3334 a <bulkDeleteRequest>. See the section titled "[SearchQueryType](#)".

3335 If the search operation is successful *but selects no matching object*, the <searchResponse> will  
3336 not contain a <pso>.

3337 If the search operation is successful *and selects at least one matching object*, the  
3338 <searchResponse> will contain any number of <pso> elements, each of which represents a  
3339 matching object. If the search operation selects more matching objects than the  
3340 <searchResponse> contains, the <searchResponse> will also contain an <iterator> that the  
3341 requestor can use to retrieve more matching objects. (See the iterate operation below.)

3342 If a search operation would select more objects than the provider can queue for subsequent  
3343 iteration by the requestor, the provider's <searchResponse> will specify  
3344 "error='resultSetTooLarge'".

3345 **Search is not batchable.** For reasons of scale, neither a search request nor an iterate request  
3346 should be nested in a [batch](#) request. When a search query matches more objects than the provider  
3347 can place directly in the response, the provider must temporarily store the remaining objects.  
3348 Storing the remaining objects allows the requestor to iterate the remaining objects, but also requires  
3349 the provider to commit resources.  
3350 See the topic named "Resource Considerations" earlier in this section.

3351 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
3352 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
3353 storing search results. Allowing a requestor to nest a search request within a batch request would  
3354 aggravate the resource problem, requiring a provider to store more information in larger chunks for  
3355 a longer amount of time.

### 3356 [3.6.7.1.1 searchRequest \(normative\)](#)

3357 A requestor **MUST** send a `<searchRequest>` to a provider in order to (ask the provider to) obtain  
3358 every object that matches specified selection criteria.

3359 **Execution.** A `<searchRequest>` **MAY** specify `"executionMode"`.  
3360 See the section titled "[Determining execution mode](#)".

3361 **query.** A `<query>` describes criteria that (the provider must use to) select objects on a target.  
3362 A `<searchRequest>` **MAY** contain at most one `<query>` element.

- 3363 • If the provider's `<listTargetsResponse>` contains only a single `<target>`,  
3364 then a `<searchRequest>` may omit the `<query>` element.
- 3365 • If the provider's `<listTargetsResponse>` contains more than one `<target>`,  
3366 then a `<searchRequest>` **MUST** contain exactly one `<query>` element  
3367 and that `<query>` must specify `"targetID"`.

3368 See the section titled "[SearchQueryType in a Request \(normative\)](#)".

3369 **ReturnData.** A `<searchRequest>` **MAY** have a `"returnData"` attribute that tells the provider  
3370 which types of data to include in each selected object.

- 3371 • A requestor that wants the provider to return *nothing* of the added object  
3372 **MUST** specify `"returnData='nothing' "`.
- 3373 • A requestor that wants the provider to return *only the identifier* of the added object  
3374 **MUST** specify `"returnData='identifier' "`.
- 3375 • A requestor that wants the provider to return the identifier of the added object  
3376 *plus the XML representation of the object (as defined in the schema of the target)*  
3377 **MUST** specify `"returnData='data' "`.
- 3378 • A requestor that wants the provider to return the identifier of the added object  
3379 *plus the XML representation of the object (as defined in the schema of the target)*  
3380 *plus any capability-specific data that is associated with the object*  
3381 **MAY** specify `"returnData='everything' "` or **MAY** omit the `"returnData"` attribute  
3382 (since `"returnData='everything' "` is the default).

3383 **maxSelect.** A `<searchRequest>` **MAY** have a `"maxSelect"` attribute. The value of the  
3384 `"maxSelect"` attribute specifies the maximum number of objects the provider should select.

3385 **IncludeDataForCapability.** A `<searchRequest>` **MAY** contain any number of  
3386 `<includeDataForCapability>` elements. Each `<includeDataForCapability>` element  
3387 specifies a capability for which the provider should return capability-specific data (unless the  
3388 `"returnData"` attribute specifies that the provider should return no capability-specific data at all).

- 3389 • A requestor that wants the provider to return (as part of each object) capability-specific data *for*  
3390 *only a certain set of capabilities* **MUST** enumerate that set of capabilities (by including an  
3391 `<includeDataForCapability>` element that specifies each such capability) in the  
3392 `<searchRequest>`.

3393 • A requestor that wants the provider to return (as part of each object) capability-specific data *for*  
3394 *all capabilities* MUST NOT include an <includeDataForCapability> element in the  
3395 <searchRequest>.

3396 • A requestor that wants the provider to return *no capability-specific data* MUST specify an  
3397 appropriate value for the “returnData” attribute.  
3398 See the topic named “ReturnData” immediately previous.

### 3399 3.6.7.1.2 *searchResponse (normative)*

3400 A provider that receives a <searchRequest> from a requestor that the provider trusts must  
3401 examine the content of the <searchRequest>. If the request is valid, the provider MUST return  
3402 (the XML that represents) every object that matches the specified <query> (if the provider can  
3403 possibly do so). However, the number of objects selected (for immediate return or for eventual  
3404 iteration) MUST NOT exceed any limit specified as “maxSelect” in the <searchRequest>.

3405 **Execution.** If an <searchRequest> does not specify “executionMode”, the provider MUST  
3406 choose a type of execution for the requested operation.  
3407 See the section titled “[Determining execution mode](#)”.

3408 A provider SHOULD execute a search operation synchronously if it is possible to do so. (The  
3409 reason for this is that the result of a search should reflect the current state of each matching object.  
3410 Other operations are more likely to intervene if a search operation is executed asynchronously.)

3411 **Response.** The provider MUST return to the requestor a <searchResponse>.

3412 **Status.** The <searchResponse> must contain a “status” attribute that indicates whether the  
3413 provider successfully selected every object that matched the specified query.  
3414 See the section titled “[Status \(normative\)](#)”.

3415 • If the provider successfully returned (the XML that represents) every object that matched the  
3416 specified <query> up to any limit specified by the value of the “maxSelect” attribute, then the  
3417 <searchResponse> MUST specify “status=’ success’”.

3418 • If the provider encountered an error in selecting any object that matched the specified <query>  
3419 or (if the provider encountered an error) in returning (the XML that represents) any of the  
3420 selected objects, then the <searchResponse> MUST specify “status=’ failure’”.

3421 **PSO.** The <searchResponse> MAY contain any number of <ps0> elements.

3422 • If the <searchResponse> specifies “status=’ success’” and *at least one object matched*  
3423 *the specified <query>*, then the <searchResponse> MUST contain at least one <ps0>  
3424 element that contains (the XML representation of) a matching object.

3425 • If the <searchResponse> specifies “status=’ success’” and *no object matched* the  
3426 *specified <query>*, then the <searchResponse> MUST NOT contain a <ps0> element.

3427 • If the <searchResponse> specifies “status=’ failure’”, then the <searchResponse>  
3428 MUST NOT contain a <ps0> element.

3429 **PSO and ReturnData.** Each <ps0> contains the subset of (the XML representation of) a requested  
3430 object that the “returnData” attribute of the <searchRequest> specified. By default, each  
3431 <ps0> contains the entire (XML representation of an) object.

- 3432 • A <ps> element MUST contain a <psID> element.  
 3433 The <psID> element MUST contain the identifier of the requested object.  
 3434 See the section titled “[PSO Identifier \(normative\)](#)”.
- 3435 • A <ps> element MAY contain a <data> element.
- 3436 - If the <searchRequest> specified “returnData=’ identifier’ ”,  
 3437 then the <ps> MUST NOT contain a <data> element.
- 3438 - Otherwise, if the <searchRequest> specified “returnData=’ data’ ”  
 3439 or (if the <searchRequest> specified) “returnData=’ everything’ ”  
 3440 or (if the <searchRequest>) omitted the “returnData” attribute  
 3441 then the <data> element MUST contain the XML representation of the object.  
 3442 This XML must be valid according to the schema of the target for the schema entity of  
 3443 which the newly created object is an instance.
- 3444 • A <ps> element MAY contain any number of <capabilityData> elements. Each  
 3445 <capabilityData> element contains a set of *capability-specific data* that is associated with  
 3446 the newly created object (for example, a *reference* to another object).
- 3447 - If the <searchRequest> specified “returnData=’ identifier’ ”  
 3448 or (if the <searchRequest> specified) “returnData=’ data’ ”  
 3449 then the <ps> MUST NOT contain a <capabilityData> element.
- 3450 - Otherwise, if the <searchRequest> specified “returnData=’ everything’ ”  
 3451 or (if the <searchRequest>) omitted the “returnData” attribute,  
 3452 then the <ps> MUST contain a <capabilityData> element for each set of capability-  
 3453 specific data that is associated with the requested object  
 3454 (and that is specific to a capability that the target supports for the schema entity of which  
 3455 the requested object is an instance).
- 3456 **PSO capabilityData and IncludeDataForCapability.** A <searchResponse> MUST include (as  
 3457 <capabilityData> sub-elements of each <ps>) any set of capability-specific data that is  
 3458 associated with a matching object and for which *all* of the following are true:
- 3459 • The <searchRequest> specifies “returnData=’ everything’ ” or (the  
 3460 <searchRequest>) omits the “returnData” attribute.
- 3461 • The schema for the target declares that the *target supports the capability* (for the schema entity  
 3462 of which each matching object is an instance).
- 3463 • The <searchRequest> contains an <includeDataForCapability> element that contains  
 3464 (as its string content) the URI of the capability to which the data are specific or the  
 3465 <searchRequest> contains no <includeDataForCapability> element.
- 3466 A <searchResponse> SHOULD NOT include (as a <capabilityData> sub-element of each  
 3467 <ps>) any set of capability-specific data for which any of the above is not true.
- 3468 **iterator.** A <searchResponse> MAY contain at most one <iterator> element.
- 3469 • If the <searchResponse> specifies “status=’ success’ ” and the search response *contains*  
 3470 *all of the objects* that matched the specified <query>, then the <searchResponse> MUST  
 3471 NOT contain an <iterator>.
- 3472 • If the <searchResponse> specifies “status=’ success’ ” and the search response *contains*  
 3473 *some but not all of the objects* that matched the specified <query>, then the  
 3474 <searchResponse> MUST contain exactly one <iterator>.

- 3475 • If the <searchResponse> specifies “status=' success' ” and *no object matched* the  
 3476 specified <query>, then the <searchResponse> MUST NOT contain an <iterator>.
- 3477 • If the <searchResponse> specifies “status=' failure' ”, then the <searchResponse>  
 3478 MUST NOT contain an <iterator>.

3479 **iterator ID.** An <iterator> MUST have an “ID” attribute.

3480 The value of the “ID” attribute uniquely identifies the <iterator> within the namespace of the  
 3481 provider. The “ID” attribute allows the provider to map each <iterator> token to the result set of  
 3482 the requestor’s <query> and (also allows the provider to map each <iterator> token) to any  
 3483 state that records the requestor’s position within that result set.

3484 The “ID” attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an  
 3485 <iterator>. An <iterator> is not a PSO.

3486 **Error.** If the <searchResponse> specifies “status=' failure' ”, then the <searchResponse>  
 3487 MUST have an “error” attribute that characterizes the failure.

3488 See the general section titled “[Error \(normative\)](#)”.

3489 The section titled “[SearchQueryType Errors \(normative\)](#)” describes errors specific to a request that  
 3490 contains a <query>. Also see the section titled “[SelectionType Errors \(normative\)](#)”.

3491 In addition, a <searchResponse> MUST specify an appropriate value of “error” if any of the  
 3492 following is true:

- 3493 • If the *number of objects that matched* the <query> that was specified in a <searchRequest>  
 3494 *exceeds any limit on the part of the provider* (but does not exceed any value of “maxSelect”  
 3495 that the requestor specified as part of the <query>). In this case, the provider’s  
 3496 <searchResponse> SHOULD specify “error=' resultSetTooLarge' ”.

### 3497 [3.6.7.1.3 search Examples \(non-normative\)](#)

3498 In the following example, a requestor asks a provider to search for every Person with an email  
 3499 address matching ‘joebob@example.com’.

```
<searchRequest requestID="137">
  <query scope="subTree" targetID="target2" >
    <select path="/Person/email="joebob@example.com"
namespaceURI="http://www.w3.org/TR/xpath20" />
  </query>
</searchRequest>
```

3500 The provider returns a <searchResponse>. The “status” attribute of the <searchResponse>  
 3501 indicates that the provider successfully executed the search operation.

```
<searchResponse requestID="137" status="success">
  <ps0>
    <data>
      <Person cn="joebob" firstName="joebob" lastName="Briggs" fullName="JoeBob
Briggs">
        <email>joebob@example.com</email>
      </Person>
    </data>
    <ps0ID ID="2244" targetID="target2"/>
  </ps0>
  <iterator ID="1826"/>
</searchResponse>
```

3502 In the following example, a requestor asks a provider to search for every account that is currently  
3503 owned by "joebob". The requestor uses the "returnData" attribute to specify that the provider  
3504 should return only the identifier for each matching object.

```
<searchRequest requestID="138" returnData="identifier">
  <query scope="subtree" targetID="target2" >
    <hasReference typeOfReference="owner">
      <toPsoID ID="2244" targetID="target2"/>
    </hasReference>
  </query>
</searchRequest>
```

3505 The provider returns a <searchResponse>. The "status" attribute of the <searchResponse>  
3506 indicates that the provider successfully executed the search operation.

```
<searchResponse requestID="138" status="success">
  <pso>
    <psoID ID="1431" targetID="target1"/>
  </pso>
</searchResponse>
```

### 3507 **3.6.7.2 iterate**

3508 The iterate operation obtains the next set of objects from the result set that the provider selected for  
3509 a search operation. (See the description of the [search operation](#) above.)

3510 The subset of the Search Capability XSD that is most relevant to the iterate operation follows.

```
<complexType name="ResultsIteratorType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="ID" type="xsd:ID"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="SearchResponseType">
  <complexContent>
    <extension base="spml:ResponseType">
      <sequence>
        <element name="pso" type="spml:PSOType" minOccurs="0"
maxOccurs="unbounded"/>
        <element name="iterator"
type="spmlsearch:ResultsIteratorType" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="IterateRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="iterator"
type="spmlsearch:ResultsIteratorType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

```
</complexContent>
</complexType>

<element name="iterateRequest" type="spmlsearch:IterateRequestType"/>
<element name="iterateResponse" type="spmlsearch:SearchResponseType"/>
```

3511 **An iterateRequest receives an iterateResponse.** A requestor supplies as input to an  
3512 <iterateRequest> the <iterator> that was part of the original <searchResponse> or the  
3513 <iterator> that was part of a subsequent <iterateResponse>, whichever is most recent. A  
3514 provider returns an <iterateResponse> in response to each <iterateRequest>. An  
3515 <iterateResponse> has the same structure as a <searchResponse>.

3516 The <iterateResponse> will contain at least one <ps0> element that represents a matching  
3517 object. If more matching objects are available to return, then the <iterateResponse> will also  
3518 contain an <iterator>. The requestor can use this <iterator> in another  
3519 <iterateRequest> to retrieve more of the matching objects.

3520 **Iterate is not batchable.** For reasons of scale, neither a search request nor an iterate request  
3521 should be nested in a [batch](#) request. When a search query matches more objects than the provider  
3522 can place directly in the response, the provider must temporarily store the remaining objects.  
3523 Storing the remaining objects allows the requestor to iterate the remaining objects, but also requires  
3524 the provider to commit resources.  
3525 See the topic named "Resource Considerations" earlier in this section.

3526 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
3527 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
3528 search results. Allowing a requestor to nest a search request or an iterate request within a batch  
3529 request would aggravate the resource problem, requiring a provider to store more information in  
3530 larger chunks for a longer amount of time.

3531 **The iterate operation must be executed synchronously.** The provider is already queuing the  
3532 result set (every object beyond those returned in the first search response), so it is unreasonable  
3533 for a requestor to ask the provider to queue the results of a request for the next item in the result  
3534 set.

3535 Furthermore, asynchronous iteration would complicate the provider's maintenance of the result set.  
3536 Since a provider could never know that the requestor had processed the results of an  
3537 asynchronous iteration, the provider would not know when to increment its position in the result set.  
3538 In order to support asynchronous iteration both correctly and generally, a provider would have to  
3539 maintain a version of every result set *for each iteration* of that result set. This would impose an  
3540 unreasonable burden on the provider.

#### 3541 [3.6.7.2.1](#) *iterateRequest (normative)*

3542 A requestor MUST send an <iterateRequest> to a provider in order to obtain any *additional*  
3543 objects that matched a previous <searchRequest> but that the provider has not yet returned to  
3544 the requestor. (That is, matching objects that were not contained in the response to that  
3545 <searchRequest> and that have not yet been contained in any response to an  
3546 <iterateRequest> associated with that <searchRequest>.)

3547 **Execution.** An <iterateRequest> MUST NOT specify "executionMode='asynchronous'".  
3548 An <iterateRequest> MUST specify "executionMode='synchronous' "  
3549 or (an <iterateRequest> MUST) omit "executionMode".  
3550 See the section titled "[Determining execution mode](#)".

3551 **iterator.** An <iterateRequest> MUST contain exactly one <iterator> element. A requestor  
3552 MUST supply as input to an <iterateRequest> the <iterator> from the original  
3553 <searchResponse> or (the requestor MUST supply as input to the <iterateRequest>) the  
3554 <iterator> from a subsequent <iterateResponse>. A requestor SHOULD supply as input  
3555 to an <iterateRequest> the most recent <iterator> that represents the search result set.

### 3556 3.6.7.2.2 *iterateResponse (normative)*

3557 A provider that receives a <iterateRequest> from a requestor that the provider trusts must  
3558 examine the content of the <iterateRequest>. If the request is valid, the provider MUST return  
3559 (the XML that represents) the next set of objects from the result set that the <iterator>  
3560 represents.

3561 **Execution.** The provider MUST execute the iterate operation synchronously (if the provider  
3562 executes the iterate operation at all). See the section titled “[Determining execution mode](#)”.

3563 **Response.** The provider MUST return to the requestor an <iterateResponse>.

3564 **Status.** The <iterateResponse> must contain a “status” attribute that indicates whether the  
3565 provider successfully returned the next set of objects from the result set that the <iterator>  
3566 represents. See the section titled “[Status \(normative\)](#)”.

3567 • If the provider successfully returned (the XML that represents) the next set of objects from the  
3568 result set that the <iterator> represents, then the <iterateResponse> MUST specify  
3569 “status=’ success’”.

3570 • If the provider encountered an error in returning (the XML that represents) the next set of  
3571 objects from the result set that the <iterator> represents, then the <iterateResponse>  
3572 MUST specify “status=’ failure’”.

3573 **PSO.** The <iterateResponse> MAY contain any number of <pso> elements.

3574 • If the <iterateResponse> specifies “status=’ success’” and *at least one object remained*  
3575 *to iterate* (in the result set that the <iterator> represents),  
3576 then the <iterateResponse> MUST contain at least one <pso> element  
3577 that contains the (XML representation of the) next matching object.

3578 • If the <iterateResponse> specifies “status=’ success’” and *no object remained to*  
3579 *iterate* (in the result set that the <iterator> represents),  
3580 then the <iterateResponse> MUST NOT contain a <pso> element.

3581 • If the <iterateResponse> specifies “status=’ failure’”,  
3582 then the <iterateResponse> MUST NOT contain a <pso> element.

3583 **PSO and ReturnData.** Each <pso> contains the subset of (the XML representation of) a requested  
3584 object that the “returnData” attribute of the original <searchRequest> specified. By default,  
3585 each <pso> contains the entire (XML representation of an) object.

3586 • A <pso> element MUST contain a <psoID> element.  
3587 The <psoID> element MUST contain the identifier of the requested object.  
3588 See the section titled “[PSO Identifier \(normative\)](#)”.

3589 • A <pso> element MAY contain a <data> element.

3590 - If the <searchRequest> specified “returnData=’ identifier’”,  
3591 then the <pso> MUST NOT contain a <data> element.

3592 - Otherwise, if the `<searchRequest>` specified `"returnData='data'"`  
3593 or (if the `<searchRequest>` specified) `"returnData='everything'"`  
3594 or (if the `<searchRequest>`) omitted the `"returnData"` attribute  
3595 then the `<data>` element MUST contain the XML representation of the object.  
3596 This XML must be valid according to the schema of the target for the schema entity of  
3597 which the newly created object is an instance.

3598 • A `<pso>` element MAY contain any number of `<capabilityData>` elements. Each  
3599 `<capabilityData>` element contains a set of *capability-specific data* that is associated with  
3600 the newly created object (for example, a *reference* to another object).

3601 - If the `<searchRequest>` specified `"returnData='identifier'"`  
3602 or (if the `<searchRequest>` specified) `"returnData='data'"`  
3603 then the `<pso>` MUST NOT contain a `<capabilityData>` element.

3604 - Otherwise, if the `<searchRequest>` specified `"returnData='everything'"`  
3605 or (if the `<searchRequest>`) omitted the `"returnData"` attribute,  
3606 then the `<pso>` MUST contain a `<capabilityData>` element for each set of capability-  
3607 specific data that is associated with the requested object  
3608 (and that is specific to a capability that the target supports for the schema entity of which  
3609 the requested object is an instance).

3610 **PSO capabilityData and IncludeDataForCapability.** An `<iterateResponse>` MUST include (as  
3611 `<capabilityData>` sub-elements of each `<pso>`) any capability-specific data that is associated  
3612 with each matching object and for which *all* of the following are true:

3613 • The original `<searchRequest>` specified `"returnData='everything'"`  
3614 or (the original `<searchRequest>`) omitted the `"returnData"` attribute.

3615 • The schema for the target declares that the *target supports the capability*  
3616 (for the schema entity of which each matching object is an instance).

3617 • The original `<searchRequest>` contained an `<includeDataForCapability>` element  
3618 that specified the capability to which the data are specific  
3619 or the original `<searchRequest>` contained no `<includeDataForCapability>` element.

3620 An `<iterateResponse>` SHOULD NOT include (as `<capabilityData>` sub-elements of each  
3621 `<pso>`) any capability-specific data for which any of the above is not true.

3622 **iterator.** A `<iterateResponse>` MAY contain at most one `<iterator>` element.

3623 • If the `<iterateResponse>` specifies `"status='success'"` and the search response  
3624 *contains the last of the objects* that matched the `<query>` that was specified in the original  
3625 `<searchRequest>`, then the `<iterateResponse>` MUST NOT contain an `<iterator>`.

3626 • If the `<iterateResponse>` specifies `"status='success'"` and the provider *still has more*  
3627 *matching objects* that have not yet been returned to the requestor, then the  
3628 `<iterateResponse>` MUST contain exactly one `<iterator>`.

3629 • If the `<iterateResponse>` specifies `"status='failure'"`, then the `<iterateResponse>`  
3630 MUST NOT contain an `<iterator>`.

3631 **iterator ID.** An `<iterator>` MUST have an "ID" attribute.

3632 The value of the "ID" attribute uniquely identifies the `<iterator>` within the namespace of the  
3633 provider. The "ID" attribute allows the provider to map each `<iterator>` token to the result set of  
3634 the requestor's `<query>` and to any state that records the requestor's position within that result set.

3635 The “ID” attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an  
3636 <iterator>. An <iterator> is not a PSO.

3637 **Error.** If the <iterateResponse> specifies “status=’ failure’”, then the  
3638 <iterateResponse> MUST have an “error” attribute that characterizes the failure.  
3639 See the general section titled ““Error (normative)”” .

3640 In addition, the <iterateResponse> MUST specify an appropriate value of “error” if any of the  
3641 following is true:

- 3642 • If the provider does not recognize the <iterator> in an <iterateRequest> as representing  
3643 a result set.
- 3644 • If the provider does not recognize the <iterator> in an <iterateRequest> as representing  
3645 any result set that the provider currently maintains.

3646 The <iterateResponse> MAY specify an appropriate value of “error” if any of the following is  
3647 true:

- 3648 • If an <iterateRequest> contains an <iterator> that is *not the most recent version* of the  
3649 <iterator>. If the provider has returned to the requestor a more recent <iterator> that  
3650 represents the same search result set, then the provider MAY reject the older <iterator>.  
3651 (A provider that changes the ID—for example, to encode the state of iteration within a search  
3652 result set—may be sensitive to this.)

### 3653 [3.6.7.2.3 \*iterate Examples \(non-normative\)\*](#)

3654 In order to illustrate the iterate operation, we first need a search operation that returns more than  
3655 one object. In the following example, a requestor asks a provider to search for every `Person` with  
3656 an email address that starts with the letter “j”.

```
<searchRequest requestID="147">
  <query scope="subTree" targetID="target2" >
    <select path="/Person/email="j*" namespaceURI="http://www.w3.org/TR/xpath20" />
  </query>
</searchRequest>
```

3657 The provider returns a <searchResponse>. The “status” attribute of the <searchResponse>  
3658 indicates that the provider successfully executed the search operation. The <searchResponse>  
3659 contains two <ps0> elements that represent the first matching objects.

```
<searchResponse requestID="147" status="success">
  <ps0>
    <data>
      <Person cn="jeff" firstName="Jeff" lastName="Beck" fullName="Jeff Beck">
        <email>jeffbeck@example.com</email>
      </Person>
    </data>
    <ps0ID ID="0001" targetID="target2"/>
  </ps0>
  <ps0>
    <data>
      <Person cn="jimi" firstName="Jimi" lastName="Hendrix" fullName="Jimi Hendrix">
        <email>jimi@example.com</email>
      </Person>
    </data>
```

```
<psoID ID="0002" targetID="target2"/>
</pso>
<iterator ID="1900"/>
</searchResponse>
```

3660 The requestor asks the provider to return the next matching objects (in the result set for the  
3661 search). The requestor supplies the <iterator> from the <searchResponse> as input to the  
3662 <iterateRequest>.

```
<iterateRequest requestID="148">
  <iterator ID="1900"/>
</iterateRequest>
```

3663 The provider returns an <iterateResponse> in response to the <iterateRequest>. The  
3664 "status" attribute of the <iterateResponse> indicates that the provider successfully executed  
3665 the iterate operation. The <iterateResponse> contains two <pso> elements that represent the  
3666 next matching objects.

```
<iterateResponse requestID="148" status="success">
  <pso>
    <data>
      <Person cn="jt" firstName="James" lastName="Taylor" fullName="James Taylor">
        <email>jt@example.com</email>
      </Person>
    </data>
    <psoID ID="0003" targetID="target2"/>
  </pso>
  <pso>
    <data>
      <Person cn="jakob" firstName="Jakob" lastName="Dylan" fullName="Jakob Dylan">
        <email>jakobdylan@example.com</email>
      </Person>
    </data>
    <psoID ID="0004" targetID="target2"/>
  </pso>
  <iterator ID="1901"/>
</iterateResponse>
```

3667 The <iterateResponse> also contains another <iterator> element. The "ID" of this  
3668 <iterator> differs from the "ID" of the <iterator> in the original <searchResponse>. The  
3669 "ID" could remain constant (for each iteration of the result set that the <iterator> represents) if  
3670 the provider so chooses, but the "ID" value could change (e.g., if the provider uses "ID" to  
3671 encode the state of the result set).

3672 To get the final matching object, the requestor again supplies the <iterator> from the  
3673 <iterateResponse> as input to the <iterateRequest>.

```
<iterateRequest requestID="149">
  <iterator ID="1901"/>
</iterateRequest>
```

3674 The provider again returns an <iterateResponse> in response to the <iterateRequest>. The  
3675 "status" attribute of the <iterateResponse> indicates that the provider successfully executed  
3676 the iterate operation. The <iterateResponse> contains a <pso> element that represents the  
3677 final matching object. Since all of the matching objects have now been returned to the requestor,  
3678 this <iterateResponse> contains no <iterator>.

```
<iterateResponse requestID="149" status="success">
  <pso>
```

```

    <data>
      <Person cn="joebob" firstName="JoeBob" lastName="Briggs" fullName="JoeBob
Briggs">
        <email>joebob@example.com</email>
      </Person>
    </data>
    <psolD ID="2244" targetID="target2"/>
  </pso>
</iterateResponse>

```

### 3679 3.6.7.3 closeliterator

3680 The closeliterator operation tells the provider that the requestor has no further need for the search  
3681 result that a specific <iterator> represents. (See the description of the [search operation](#) above.)

3682 A requestor should send a <closeIteratorRequest> to the provider when the requestor no  
3683 longer intends to iterate a search result. (A provider will eventually free an inactive search result --  
3684 even if the provider never receives a <closeIteratorRequest> from the requestor-- but this  
3685 behavior is unspecified.) For more information, see the topic named "Resource Considerations"  
3686 topic earlier within this section.

3687 The subset of the Search Capability XSD that is most relevant to the iterate operation follows.

```

<complexType name="ResultsIteratorType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="ID" type="xsd:ID"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="CloseIteratorRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="iterator"
type="spmlsearch:ResultsIteratorType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="closeIteratorRequest"
type="spmlsearch:CloseIteratorRequestType"/>
<element name="closeIteratorResponse" type="spml:ResponseType"/>

```

3688 **A closeliteratorRequest receives a closeliteratorResponse.** A requestor supplies as input to a  
3689 <closeIteratorRequest> the <iterator> that was part of the original <searchResponse>  
3690 or the <iterator> that was part of a subsequent <iterateResponse>, whichever is most  
3691 recent. A provider returns a <closeIteratorResponse> in response to each  
3692 <closeIteratorRequest>. A <closeIteratorResponse> has the same structure as an  
3693 <spml:response>.

3694 **closeliterator is not batchable.** For reasons of scale, neither of a search request nor an iterate  
3695 request nor a closeliterator request should be nested in a [batch](#) request. When a search query

3696 matches more objects than the provider can place directly in the response, the provider must  
3697 temporarily store the remaining objects. Storing the remaining objects allows the requestor to  
3698 iterate the remaining objects, but also requires the provider to commit resources.  
3699 See the topic named "Resource Considerations" earlier in this section.

3700 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
3701 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
3702 search results. Allowing a requestor to nest a search request or an iterate request or a closeliterator  
3703 request within a batch request would aggravate the resource problem, requiring a provider to store  
3704 more information in larger chunks for a longer amount of time.

3705 **The closeliterator operation must be executed synchronously.** The provider is already queuing  
3706 the result set (every object beyond those returned in the first search response), so a request to  
3707 close the iterator (and thus to free the system resources associated with the result set) should be  
3708 executed as soon as possible. It is unreasonable for a requestor to ask the provider to queue the  
3709 results of a request to close an iterator (especially since the close iterator response contains little or  
3710 no information beyond success or failure).

### 3711 [3.6.7.3.1 closeliteratorRequest \(normative\)](#)

3712 A requestor SHOULD send a `<closeIteratorRequest>` to a provider when the requestor no  
3713 longer intends to iterate a search result. (This allows the provider to free any system resources  
3714 associated with the search result.).

3715 **Execution.** A `<closeIteratorRequest>` MUST NOT specify  
3716 "executionMode='asynchronous'".

3717 A `<closeIteratorRequest>` MUST specify "executionMode='synchronous' "  
3718 or (a `<closeIteratorRequest>` MUST) omit "executionMode".  
3719 See the section titled "[Determining execution mode](#)".

3720 **iterator.** A `<closeIteratorRequest>` MUST contain exactly one `<iterator>` element. A  
3721 requestor MUST supply as input to a `<closeIteratorRequest>` the `<iterator>` from the  
3722 original `<searchResponse>` or (a requestor MUST supply the `<iterator>` from a subsequent  
3723 `<iterateResponse>`). A requestor SHOULD supply as input to a  
3724 `<closeIteratorRequest>` the most recent `<iterator>` that represents the search result set.

3725 **iterator ID.** An `<iterator>` that is part of a `<closeIteratorRequest>` MUST have an "ID"  
3726 attribute. (The value of the "ID" attribute uniquely identifies the `<iterator>` within the  
3727 namespace of the provider. The "ID" attribute allows the provider to map each `<iterator>`  
3728 token to the result set of the requestor's `<query>` and also (allows the provider to map each  
3729 `<iterator>` token) to any state that records the requestor's iteration *within* that result set.)

### 3730 [3.6.7.3.2 closeliteratorResponse \(normative\)](#)

3731 A provider that receives a `<closeIteratorRequest>` from a requestor that the provider trusts  
3732 must examine the content of the `<closeIteratorRequest>`. If the request is valid, the provider  
3733 MUST release any search result set that the `<iterator>` represents. Any subsequent request to  
3734 iterate that same search result set MUST fail.

3735 **Execution.** The provider MUST execute the closeliterator operation synchronously (if the provider  
3736 executes the closeliterator operation at all). See the section titled "[Determining execution mode](#)".

3737 **Response.** The provider MUST return to the requestor a `<closeIteratorResponse>`.

3738 **Status.** The `<closeIteratorResponse>` must contain a "status" attribute that indicates  
3739 whether the provider successfully released the search result set that the `<iterator>` represents.  
3740 See the section titled "[Status \(normative\)](#)".

3741 • If the provider successfully released the search result set that the `<iterator>` represents,  
3742 then the `<closeIteratorResponse>` MUST specify "status='success'".

3743 • If the provider encountered an error in releasing the search result set that the `<iterator>`  
3744 represents, then the `<closeIteratorResponse>` MUST specify "status='failure'".

3745 **Error.** If the `<closeIteratorResponse>` specifies "status='failure'", then the  
3746 `<closeIteratorResponse>` MUST have an "error" attribute that characterizes the failure.  
3747 See the general section titled "[Error \(normative\)](#)".

3748 In addition, the `<closeIteratorResponse>` MUST specify an appropriate value of "error" if  
3749 any of the following is true:

3750 • If the provider does not recognize the `<iterator>` in a `<closeIteratorRequest>` as  
3751 representing a search result set.

3752 • If the provider does not recognize the `<iterator>` in a `<closeIteratorRequest>` as  
3753 representing any search result set that the provider currently maintains.

3754 • If the provider recognized the `<iterator>` in a `<closeIteratorRequest>` as representing  
3755 a search result set that the provider currently maintains but *cannot release the resources*  
3756 *associated with that search result set*.

3757 The `<closeIteratorResponse>` MAY specify an appropriate value of "error" if any of the  
3758 following is true:

3759 • If a `<closeIteratorRequest>` contains an `<iterator>` that is *not the most recent version*  
3760 *of the <iterator>*. If the provider has returned to the requestor a more recent `<iterator>`  
3761 that represents the same search result set, then the provider MAY reject the older  
3762 `<iterator>`.

3763 (A provider that changes the ID—for example, to encode the state of iteration within a search  
3764 result set—may be sensitive to this.)

### 3765 [3.6.7.3.3 closeIterator Examples \(non-normative\)](#)

3766 In order to illustrate the `closeIterator` operation, we first need a search operation that returns more  
3767 than one object. In the following example, a requestor asks a provider to search for every `Person`  
3768 with an email address that starts with the letter "j".

```
<searchRequest requestID="150">
  <query scope="subTree" targetID="target2" >
    <select path="/Person/email="j*" namespaceURI="http://www.w3.org/TR/xpath20" />
  </query>
</searchRequest>
```

3769 The provider returns a `<searchResponse>`. The "status" attribute of the `<searchResponse>`  
3770 indicates that the provider successfully executed the search operation. The `<searchResponse>`  
3771 contains two `<pso>` elements that represent the first matching objects.

```
<searchResponse request="150" status="success">
  <pso>
    <data>
      <Person cn="jeff" firstName="Jeff" lastName="Beck" fullName="Jeff Beck">
        <email>jeffbeck@example.com</email>
      </Person>
    </data>
    <psolD ID="0001" targetID="target2"/>
  </pso>
  <pso>
    <data>
      <Person cn="jimi" firstName="Jimi" lastName="Hendrix" fullName="Jimi Hendrix">
        <email>jimi@example.com</email>
      </Person>
    </data>
    <psolD ID="0002" targetID="target2"/>
  </pso>
  <iterator ID="1900"/>
</searchResponse>
```

- 3772 The requestor decides that the two objects in the initial `<searchResponse>` will suffice, and does  
3773 not intend to retrieve any more matching objects (in the result set for the search). The requestor  
3774 supplies the `<iterator>` from the `<searchResponse>` as input to the  
3775 `<closeIteratorRequest>`.

```
<closeIteratorRequest requestID="151">
  <iterator ID="1900"/>
</closeIteratorRequest>
```

- 3776 The provider returns a `<closeIteratorResponse>` in response to the  
3777 `<closeIteratorRequest>`. The "status" attribute of the `<closeIteratorResponse>`  
3778 indicates that the provider successfully released the result set.

```
<closeIteratorResponse requestID="151" status="success"/>
```

### 3779 **3.6.8 Suspend Capability**

3780 The Suspend Capability is defined in a schema associated with the following XML namespace:  
3781 `urn:oasis:names:tc:SPML:2:0:suspend`. This document includes the Suspend Capability  
3782 XSD as Appendix H.

3783 The Suspend Capability defines three operations: suspend, resume and active.

- 3784 • The suspend operation *disables an object* (immediately or on a specified date).
- 3785 • The resume operation *re-enables an object* (immediately or on a specified date).
- 3786 • The active operation *tests whether an object is currently suspended*.

3787 The suspend operation disables an object *persistently* (rather than transiently). The suspend  
3788 operation is intended to revoke the privileges of an account, for example, while the authorized user  
3789 of the account is on vacation.

3790 The resume operation re-enables an object persistently. One might use the resume operation to  
3791 restore privileges for an account, for example, when the authorized user of the account returns from  
3792 vacation.

3793 A provider that supports the suspend, resume and active operations for a target SHOULD declare  
3794 that the target supports the Suspend Capability. A provider that does not support all of suspend,  
3795 resume and active MUST NOT declare that the target supports the Suspend Capability.

3796 **Idempotent.** The suspend operation and the resume operation are both *idempotent*. Any requestor  
3797 should be able to suspend (or to resume) the same object multiple times without error.

3798 **Search.** A requestor can *search for objects based on enabled state* using the `<isActive>` query  
3799 clause. The `{IsActiveType}` extends `{QueryClauseType}`, which indicates that an instance  
3800 of `{IsActiveType}` can be used to select objects. An `<isActive>` clause matches an object if  
3801 and only if the object is currently enabled. In order to select disabled objects, a requestor would  
3802 combine this clause with the logical operator `<not>`. See the section titled “[Selection](#)”.

#### 3803 **3.6.8.1 suspend**

3804 The suspend operation enables a requestor to disable an object.

3805 The subset of the Suspend Capability XSD that is most relevant to the suspend operation follows.

```
<complexType name="SuspendRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType"/>
      </sequence>
      <attribute name="effectiveDate" type="dateTime"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

<element name="suspendRequest" type="spml:suspend:SuspendRequestType"/>
<element name="suspendResponse" type="spml:ResponseType"/>
```

3806 **3.6.8.1.1 suspendRequest (normative)**

3807 A requestor MUST send a <suspendRequest> to a provider in order to (ask the provider to)  
3808 disable an existing object.

3809 **Execution.** A <suspendRequest> MAY specify "executionMode".  
3810 See the section titled "Determining execution mode".

3811 **psoid.** A <suspendRequest> MUST contain exactly one <psoid> element. A <psoid> element  
3812 MUST identify an object that exists on a target that is exposed by the provider.  
3813 See the section titled "PSO Identifier (normative)".

3814 **EffectiveDate.** A <suspendRequest> MAY specify an "effectiveDate". Any  
3815 "effectiveDate" value MUST be expressed in UTC form, with no time zone component.  
3816 A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.  
3817 A requestor MUST NOT generate time instants that specify leap seconds.

3818 **3.6.8.1.2 suspendResponse (normative)**

3819 A provider that receives a <suspendRequest> from a requestor that the provider trusts MUST  
3820 examine the content of the <suspendRequest>. If the request is valid and if the specified object  
3821 exists, then the provider MUST disable the object that the <psoid> specifies.

3822 If the <suspendRequest> specifies an "effectiveDate", the provider MUST enable the  
3823 specified object as of that date.

- 3824 • If the "effectiveDate" of the <suspendRequest> is in the past, then  
3825 the provider MUST do one of the following:
- 3826 - The provider MAY disable the specified object *immediately*.
  - 3827 - The provider MAY return an error. (The provider's response SHOULD indicate that the  
3828 request failed because the effective date is past.)
- 3829 • If the "effectiveDate" of the <suspendRequest> is in the future, then
- 3830 - The provider MUST NOT disable the specified object until that future date and time.
  - 3831 - The provider MUST disable the specified object at that future date and time  
3832 (unless a subsequent request countermands this request).

3833 **Execution.** If an <suspendRequest> does not specify "executionMode",  
3834 the provider MUST choose a type of execution for the requested operation.  
3835 See the section titled "Determining execution mode".

3836 **Response.** The provider must return to the requestor a <suspendResponse>. The  
3837 <suspendResponse> must have a "status" attribute that indicates whether the provider  
3838 successfully disabled the specified object. See the section titled "Status (normative)".

3839 **Error.** If the provider cannot create the requested object, the <suspendResponse> must contain  
3840 an error attribute that characterizes the failure. See the general section titled "Error (normative)".

3841 In addition, the <suspendResponse> MUST specify an appropriate value of "error" if any of the  
3842 following is true:

- 3843 • The <suspendRequest> contains a <psoid> for an object that does not exist.
- 3844 • The <suspendRequest> specifies an "effectiveDate" that is not valid.

3845 The provider MAY return an error if any of the following is true:

- 3846 • The `<suspendRequest>` specifies an "effectiveDate" that is in the past.
- 3847 The provider MUST NOT return an error when (the operation would otherwise succeed and) the  
3848 object is already disabled. In this case, the `<suspendResponse>` MUST specify  
3849 "status='success'".

### 3850 **3.6.8.1.3** *suspend Examples (non-normative)*

3851 In the following example, a requestor asks a provider to suspend an existing `Person` object.

```
<suspendRequest requestID="139">  
  <psoID ID="2244" targetID="target2"/>  
</suspendRequest>
```

3852 The provider returns an `<suspendResponse>` element. The "status" attribute of the  
3853 `<suspendResponse>` indicates that the provider successfully disabled the specified object.

```
<suspendResponse requestID="139" status="success"/>
```

3854 In the following example, a requestor asks a provider to suspend an existing account.

```
<suspendRequest requestID="140">  
  <psoID ID="1431" targetID="target1"/>  
</suspendRequest>
```

3855 The provider returns a `<suspendResponse>`. The "status" attribute of the  
3856 `<suspendResponse>` indicates that the provider successfully disabled the specified account.

```
<suspendResponse requestID="140" status="success"/>
```

### 3857 **3.6.8.2** *resume*

3858 The resume operation enables a requestor to re-enable an object that has been suspended. (See  
3859 the description of the [suspend](#) operation above.)

3860 The subset of the Suspend Capability XSD that is most relevant to the resume operation follows.

```
<complexType name="ResumeRequestType">  
  <complexContent>  
    <extension base="spml:RequestType">  
      <sequence>  
        <element name="psoID" type="spml:PSOIdentifierType"/>  
      </sequence>  
      <attribute name="effectiveDate" type="dateTime"  
use="optional"/>  
    </extension>  
  </complexContent>  
</complexType>  
  
<element name="ResumeRequest" type="spml:suspend:ResumeRequestType"/>  
<element name="ResumeResponse" type="spml:ResponseType"/>
```

### 3861 **3.6.8.2.1** *resumeRequest (normative)*

3862 A requestor MUST send a `<resumeRequest>` to a provider in order to (ask the provider to) re-  
3863 enable an existing object.

3864 **Execution.** A <resumeRequest> MAY specify "executionMode".  
3865 See the section titled "Determining execution mode".

3866 **psoid.** A <resumeRequest> MUST contain exactly one <psoid> element. A <psoid> element  
3867 MUST identify an object that exists on a target (that is supported by the provider).  
3868 See the section titled "PSO Identifier (normative)".

3869 **EffectiveDate.** A <resumeRequest> MAY specify an "effectiveDate". Any  
3870 "effectiveDate" value MUST be expressed in UTC form, with no time zone component.  
3871 A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.  
3872 A requestor MUST NOT generate time instants that specify leap seconds.

### 3873 3.6.8.2.2 resumeResponse (normative)

3874 A provider that receives a <resumeRequest> from a requestor that the provider trusts MUST  
3875 examine the content of the <resumeRequest>. If the request is valid and if the specified object  
3876 exists, then the provider MUST enable the object that is specified by the <psoid>.

3877 If the <resumeRequest> specifies an "effectiveDate", the provider MUST enable the  
3878 specified object as of that date.

3879 • If the "effectiveDate" of the <resumeRequest> is in the past, then  
3880 the provider MUST do one of the following:  
3881 - The provider MAY enable the specified object *immediately*.  
3882 - The provider MAY return an error. (The provider's response SHOULD indicate that the  
3883 request failed because the effective date is past.)

3884 • If the "effectiveDate" of the <resumeRequest> is in the future, then  
3885 - The provider MUST NOT enable the specified object until that future date and time.  
3886 - The provider MUST enable the specified object at that future date and time  
3887 (unless a subsequent request countermands this request).

3888 **Execution.** If an <resumeRequest> does not specify "executionMode",  
3889 the provider MUST choose a type of execution for the requested operation.  
3890 See the section titled "Determining execution mode".

3891 **Response.** The provider must return to the requestor a <resumeResponse>. The  
3892 <resumeResponse> must have a "status" attribute that indicates whether the provider  
3893 successfully enabled the specified object. See the section titled "Status (normative)".

3894 **Error.** If the provider cannot enable the requested object, the <resumeResponse> must contain  
3895 an error attribute that characterizes the failure. See the general section titled "Error (normative)".

3896 In addition, the <resumeResponse> MUST specify an appropriate value of "error" if any of the  
3897 following is true:

- 3898 • The <resumeRequest> contains a <psoid> for an object that does not exist.
- 3899 • The <resumeRequest> specifies an "effectiveDate" that is not valid.

3900 The provider MAY return an error if any of the following is true:

- 3901 • The <resumeRequest> specifies an "effectiveDate" that is in the past.

3902 The provider MUST NOT return an error when (the operation would otherwise succeed and) the  
3903 object is already enabled. In this case, the response should specify "status='success'".

3904 **3.6.8.2.3** *resume Examples (non-normative)*

3905 In the following example, a requestor asks a provider to resume an existing `Person` object.

```
<resumeRequest requestID="141">
  <psoid ID="2244" targetID="target2"/>
</resumeRequest>
```

3906 The provider returns a `<resumeResponse>` element. The `"status"` attribute of the  
3907 `<resumeResponse>` element indicates that the provider successfully disabled the specified object.

```
<resumeResponse requestID="141" status="success"/>
```

3908 In the following example, a requestor asks a provider to resume an existing account.

```
<resumeRequest requestID="142">
  <psoid ID="1431" targetID="target1"/>
</resumeRequest>
```

3909 The provider returns a `<resumeResponse>`. The `"status"` attribute of the  
3910 `<resumeResponse>` indicates that the provider successfully enabled the specified account.

```
<resumeResponse requestID="142" status="success"/>
```

3911 **3.6.8.3** *active*

3912 The active operation enables a requestor to determine whether a specified object has been  
3913 suspended. (See the description of the [suspend](#) operation above.)

3914 The subset of the Suspend Capability XSD that is most relevant to the active operation follows.

```
<complexType name="ActiveRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="psoid" type="spml:PSOIdentifierType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="ActiveResponseType">
  <complexContent>
    <extension base="spml:ResponseType">
      <attribute name="active" type="boolean" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<element name="ActiveRequest" type="spmlsuspend:ActiveRequestType"/>
<element name="ActiveResponse" type="spmlsuspend:ActiveResponseType"/>
```

3915 **3.6.8.3.1** *activeRequest (normative)*

3916 A requestor **MUST** send an `<activeRequest>` to a provider in order to (ask the provider to)  
3917 determine whether the specified object is enabled (active) or disabled.

3918 **Execution.** An `<activeRequest>` MAY specify "executionMode".  
3919 See the section titled "[Determining execution mode](#)".

3920 **psoid.** A `<activeRequest>` MUST contain exactly one `<psoid>` element. A `<psoid>` element  
3921 MUST identify an object that exists on a target that is exposed by the provider.  
3922 See the section titled "[PSO Identifier \(normative\)](#)".

### 3923 [3.6.8.3.2 activeResponse \(normative\)](#)

3924 A provider that receives a `<activeRequest>` from a requestor that the provider trusts MUST  
3925 examine the content of the `<activeRequest>`. If the request is valid and if the specified object  
3926 exists, then the provider MUST disable the object that is specified by the `<psoid>`.

3927 **Execution.** If an `<activeRequest>` does not specify "executionMode", the provider MUST  
3928 choose a type of execution for the requested operation.  
3929 See the section titled "[Determining execution mode](#)".

3930 **Response.** The provider must return to the requestor an `<activeResponse>`. The  
3931 `<activeResponse>` must have a "status" attribute that indicates whether the provider  
3932 successfully determined whether the specified object is enabled (i.e. active).  
3933 See the section titled "[Status \(normative\)](#)".

3934 **active.** An `<activeResponse>` MAY have an "active" attribute that indicates whether the  
3935 specified object is suspended. An `<activeResponse>` that specifies "status=' success' "  
3936 MUST have an "active" attribute.

3937 • If the specified object is suspended, the `<activeResponse>` MUST specify  
3938 "active=' false'".

3939 • If the specified object is not suspended, the `<activeResponse>` MUST specify  
3940 "active=' true'".

3941 **Error.** If the provider cannot determine whether the requested object is suspended, the  
3942 `<activeResponse>` must contain an "error" attribute that characterizes the failure.  
3943 See the general section titled "[Error \(normative\)](#)".

3944 In addition, the `<activeResponse>` MUST specify an appropriate value of "error" if any of the  
3945 following is true:

3946 • The `<activeRequest>` contains a `<psoid>` that specifies an object that does not exist.

### 3947 [3.6.8.3.3 active Examples \(non-normative\)](#)

3948 In the following example, a requestor asks a provider whether a `Person` object is active.

```
<activeRequest requestID="143">  
  <psoid ID="2244" targetID="target2"/>  
</activeRequest>
```

3949 The provider returns an `<activeResponse>` element. The "status" attribute of the  
3950 `<activeResponse>` element indicates that the provider successfully completed the requested  
3951 operation. The "active" attribute of the `<activeResponse>` indicates that the specified object is  
3952 active.

```
<activeResponse requestID="143" status="success" active="true"/>
```

3953 In the following example, a requestor asks a provider whether an account is active.

```
<activeRequest requestID="144">  
  <psolD ID="1431" targetID="target1"/>  
</activeRequest>
```

- 3954 The provider returns an <activeResponse>. The "status" attribute of the  
3955 <activeResponse> indicates that the provider successfully completed the requested operation.  
3956 The "active" attribute of the <activeResponse> indicates that the specified object is active.

```
<activeResponse requestID="144" status="success" active="true"/>
```

### 3957 3.6.9 Updates Capability

3958 The Updates Capability is defined in a schema associated with the following XML namespace:  
3959 `urn:oasis:names:tc:SPML:2:0:updates`. This document includes the Updates Capability  
3960 XSD as Appendix I.

3961 The Updates Capability defines three operations: `updates`, `iterate` and `closeiterator`. The updates  
3962 and iterate operations together allow a requestor to obtain *in a scalable manner* every recorded  
3963 *update* (i.e., modification to an object) that matches specified selection criteria. The updates  
3964 operation returns in its response a first set of matching updates. Each subsequent iterate operation  
3965 returns more matching updates. The closeiterator operation allows a requestor to tell a provider that  
3966 it does not intend to finish iterating a result set and that the provider may therefore release the  
3967 associated resources).

3968 A provider that supports the updates and iterate operations for a target SHOULD declare that the  
3969 target supports the Updates Capability. A provider that does not support both updates and iterate  
3970 MUST NOT declare that the target supports the Updates Capability.

3971 **Resource considerations.** A provider must limit the size and duration of its updates result sets (or  
3972 that provider will exhaust available resources). A provider must decide:

- 3973 • How large of an updates result set the provider will *select* on behalf of a requestor.
- 3974 • How large of an updates result set the provider will *queue* on behalf of a requestor  
3975 (so that the requestor may iterate the updates result set).
- 3976 • For *how long a time* the provider will queue an updates result set on behalf of a requestor.

3977 These decisions may be governed by the provider's implementation, by its configuration, or by  
3978 runtime computation.

3979 A provider that wishes to *never to queue updates result sets* may return every matching object (up  
3980 to the provider's limit and up to any limit that the request specifies) in the updates response. Such  
3981 a provider would never return an iterator, and would not need to support the iterate operation. The  
3982 disadvantage is that, without an iterate operation, a provider's updates capability either is limited to  
3983 small results or produces large updates responses.

3984 A provider that wishes to support the iterate operation must store (or somehow queue) the updates  
3985 selected by an updates operation until the requestor has a chance to iterate those results. (That is,  
3986 a provider must somehow queue the updates that matched the criteria of an updates operation and  
3987 that were not returned in the updates response.)

3988 If all goes well, the requestor will continue to iterate the updates result set until the provider has  
3989 sent all of the updates to the requestor. The requestor may also use the closeiterator operation to  
3990 tell the provider that the requestor is no longer interested in the search result. Once all of the  
3991 updates have been sent to the requestor, the provider may free any resource that is still associated  
3992 with the updates result set. However, it is possible that the requestor may not iterate the updates  
3993 result set in a timely manner--or that the requestor may *never* iterate the updates result set  
3994 completely. Such a requestor may also neglect to close the iterator.

3995 A provider cannot queue updates result sets indefinitely. The provider must eventually release the  
3996 resources associated with an updates result set. (Put differently, any iterator that a provider returns  
3997 to a requestor must eventually expire.) Otherwise, the provider may run out of resources.

3998 Providers should carefully manage the resources associated with updates result sets. For example:

- 3999 • A provider may define a *timeout interval* that specifies the maximum time between iterate  
4000 requests. If a requestor does not request an iterate operation within this interval, the provider  
4001 will release the resources associated with the result set. This invalidates any iterator that  
4002 represents this result set.
- 4003 • A provider may also define an overall *result lifetime* that specifies the maximum length of time  
4004 to retain a result set. After this amount of time has passed, the provider will release the result  
4005 set.
- 4006 • A provider may also wish to enforce an *overall limit* on the resources available to queue result  
4007 sets, and may wish to adjust its behavior (or even to refuse updates requests) accordingly.
- 4008 • To prevent denial of service attacks, the provider should not allocate any resource on behalf of  
4009 a requestor until that requestor is properly authenticated.  
4010 See the section titled "[Security and Privacy Considerations](#)".

### 4011 **3.6.9.1 updates**

4012 The updates operation obtains *records of changes to objects*. A requestor may select change  
4013 records based on changed-related criteria and (may also select change records) based on the set  
4014 of objects.

4015 The subset of the Updates Capability XSD that is most relevant to the updates operation follows.

```

<complexType name="UpdatesRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element ref="spmlsearch:query" minOccurs="0"/>
        <element name="updatedByCapability" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="updatedSince" type="xsd:dateTime"
use="optional"/>
      <attribute name="token" type="xsd:string" use="optional"/>
      <attribute name="maxSelect" type="xsd:int" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<simpleType name="UpdateKindType">
  <restriction base="string">
    <enumeration value="add"/>
    <enumeration value="modify"/>
    <enumeration value="delete"/>
    <enumeration value="capability"/>
  </restriction>
</simpleType>

<complexType name="UpdateType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

```

        <attribute name="timestamp" type="xsd:dateTime"
use="required"/>
        <attribute name="updateKind"
type="spmlupdates:UpdateKindType" use="required"/>
        <attribute name="wasUpdatedByCapability" type="xsd:string"
use="optional"/>
    </extension>
</complexContent>
</complexType>

<complexType name="ResultsIteratorType">
    <complexContent>
        <extension base="spml:ExtensibleType">
            <attribute name="ID" type="xsd:ID"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="UpdatesResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <sequence>
                <element name="update" type="spmlupdates:UpdateType"
minOccurs="0" maxOccurs="unbounded"/>
                <element name="iterator"
type="spmlupdates:ResultsIteratorType" minOccurs="0"/>
            </sequence>
            <attribute name="token" type="xsd:string" use="optional"/>
        </extension>
    </complexContent>
</complexType>

    <element name="updatesRequest" type="spmlupdates:UpdatesRequestType"/>
    <element name="updatesResponse"
type="spmlupdates:UpdatesResponseType"/>

```

4016 The <query> is the same type of element that is specified as part of a <bulkModifyRequest> or  
4017 a <bulkDeleteRequest> or a <searchRequest>. This <query> selects the objects for which  
4018 the provider will return recorded updates. See the section titled "[SearchQueryType](#)".

4019 The "updatedSince" attribute allows the requestor to select only updates that occurred since a  
4020 specific date and time.

4021 If the updates operation is successful *but selects no matching update*, the <updatesResponse>  
4022 will not contain an <update>.

4023 If the updates operation is successful *and selects at least one matching update*, the  
4024 <updatesResponse> will contain any number of <update> elements, each of which represents a  
4025 matching update. If the updates operation selects more matching updates than the  
4026 <updatesResponse> contains, the <updatesResponse> will also contain an <iterator> that  
4027 the requestor can use to retrieve more matching updates. (See the description of the [iterate](#)  
4028 operation below.)

4029 If an updates operation would select more updates than the provider can queue for subsequent  
4030 iteration by the requestor, the provider's <updatesResponse> will specify  
4031 "error='resultSetTooLarge'".

4032 **Updates is not batchable.** For reasons of scale, neither an updates request nor an iterate request  
4033 should be nested in a [batch](#) request. When an updates query matches more updates than the  
4034 provider can place directly in the response, the provider must temporarily store the remaining  
4035 updates. Storing the remaining updates allows the requestor to iterate the remaining updates, but  
4036 also requires the provider to commit resources.  
4037 See the topic named "Resource Considerations" earlier in this section.

4038 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
4039 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
4040 updates result sets. Allowing a requestor to nest an updates request within a batch request would  
4041 aggravate the resource problem, requiring a provider to store more information in larger chunks for  
4042 a longer amount of time.

### 4043 [3.6.9.1.1 updatesRequest \(normative\)](#)

4044 A requestor **MUST** send an `<updatesRequest>` to a provider in order to (ask the provider to)  
4045 obtain every update that matches specified selection criteria.

4046 **Execution.** An `<updatesRequest>` **MAY** specify "executionMode".  
4047 See the section titled "[Determining execution mode](#)".

4048 **query.** A `<query>` describes criteria that (the provider must use to) select objects on a target.  
4049 The provider will return only updates that affect objects that match these criteria.  
4050 An `<updatesRequest>` **MAY** contain at most one `<query>` element.

- 4051 • If the provider's `<listTargetsResponse>` contains only a single `<target>`,  
4052 then an `<updatesRequest>` may omit the `<query>` element.
- 4053 • If the provider's `<listTargetsResponse>` contains more than one `<target>`,  
4054 then an `<updatesRequest>` **MUST** contain exactly one `<query>` element  
4055 and that `<query>` must specify "targetID".

4056 See the section titled "[SearchQueryType in a Request \(normative\)](#)".

4057 **updatedByCapability.** An `<updatesRequest>` **MAY** contain any number of  
4058 `<updatedByCapability>` elements. Each `<updatedByCapability>` element contains the  
4059 URN of an XML namespace that uniquely identifies a capability. Each `<updatedByCapability>`  
4060 element must identify a capability that the target supports.

- 4061 • A requestor that wants the provider to return no update that reflects a change to capability-  
4062 specific data associated with an object **MUST NOT** place an `<updatedByCapability>`  
4063 element in its `<updatesRequest>`.
- 4064 • A requestor that wants the provider to return updates that reflect changes to capability-specific  
4065 data associated with one or more objects **MUST** specify each capability (for which the provider  
4066 should return updates) as an `<updatedByCapability>` element in its `<updatesRequest>`.

4067 **updatedSince.** A `<updatesRequest>` **MAY** have an "updatedSince" attribute. (The provider  
4068 will return only updates with a timestamp greater than this value.)

4069 Any "updatedSince" value **MUST** be expressed in UTC form, with no time zone component.  
4070 A requestor or a provider **SHOULD NOT** rely on time resolution finer than milliseconds.  
4071 A requestor **MUST NOT** generate time instants that specify leap seconds.

4072 **maxSelect.** An `<updatesRequest>` **MAY** have a "maxSelect" attribute. The value of the  
4073 "maxSelect" attribute specifies the maximum number of updates the provider should select.

4074 **token.** An <updatesRequest> MAY have a "token" attribute. Any "token" value MUST  
4075 match a value that the provider returned to the requestor as the value of the "token" attribute in a  
4076 previous <updatesResponse> for the same target. Any "token" value SHOULD match the  
4077 (value of the "token" attribute in the) provider's *most recent* <updatesResponse> for the same  
4078 target.

### 4079 **3.6.9.1.2 updatesResponse (normative)**

4080 A provider that receives an <updatesRequest> from a requestor that the provider trusts must  
4081 examine the content of the <updatesRequest>. If the request is valid, the provider MUST return  
4082 updates that represent every change (that occurred since any time specified as "updatedSince")  
4083 to every object that matches the specified <query> (if the provider can possibly do so). However,  
4084 the number of updates selected (for immediate return or for eventual iteration) MUST NOT exceed  
4085 any limit specified as "maxSelect" in the <updatesRequest>.

4086 **Execution.** If an <updatesRequest> does not specify "executionMode",  
4087 the provider MUST choose a type of execution for the requested operation.  
4088 See the section titled "[Determining execution mode](#)".

4089 A provider SHOULD execute an updates operation synchronously if it is possible to do so. (The  
4090 reason for this is that the result of an updates should reflect the set of changes currently recorded  
4091 for each matching object. Other operations are more likely to intervene if an updates operation is  
4092 executed asynchronously.)

4093 **Response.** The provider MUST return to the requestor a <updatesResponse>.

4094 **Status.** The <updatesResponse> must contain a "status" attribute that indicates whether the  
4095 provider successfully selected every object that matched the specified query.  
4096 See the section titled "[Status \(normative\)](#)" for values of this attribute.

- 4097 • If the provider successfully returned every update that occurred (since any time specified by  
4098 "updatedSince") to every object that matched the specified <query>  
4099 up to any limit specified by the value of the "maxSelect" attribute,  
4100 then the <updatesResponse> MUST specify "status='success'".
- 4101 • If the provider encountered an error in selecting any object that matched the specified <query>  
4102 or (if the provider encountered an error) in returning any of the selected updates, then the  
4103 <updatesResponse> MUST specify "status='failure'".

4104 **Update.** The <updatesResponse> MAY contain any number of <update> elements.

- 4105 • If the <updatesResponse> specifies "status='success'" and *at least one update matched*  
4106 the specified criteria, then the <updatesResponse> MUST contain at least one <update>  
4107 element that describes a change to a matching object.
- 4108 • If the <updatesResponse> specifies "status='success'" and *no object matched the*  
4109 specified criteria, then the <updatesResponse> MUST NOT contain an <update> element.
- 4110 • If the <updatesResponse> specifies "status='failure'", then the <updatesResponse>  
4111 MUST NOT contain an <update> element.

4112 **Update Psoid.** Each <update> MUST contain exactly one <psoid> element. Each <psoid>  
4113 element uniquely identifies the object that was changed.

4114 **Update timestamp.** Each <update> must have a "timestamp" attribute that specifies when the  
4115 object was changed.

- 4116 Any "timestamp" value MUST be expressed in UTC form, with no time zone component.  
 4117 A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.
- 4118 **Update updateKind.** Each <update> must have an "updateKind" attribute that describes how  
 4119 the object was changed.
- 4120 • If the <update> specifies "updateKind='add'", then the object was added.
  - 4121 • If the <update> specifies "updateKind='modify'",  
 4122 then the (schema-defined XML data that represents the) object was modified.
  - 4123 • If the <update> specifies "updateKind='delete'", then the object was deleted.
  - 4124 • If the <update> specifies "updateKind='capability'",  
 4125 then a set of capability-specific data that is (or was) associated with the object was modified.
- 4126 **Update wasUpdatedByCapability.** Each <update> MAY have a "wasUpdatedByCapability"  
 4127 attribute that identifies the capability for which data (specific to that capability and associated with  
 4128 the object) was changed.
- 4129 • An <update> that specifies "updateKind='capability'"  
 4130 MUST have a "wasUpdatedByCapability" attribute.
  - 4131 • An <update> that specifies "updateKind='add'" or (that specifies)  
 4132 "updateKind='modify'" or (that specifies) "updateKind='delete'"  
 4133 MUST NOT have a "wasUpdatedByCapability" attribute.
  - 4134 • The value of each "wasUpdatedByCapability" MUST be the URN of an XML namespace  
 4135 that uniquely identifies a capability. Each "wasUpdatedByCapability" attribute MUST  
 4136 identify a capability that the target supports.
- 4137 **iterator.** A <updatesResponse> MAY contain at most one <iterator> element.
- 4138 • If the <updatesResponse> specifies "status='success'" and the updates response  
 4139 *contains all of the objects* that matched the specified <query>, then the  
 4140 <updatesResponse> MUST NOT contain an <iterator>.
  - 4141 • If the <updatesResponse> specifies "status='success'" and the updates response  
 4142 *contains some but not all of the objects* that matched the specified <query>, then the  
 4143 <updatesResponse> MUST contain exactly one <iterator>.
  - 4144 • If the <updatesResponse> specifies "status='success'" and *no object matched* the  
 4145 specified <query>, then the <updatesResponse> MUST NOT contain an <iterator>.
  - 4146 • If the <updatesResponse> specifies "status='failure'", then the <updatesResponse>  
 4147 MUST NOT contain an <iterator>.
- 4148 **iterator ID.** An <iterator> MUST have an "ID" attribute.
- 4149 The value of the "ID" attribute uniquely identifies the <iterator> within the namespace of the  
 4150 provider. The "ID" attribute allows the provider to map each <iterator> token to the result set of  
 4151 the requestor's <query> and to any state that records the requestor's position within that result set.
- 4152 The "ID" attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an  
 4153 <iterator>. An <iterator> is not a PSO.

4154 **token.** An <updatesResponse> MAY have a "token" attribute. (The requestor may pass this  
4155 "token" value in the next <updatesRequest> for the same target. See the topic named "token"  
4156 within the section titled "[UpdatesRequest](#)" above.)

4157 **Error.** If the <updatesResponse> specifies "status='failure'", then the  
4158 <updatesResponse> MUST have an "error" attribute that characterizes the failure.  
4159 See the general section titled "[Error \(normative\)](#)".

4160 The section titled "[SearchQueryType Errors \(normative\)](#)" describes errors specific to a request that  
4161 contains a <query>. Also see the section titled "[SelectionType Errors \(normative\)](#)".  
4162 In addition, the <updatesResponse> MUST specify an appropriate value of "error" if any of the  
4163 following is true:

- 4164 • If the *number of updates that matched* the criteria that were specified in an  
4165 <updatesRequest> *exceeds any limit on the part of the provider.* (but does not exceed any  
4166 value of "maxSelect" that the requestor specified as part of the <query>).  
4167 In this case, the provider's <updatesResponse> SHOULD specify  
4168 "error='resultSetTooLarge'".

### 4169 [3.6.9.1.3 updates Examples \(non-normative\)](#)

4170 In the following example, a requestor asks a provider to updates for every Person with an email  
4171 address matching "joebob@example.com". The requestor includes no <updatedByCapability>  
4172 element, which indicates that only updates to the schema-defined data for each matching object  
4173 interest the requestor.

```
<updatesRequest requestID="145">  
  <query scope="subTree" targetID="target2" >  
    <select path="/Person/email="joebob@example.com"  
namespaceURI="http://www.w3.org/TR/xpath20" />  
  </query>  
</updatesRequest>
```

4174 The provider returns a <updatesResponse>. The "status" attribute of the  
4175 <updatesResponse> indicates that the provider successfully executed the updates operation.

```
<updatesResponse requestID="145" status="success">  
  <update timestamp="20050704115900" updateKind="modify">  
    <psoid ID="2244" targetID="target2"/>  
  </update>  
</updatesResponse>
```

4176 The requestor next asks the provider to include capability-specific updates (i.e., recorded changes  
4177 to capability-specific data items that are associated with each matching object). The requestor  
4178 indicates interest in updates specific to the reference capability and (indicates interest in updates  
4179 specific to the) the Suspend Capability.

```
<updatesRequest requestID="146">  
  <query scope="subTree" targetID="target2" >  
    <select path="/Person/email="joebob@example.com"  
namespaceURI="http://www.w3.org/TR/xpath20" />  
  </query>  
  <updatedByCapability>urn:oasis:names:tc:SPML:2.0:reference</updatedByCapability>  
  <updatedByCapability>urn:oasis:names:tc:SPML:2.0:suspend</updatedByCapability>  
</updatesRequest>
```

4180 The provider returns a <updatesResponse>. The "status" attribute of the  
4181 <updatesResponse> indicates that the provider successfully executed the updates operation.

```

<updatesResponse requestID="146" status="success">
  <update timestamp="20050704115911" updateKind="modify">
    <psolD ID="2244" targetID="target2"/>
  </update>
  <update timestamp="20050704115923" updateKind="capability"
wasUpdatedByCapability="urn:oasis:names:tc:SPML:2.0:reference">
    <psolD ID="2244" targetID="target2"/>
  </update>
</updatesResponse>

```

4182 This time the provider's response contains two updates: the "modify" update from the original  
4183 response plus a second "capability" update that is specific to the Reference Capability.

### 4184 **3.6.9.2 iterate**

4185 The iterate operation obtains the next set of objects from the result set that the provider selected for  
4186 a updates operation. (See the description of the [updates operation](#) above.)

4187 The subset of the Updates Capability XSD that is most relevant to the iterate operation follows.

```

<simpleType name="UpdateKindType">
  <restriction base="string">
    <enumeration value="add"/>
    <enumeration value="modify"/>
    <enumeration value="delete"/>
    <enumeration value="capability"/>
  </restriction>
</simpleType>

<complexType name="UpdateType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="psolD" type="spml:PSOIdentifierType" />
      </sequence>
      <attribute name="timestamp" type="xsd:dateTime"
use="required"/>
      <attribute name="updateKind"
type="spmlupdates:UpdateKindType" use="required"/>
      <attribute name="wasUpdatedByCapability" type="xsd:string"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="ResultsIteratorType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="ID" type="xsd:ID"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="UpdatesResponseType">
  <complexContent>
    <extension base="spml:ResponseType">

```

```

        <sequence>
            <element name="update" type="spmlupdates:UpdateType"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="iterator"
type="spmlupdates:ResultsIteratorType" minOccurs="0"/>
        </sequence>
        <attribute name="token" type="xsd:string" use="optional"/>
    </extension>
</complexContent>
</complexType>

<complexType name="IterateRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="iterator"
type="spmlupdates:ResultsIteratorType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

    <element name="iterateRequest" type="spmlupdates:IterateRequestType"/>
    <element name="iterateResponse"
type="spmlupdates:UpdatesResponseType"/>

```

4188 **An iterateRequest receives an iterateResponse.** A requestor supplies as input to an  
4189 <iterateRequest> the <iterator> that was part of the original <updatesResponse> or the  
4190 <iterator> that was part of a subsequent <iterateResponse>, whichever is most recent. A  
4191 provider returns an <iterateResponse> in response to each <iterateRequest>. An  
4192 <iterateResponse> has the same structure as a <updatesResponse>.

4193 The <iterateResponse> will contain at least one <update> element that records a change to  
4194 an object. If more matching updates are available to return, then the <iterateResponse> will  
4195 also contain an <iterator>. The requestor can use this <iterator> in another  
4196 <iterateRequest> to retrieve more of the matching objects.

4197 **Iterate is not batchable.** For reasons of scale, neither an updates request nor an iterate request  
4198 should be nested in a [batch](#) request. When an updates query matches more updates than the  
4199 provider can place directly in the response, the provider must temporarily store the remaining  
4200 updates. Storing the remaining updates allows the requestor to iterate the remaining updates, but  
4201 also requires the provider to commit resources.  
4202 See the topic named "Resource Considerations" earlier in this [Updates Capability](#) section.

4203 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
4204 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
4205 updates result sets. Allowing a requestor to nest a updates request or an iterate request within a  
4206 batch request would aggravate the resource problem, requiring a provider to store more information  
4207 in larger chunks for a longer amount of time.

4208 **The iterate operation must be executed synchronously.** The provider is already queuing the  
4209 result set (every update beyond those returned in the first updates response), so it is unreasonable  
4210 for a requestor to ask the provider to queue the results of a request for the next item in the result  
4211 set.

4212 Furthermore, asynchronous iteration would complicate the provider's maintenance of the result set.  
4213 Since a provider could never know that the requestor had processed the results of an  
4214 asynchronous iteration, the provider would not know when to increment its position in the result set.  
4215 In order to support asynchronous iteration both correctly and generally, a provider would have to  
4216 maintain a version of every result set for each iteration of that result set. This would impose an  
4217 unreasonable burden on the provider.

#### 4218 **3.6.9.2.1 *iterateRequest (normative)***

4219 A requestor MUST send an <iterateRequest> to a provider in order to obtain any *additional*  
4220 objects that matched a previous <updatesRequest> but that the provider has not yet returned to  
4221 the requestor. (That is, matching objects that were not contained in the response to that  
4222 <updatesRequest> and that have not yet been contained in any response to an  
4223 <iterateRequest> associated with that <updatesRequest>.)

4224 **Execution.** An <iterateRequest> MUST NOT specify "executionMode='asynchronous'".  
4225 An <iterateRequest> MUST specify "executionMode='synchronous' " or (an  
4226 <iterateRequest> MUST) omit "executionMode".  
4227 See the section titled "[Determining execution mode](#)".

4228 **iterator.** An <iterateRequest> MUST contain exactly one <iterator> element. A requestor  
4229 MUST supply as input to an <iterateRequest> the <iterator> from the original  
4230 <searchResponse> or (the requestor MUST supply as input to the <iterateRequest>) the  
4231 <iterator> from a subsequent <iterateResponse>. A requestor SHOULD supply as input  
4232 to an <iterateRequest> the most recent <iterator> that represents the updates result set.

#### 4233 **3.6.9.2.2 *iterateResponse (normative)***

4234 A provider that receives a <iterateRequest> from a requestor that the provider trusts must  
4235 examine the content of the <iterateRequest>. If the request is valid, the provider MUST return  
4236 (the XML that represents) the next object in the result set that the <iterator> represents.

4237 **Execution.** The provider MUST execute the iterate operation synchronously (if the provider  
4238 executes the iterate operation at all). See the section titled "[Determining execution mode](#)".

4239 **Response.** The provider MUST return to the requestor an <iterateResponse>.

4240 **Status.** The <iterateResponse> must contain a "status" attribute that indicates whether the  
4241 provider successfully returned the next update from the result set that the <iterator> represents.  
4242 See the section titled "[Status \(normative\)](#)".

4243 • If the provider successfully returned (the XML that represents) the next update from the result  
4244 set that the <iterator> represents, then the <iterateResponse> MUST specify  
4245 "status='success'".

4246 • If the provider encountered an error in returning (the XML that represents) the next update from  
4247 the result set that the <iterator> represents, then the <iterateResponse> MUST specify  
4248 "status='failure'".

4249 **Update.** The <iterateResponse> MAY contain any number of <update> elements.

4250 • If the <iterateResponse> specifies "status='success'" and *at least one update*  
4251 *remained to iterate* (in the updates result set that the <iterator> represents), then the  
4252 <iterateResponse> MUST contain at least one <update> element that records a change to  
4253 an object.

- 4254 • If the `<iterateResponse>` specifies “status=' success' ” and *no update remained to*  
4255 *iterate* (in the updates result set that the `<iterator>` represents), then the  
4256 `<iterateResponse>` MUST NOT contain an `<update>` element.
- 4257 • If the `<iterateResponse>` specifies “status=' failure' ”, then the `<iterateResponse>`  
4258 MUST NOT contain an `<update>` element.
- 4259 **iterator.** A `<iterateResponse>` to an `<iterateRequest>` MAY contain at most one  
4260 `<iterator>` element.
- 4261 • If the `<iterateResponse>` specifies “status=' success' ” and the `<iterateResponse>`  
4262 *contains the last of the updates* that matched the criteria that the original `<updatesRequest>`  
4263 specified, then the `<updatesResponse>` MUST NOT contain an `<iterator>`.
- 4264 • If the `<iterateResponse>` specifies “status=' success' ” and the provider *still has more*  
4265 *matching updates* that have not yet been returned to the requestor, then the  
4266 `<iterateResponse>` MUST contain exactly one `<iterator>`.
- 4267 • If the `<iterateResponse>` specifies “status=' failure' ”, then the `<iterateResponse>`  
4268 MUST NOT contain an `<iterator>`.
- 4269 **iterator ID.** An `<iterator>` MUST have an “ID” attribute.
- 4270 The value of the “ID” attribute uniquely identifies the `<iterator>` within the namespace of the  
4271 provider. The “ID” attribute allows the provider to map each `<iterator>` token to the result set of  
4272 the requestor’s `<query>` and to any state that records the requestor’s position within that result set.
- 4273 The “ID” attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an  
4274 `<iterator>`. An `<iterator>` is not a PSO.
- 4275 **Error.** If the `<iterateResponse>` specifies “status=' failure' ”, then the  
4276 `<iterateResponse>` MUST have an “error” attribute that characterizes the failure.  
4277 See the general section titled “[Error \(normative\)](#)”.
- 4278 In addition, the `<iterateResponse>` MUST specify an appropriate value of “error” if any of the  
4279 following is true:
- 4280 • The provider does not recognize the `<iterator>` in an `<iterateRequest>` as representing  
4281 an updates result set.
- 4282 • The provider does not recognize the `<iterator>` in an `<iterateRequest>` as representing  
4283 any updates result set that the provider currently maintains.
- 4284 The `<iterateResponse>` MAY specify an appropriate value of “error” if any of the following is  
4285 true:
- 4286 • An `<iterateRequest>` contains an `<iterator>` that is *not the most recent version* of the  
4287 `<iterator>`. If the provider has returned to the requestor a more recent `<iterator>` that  
4288 represents the same updates result set, then the provider MAY reject the older `<iterator>`.  
4289 (A provider that changes the ID—for example, to encode the state of iteration within an updates  
4290 result set—may be sensitive to this.)

4291 **3.6.9.2.3** *iterate Examples (non-normative)*

4292 In order to illustrate the iterate operation, we first need an updates operation that returns more than  
4293 one update. In the following example, a requestor asks a provider to return updates for every  
4294 `Person` with an email address that starts with the letter "j".

```
<updatesRequest requestID="152">  
  <query scope="subTree" targetID="target2" >  
    <select path='/Person/email="j*"' namespaceURI="http://www.w3.org/TR/xpath20" />  
  </query>  
</updatesRequest>
```

4295 The provider returns a `<updatesResponse>`. The "status" attribute of the  
4296 `<updatesResponse>` indicates that the provider successfully executed the updates operation.  
4297 The `<updatesResponse>` contains two `<update>` elements that represent the first matching  
4298 updates.

```
<updatesResponse requestID="152" status="success">  
  <update timestamp="1944062400000000" updateKind="add">  
    <psolD ID="0001" targetID="target2"/>  
  </update>  
  <update timestamp="1942092700000000" updateKind="add">  
    <psolD ID="0002" targetID="target2"/>  
  </update>  
  <update timestamp="1970091800000000" updateKind="delete">  
    <psolD ID="0002" targetID="target2"/>  
  </update>  
  <iterator ID="1970"/>  
</updatesResponse>
```

4299 The requestor asks the provider to return the next set of matching updates (from the original result  
4300 set). The requestor supplies the `<iterator>` from the `<updatesResponse>` as input to the  
4301 `<iterateRequest>`.

```
<iterateRequest requestID="153">  
  <iterator ID="1970"/>  
</iterateRequest>
```

4302 The provider returns an `<iterateResponse>` in response to the `<iterateRequest>`. The  
4303 "status" attribute of the `<iterateResponse>` indicates that the provider successfully executed  
4304 the iterate operation. The `<iterateResponse>` contains two `<update>` elements that represent  
4305 the next matching updates.

```
<iterateResponse requestID="153" status="success">  
  <update timestamp="1948031200000000" updateKind="add">  
    <psolD ID="0003" targetID="target2"/>  
  </update>  
  <update timestamp="1969120900000000" updateKind="add">  
    <psolD ID="0004" targetID="target2"/>  
  </update>  
  <iterator ID="1971"/>  
</iterateResponse>
```

4306 The `<iterateResponse>` also contains another `<iterator>` element. The "ID" of this  
4307 `<iterator>` differs from the "ID" of the `<iterator>` in the original `<updatesResponse>`. The  
4308 "ID" could remain constant (for each iteration of the result set that the `<iterator>` represents) if  
4309 the provider so chooses, but the "ID" value could change (e.g., if the provider uses "ID" to  
4310 encode the state of the result set).

4311 To get the next set of matching updates, the requestor again supplies the `<iterator>` from the  
4312 `<iterateResponse>` as input to an `<iterateRequest>`.

```
<iterateRequest requestID="154">  
  <iterator ID="1971"/>  
</iterateRequest>
```

4313 The provider again returns an `<iterateResponse>` in response to the `<iterateRequest>`. The  
4314 "status" attribute of the `<iterateResponse>` indicates that the provider successfully executed  
4315 the iterate operation. The `<iterateResponse>` contains an `<update>` element that represents  
4316 the final matching object. Since all of the matching objects have now been returned to the  
4317 requestor, this `<iterateResponse>` contains no `<iterator>`.

```
<iterateResponse requestID="154" status="success">  
  <update timestamp="20050704115900" updateKind="modify">  
    <psolID ID="2244" targetID="target2"/>  
  </update>  
</iterateResponse>
```

4318

### 4319 **3.6.9.3 closeliterator**

4320 The closeliterator operation tells the provider that the requestor has no further need for the updates  
4321 result set that a specific `<iterator>` represents. (See the description of the [updates operation](#)  
4322 above.)

4323 A requestor should send a `<closeIteratorRequest>` to the provider when the requestor no  
4324 longer intends to iterate an updates result set. (A provider will eventually free an inactive updates  
4325 result set--even if the provider never receives a `<closeIteratorRequest>` from the requestor--  
4326 but this behavior is unspecified.) For more information, see the topic named "Resource  
4327 Considerations" topic earlier within this section.

4328 The subset of the Search Capability XSD that is most relevant to the iterate operation follows.

```

<complexType name="ResultsIteratorType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="ID" type="xsd:ID"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="CloseIteratorRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="iterator"
type="spmlupdates:ResultsIteratorType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="closeIteratorRequest"
type="spmlupdates:CloseIteratorRequestType"/>
<element name="closeIteratorResponse" type="spml:ResponseType"/>

```

4329 **A closeliteratorRequest receives a closeliteratorResponse.** A requestor supplies as input to a  
4330 <closeIteratorRequest> the <iterator> that was part of the original <updatesResponse>  
4331 or the <iterator> that was part of a subsequent <iterateResponse>, whichever is most  
4332 recent. A provider returns a <closeIteratorResponse> in response to each  
4333 <closeIteratorRequest>. A <closeIteratorResponse> has the same structure as an  
4334 <spml:response>.

4335 **closeliterator is not batchable.** For reasons of scale, neither an updates request nor an iterate  
4336 request nor a closeliterator request should be nested in a [batch](#) request. When an updates query  
4337 matches more updates than the provider can place directly in the response, the provider must  
4338 temporarily store the remaining updates. Storing the remaining updates allows the requestor to  
4339 iterate the remaining updates, but also requires the provider to commit resources.  
4340 See the topic named "Resource Considerations" earlier in this section.

4341 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
4342 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
4343 search results. Allowing a requestor to nest an updates request or an iterate request or a  
4344 closeliterator request within a batch request would aggravate the resource problem, requiring a  
4345 provider to store more information in larger chunks for a longer amount of time.

4346 **The closeliterator operation must be executed synchronously.** The provider is already queuing  
4347 the result set (every update beyond those returned in the first updates response), so a request to  
4348 close the iterator (and thus to free the system resources associated with the result set) should be  
4349 executed as soon as possible. It is unreasonable for a requestor to ask the provider to queue the  
4350 results of a request to close an iterator (especially since the close iterator response contains little or  
4351 no information beyond success or failure).

### 4352 [3.6.9.3.1](#) *closeIteratorRequest (normative)*

4353 A requestor SHOULD send a `<closeIteratorRequest>` to a provider when the requestor no  
4354 longer intends to iterate an updates result set. (This allows the provider to free any system  
4355 resources associated with the updates result set.)

4356 **Execution.** A `<closeIteratorRequest>` MUST NOT specify  
4357 "executionMode='asynchronous'".  
4358 A `<closeIteratorRequest>` MUST specify "executionMode='synchronous' "  
4359 or (a `<closeIteratorRequest>` MUST) omit "executionMode".  
4360 See the section titled "[Determining execution mode](#)".

4361 **iterator.** A `<closeIteratorRequest>` MUST contain exactly one `<iterator>` element. A  
4362 requestor MUST supply as input to a `<closeIteratorRequest>` the `<iterator>` from the  
4363 original `<updatesResponse>` or (a requestor MUST supply the `<iterator>`) from a subsequent  
4364 `<iterateResponse>`. A requestor SHOULD supply as input to a  
4365 `<closeIteratorRequest>` the most recent `<iterator>` that represents the updates result set.

4366 **iterator ID.** An `<iterator>` that is part of a `<closeIteratorRequest>` MUST have an "ID"  
4367 attribute. (The value of the "ID" attribute uniquely identifies the `<iterator>` within the  
4368 namespace of the provider. The "ID" attribute allows the provider to map each `<iterator>`  
4369 token to the result set of the requestor's `<query>` and also (allows the provider to map each  
4370 `<iterator>` token) to any state that records the requestor's iteration *within* that result set.)

### 4371 [3.6.9.3.2](#) *closeIteratorResponse (normative)*

4372 A provider that receives a `<closeIteratorRequest>` from a requestor that the provider trusts  
4373 must examine the content of the `<closeIteratorRequest>`. If the request is valid, the provider  
4374 MUST release any updates result set that the `<iterator>` represents. Any subsequent request to  
4375 iterate that same updates result set MUST fail.

4376 **Execution.** The provider MUST execute the `closeIterator` operation synchronously (if the provider  
4377 executes the `closeIterator` operation at all). See the section titled "[Determining execution mode](#)".

4378 **Response.** The provider MUST return to the requestor a `<closeIteratorResponse>`.

4379 **Status.** The `<closeIteratorResponse>` must contain a "status" attribute that indicates  
4380 whether the provider successfully released the updates result set that the `<iterator>` represents.  
4381 See the section titled "[Status \(normative\)](#)".

4382 • If the provider successfully released the updates result set that the `<iterator>` represents,  
4383 then the `<closeIteratorResponse>` MUST specify "status='success'".

4384 • If the provider encountered an error in releasing the updates result set that the `<iterator>`  
4385 represents, then the `<closeIteratorResponse>` MUST specify "status='failure'".

4386 **Error.** If the `<closeIteratorResponse>` specifies "status='failure'", then the  
4387 `<closeIteratorResponse>` MUST have an "error" attribute that characterizes the failure.  
4388 See the general section titled "[Error \(normative\)](#)".

4389 In addition, the `<closeIteratorResponse>` MUST specify an appropriate value of "error" if  
4390 any of the following is true:

4391 • If the provider does not recognize the `<iterator>` in a `<closeIteratorRequest>` as  
4392 representing an updates result set.

- 4393 • If the provider does not recognize the `<iterator>` in a `<closeIteratorRequest>` as  
4394 representing any updates result set that the provider currently maintains.
- 4395 • If the provider recognized the `<iterator>` in a `<closeIteratorRequest>` as representing  
4396 a updates result set that the provider currently maintains but *cannot release the resources*  
4397 *associated with that updates result set*.

4398 The `<closeIteratorResponse>` MAY specify an appropriate value of "error" if any of the  
4399 following is true:

- 4400 • If a `<closeIteratorRequest>` contains an `<iterator>` that is *not the most recent version*  
4401 *of the <iterator>*. If the provider has returned to the requestor a more recent `<iterator>`  
4402 that represents the same updates result set, then the provider MAY reject the older  
4403 `<iterator>`.  
4404 (A provider that changes the ID—for example, to encode the state of iteration within a updates  
4405 result set—may be sensitive to this.)

### 4406 3.6.9.3.3 *closeIterator Examples (non-normative)*

4407 In order to illustrate the iterate operation, we first need an updates operation that returns more than  
4408 one update. In the following example, a requestor asks a provider to return updates for every  
4409 `Person` with an email address that starts with the letter "j".

```
<updatesRequest requestID="152">
  <query scope="subTree" targetID="target2" >
    <select path="/Person/email="j*" namespaceURI="http://www.w3.org/TR/xpath20" />
  </query>
</updatesRequest>
```

4410 The provider returns a `<updatesResponse>`. The "status" attribute of the  
4411 `<updatesResponse>` indicates that the provider successfully executed the updates operation.  
4412 The `<updatesResponse>` contains two `<update>` elements that represent the first matching  
4413 updates.

```
<updatesResponse requestID="152" status="success">
  <update timestamp="1944062400000000" updateKind="add">
    <psolD ID="0001" targetID="target2"/>
  </update>
  <update timestamp="1942092700000000" updateKind="add">
    <psolD ID="0002" targetID="target2"/>
  </update>
  <update timestamp="1970091800000000" updateKind="delete">
    <psolD ID="0002" targetID="target2"/>
  </update>
  <iterator ID="1970"/>
</updatesResponse>
```

4414 The requestor decides that the two objects in the initial `<searchResponse>` will suffice, and does  
4415 not intend to retrieve any more matching objects (in the result set for the search). The requestor  
4416 supplies the `<iterator>` from the `<updatesResponse>` as input to the  
4417 `<closeIteratorRequest>`.

```
<closeIteratorRequest requestID="153">
  <iterator ID="1900"/>
</closeIteratorRequest>
```

4418 The provider returns a `<closeIteratorResponse>` in response to the  
4419 `<closeIteratorRequest>`. The "status" attribute of the `<closeIteratorResponse>`  
4420 indicates that the provider successfully released the result set.

```
<closeIteratorResponse requestID="153" status="success"/>
```

4421

### 4422 **3.7 Custom Capabilities**

4423 The features of SPMLv2 that allow the PSTC to define optional operations as part of standard  
4424 capabilities are *open mechanisms* that will work for anyone. An individual provider (or any third  
4425 party) can define a custom capability that integrates with SPMLv2. Whoever controls the  
4426 namespace of the capability controls the extent to which it can be shared. Each provider  
4427 determines which capabilities are supported for which types of objects on which types of targets.

4428 The SPMLv2 capability mechanism is extensible. Any party may define additional capabilities. A  
4429 provider declares its support for a custom capability in exactly the same way that it declares support  
4430 for a standard capability: as a target <capability> element.

4431 The standard capabilities that SPMLv2 defines will not address all needs. Contributors may define  
4432 additional custom capabilities.

4433 Since the schema for each capability is defined in a separate namespace, a custom capability will  
4434 not ordinarily conflict with a standard capability that is defined as part of SPMLv2, nor will a custom  
4435 capability ordinarily conflict with another custom capability. In order for a custom capability B to  
4436 conflict with another capability A, capability B would have to import the namespace of capability A  
4437 and re-declare a schema element from capability A. Such a conflict is clearly intentional and a  
4438 provider can easily avoid such a conflict by not declaring support for capability B.

4439 Also see the section titled "[Conformance](#)".

---

4440 **4 Conformance (normative)**

4441 **4.1 Core operations and schema are mandatory**

4442 A conformant provider **MUST** support the elements, attributes, and types defined in the SPMLv2  
4443 Core XSD. This includes all the core operations and protocol behavior.

4444 Schema syntax for the SPMLv2 core operations is defined in a schema that is associated with the  
4445 following XML namespace: `urn:oasis:names:tc:SPML:2:0`. This document includes the Core  
4446 XSD as Appendix A.

4447 **4.2 Standard capabilities are optional**

4448 A conformant provider **SHOULD** support the XML schema and operations defined by each standard  
4449 capability of SPMLv2.

4450 **4.3 Custom capabilities must not conflict**

4451 A conformant provider **MUST** use the custom capability mechanism of SPMLv2 to expose any  
4452 operation beyond those specified by the core and standard capabilities of SPMLv2.

4453 A conformant provider **MAY** support any custom capability that conforms to SPMLv2.

4454 **Must conform to standard schema.** Any operation that a custom capability defines **MUST** be  
4455 defined as a request-response pair such that all of the following are true:

- 4456 • The request type (directly or indirectly) extends `{RequestType}`  
4457 • The response type is `{ResponseType}` or (the response type directly or indirectly) extends  
4458 `{ResponseType}`.

4459 **Must not conflict with another capability.** Since each custom capability is defined in its own  
4460 namespace, an element or attribute in the XML schema that is associated with a *custom capability*  
4461 **SHOULD NOT conflict with** (i.e., **SHOULD NOT** redefine and **SHOULD NOT** otherwise change the  
4462 definition of) any element or attribute in any other namespace:

- 4463 • A custom capability **MUST NOT** conflict with the Core XSD of SPMLv2.  
4464 • A custom capability **MUST NOT** conflict with any standard capability of SPMLv2.  
4465 • A custom capability **SHOULD NOT** conflict with another custom capability.

4466 **Must not bypass standard capability.** A conformant provider **MUST NOT** expose an operation  
4467 that competes with (i.e., whose functions overlap those of) an operation defined by a standard  
4468 capability of SPMLv2) **UNLESS** all of the following are true:

- 4469 • The provider **MUST** *define the competing operation in a custom capability*.  
4470 • Every target (and every schema entity on a target) that supports the provider's custom  
4471 capability **MUST** also *support the standard capability*.

4472 **4.4 Capability Support is all-or-nothing**

4473 A provider that claims to support a particular capability for (a set of schema entities on) a target  
4474 MUST support (for every instance of those schema entities on the target) every operation that the  
4475 capability defines.

4476 **4.5 Capability-specific data**

4477 A capability MAY imply capability-specific data. That is, a capability MAY specify that data specific  
4478 to that capability may be associated with one or more objects. (For example, the Reference  
4479 Capability implies that each object may contain a set of references to other objects.)

4480 Any capability that implies capability-specific data MAY rely on the default processing that SPMLv2  
4481 specifies for capability-specific data (see the section titled “CapabilityData Processing (normative)”).  
4482 However, any capability that implies capability-specific data SHOULD specify the structure of that  
4483 data. (For example, the Reference Capability specifies that its capability-specific data must contain  
4484 at least one <reference> and should contain only <reference> elements.)

4485 Furthermore, any capability that implies capability-specific data and *for which the default processing*  
4486 *of capability-specific data is inappropriate* (i.e., any capability for which an instance of  
4487 {CapabilityDataType} that refers to the capability would specify “mustUnderstand=’true’”)

- 4488 • MUST specify the structure of that capability-specific data.
- 4489 • MUST specify how core operations should handle that capabilityData.  
4490 (For example, the Reference Capability specifies how each reference must be validated and  
4491 processed. See the section titled “Reference CapabilityData Processing (normative).)

---

4492

## 5 Security Considerations

4493

### 5.1 Use of SSL 3.0 or TLS 1.0

4494 When using Simple Object Access Protocol (SOAP) **[SOAP]** as the protocol for the [requestor](#)  
4495 (client) to make SPMLv2 requests to a [provider](#) (server), Secure Sockets Layer (SSL 3.0) or  
4496 Transport Layer Security (TLS 1.0) **[RFC 2246]** SHOULD be used.

4497 The TLS implementation SHOULD implement the TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA or the  
4498 TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA **[AES]** cipher suite.

4499

### 5.2 Authentication

4500 When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as  
4501 the SOAP **[SOAP]** transport protocol, the [provider](#) (server) SHOULD be authenticated to the  
4502 [requestor](#) (client) using X.509 v3 **[X509]** service certificates. The [requestor](#) (client) SHOULD be  
4503 authenticated to the [provider](#) (server) using X.509 v3 service certificates.

4504 For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require  
4505 intermediaries, Web Services Security **[WSS]** SHOULD be used for authentication.

4506

### 5.3 Message Integrity

4507 When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as  
4508 the SOAP **[SOAP]** transport protocol, message integrity is reasonably assured for point-to-point  
4509 message exchanges.

4510 For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require  
4511 intermediaries, Web Services Security **[WSS]** SHOULD be used to ensure message integrity.

4512

### 5.4 Message Confidentiality

4513 When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as  
4514 the SOAP **[SOAP]** transport protocol, message confidentiality is reasonably assured for point-to-  
4515 point message exchanges, and for the entire message.

4516 For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require  
4517 intermediaries, Web Services Security **[WSS]** SHOULD be used to ensure confidentiality for the  
4518 sensitive portions of the message.

## Appendix A. Core XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_SPMLv2_core_27.xsd -->
<!-- -->
<!-- Draft schema for SPML v2.0 core capabilities. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:spml="urn:oasis:names:tc:SPML:2:0" elementFormDefault="qualified">

  <complexType name="ExtensibleType">
    <sequence>
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded"
processContents="lax"/>
    </sequence>
    <anyAttribute namespace="##other" processContents="lax"/>
  </complexType>

  <simpleType name="ExecutionModeType">
    <restriction base="string">
      <enumeration value="synchronous"/>
      <enumeration value="asynchronous"/>
    </restriction>
  </simpleType>

  <complexType name="CapabilityDataType">
    <complexContent>
      <extension base="spml:ExtensibleType">
        <annotation>
          <documentation>Contains elements specific to a
capability.</documentation>
        </annotation>
        <attribute name="mustUnderstand" type="boolean"
use="optional"/>
        <attribute name="capabilityURI" type="anyURI"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="RequestType">
    <complexContent>
      <extension base="spml:ExtensibleType">
        <attribute name="requestID" type="xsd:ID" use="optional"/>
        <attribute name="executionMode" type="spml:ExecutionModeType"

```

```

use="optional"/>
  </extension>
</complexContent>
</complexType>

<simpleType name="StatusCodeType">
  <restriction base="string">
    <enumeration value="success"/>
    <enumeration value="failure"/>
    <enumeration value="pending"/>
  </restriction>
</simpleType>

<simpleType name="ErrorCode">
  <restriction base="string">
    <enumeration value="malformedRequest"/>
    <enumeration value="unsupportedOperation"/>
    <enumeration value="unsupportedIdentifierType"/>
    <enumeration value="noSuchIdentifier"/>
    <enumeration value="customError"/>
    <enumeration value="unsupportedExecutionMode"/>
    <enumeration value="invalidContainment"/>
    <enumeration value="noSuchRequest"/>
    <enumeration value="unsupportedSelectionType"/>
    <enumeration value="resultSetTooLarge"/>
    <enumeration value="unsupportedProfile"/>
    <enumeration value="invalidIdentifier"/>
    <enumeration value="alreadyExists"/>
    <enumeration value="containerNotEmpty"/>
  </restriction>
</simpleType>

<simpleType name="ReturnDataType">
  <restriction base="string">
    <enumeration value="identifier"/>
    <enumeration value="data"/>
    <enumeration value="everything"/>
  </restriction>
</simpleType>

<complexType name="ResponseType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="errorMessage" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="status" type="spml:StatusCodeType"
use="required"/>
      <attribute name="requestID" type="xsd:ID" use="optional"/>
      <attribute name="error" type="spml:ErrorCode"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

```

```

<complexType name="IdentifierType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="ID" type="string" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="PSOIdentifierType">
  <complexContent>
    <extension base="spml:IdentifierType">
      <sequence>
        <element name="containerID" type="spml:PSOIdentifierType"
minOccurs="0"/>
      </sequence>
      <attribute name="targetID" type="string" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="PSOType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType"/>
        <element name="data" type="spml:ExtensibleType"
minOccurs="0"/>
        <element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="AddRequestType">
  <complexContent>
    <extension base="spml:RequestType">
      <sequence>
        <element name="psoID" type="spml:PSOIdentifierType"
minOccurs="0" />
        <element name="containerID" type="spml:PSOIdentifierType"
minOccurs="0" />
        <element name="data" type="spml:ExtensibleType"/>
        <element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded" />
      </sequence>
      <attribute name="targetID" type="string" use="optional"/>
      <attribute name="returnData" type="spml:ReturnDataType"
use="optional" default="everything"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="AddResponseType">
  <complexContent>
    <extension base="spml:ResponseType">

```

```

        <sequence>
            <element name="pso" type="spml:PSOType" minOccurs="0"/>
        </sequence>
    </extension>
</complexContent>
</complexType>

<simpleType name="ModificationModeType">
    <restriction base="string">
        <enumeration value="add"/>
        <enumeration value="replace"/>
        <enumeration value="delete"/>
    </restriction>
</simpleType>

<complexType name="NamespacePrefixMappingType">
    <complexContent>
        <extension base="spml:ExtensibleType">
            <attribute name="prefix" type="string" use="required"/>
            <attribute name="namespace" type="string" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="QueryClauseType">
    <complexContent>
        <extension base="spml:ExtensibleType">
        </extension>
    </complexContent>
</complexType>

<complexType name="SelectionType">
    <complexContent>
        <extension base="spml:QueryClauseType">
            <sequence>
                <element name="namespacePrefixMap"
type="spml:NamespacePrefixMappingType" minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
            <attribute name="path" type="string" use="required"/>
            <attribute name="namespaceURI" type="string" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="ModificationType">
    <complexContent>
        <extension base="spml:ExtensibleType">
            <sequence>
                <element name="component" type="spml:SelectionType"
minOccurs="0"/>
                <element name="data" type="spml:ExtensibleType"
minOccurs="0"/>
                <element name="capabilityData"
type="spml:CapabilityDataType" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

```

```

        <attribute name="modificationMode"
type="spml:ModificationModeType" use="optional"/>
    </extension>
</complexContent>
</complexType>

<complexType name="ModifyRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="psoID" type="spml:PSOIdentifierType"/>
                <element name="modification" type="spml:ModificationType"
maxOccurs="unbounded"/>
            </sequence>
            <attribute name="returnData" type="spml:ReturnDataType"
use="optional" default="everything"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="ModifyResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <sequence>
                <element name="pso" type="spml:PSOType" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="DeleteRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="psoID" type="spml:PSOIdentifierType"/>
            </sequence>
            <attribute name="recursive" type="xsd:boolean" use="optional"
default="false"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="LookupRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="psoID" type="spml:PSOIdentifierType"/>
            </sequence>
            <attribute name="returnData" type="spml:ReturnDataType"
use="optional" default="everything"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="LookupResponseType">
    <complexContent>

```

```

    <extension base="spml:ResponseType">
      <sequence>
        <element name="pso" type="spml:PSOType" minOccurs="0" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="SchemaType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <annotation>
          <documentation>Profile specific schema elements should
be included here</documentation>
        </annotation>
        <element name="supportedSchemaEntity"
type="spml:SchemaEntityRefType" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="ref" type="anyURI" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="SchemaEntityRefType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <attribute name="targetID" type="string" use="optional"/>
      <attribute name="entityName" type="string" use="optional"/>
      <attribute name="isContainer" type="xsd:boolean"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="CapabilityType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="appliesTo" type="spml:SchemaEntityRefType"
minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="namespaceURI" type="anyURI"/>
      <attribute name="location" type="anyURI" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="CapabilitiesListType">
  <complexContent>
    <extension base="spml:ExtensibleType">
      <sequence>
        <element name="capability" type="spml:CapabilityType"
minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

```

    </complexContent>
  </complexType>

  <complexType name="TargetType">
    <complexContent>
      <extension base="spml:ExtensibleType">
        <sequence>
          <element name="schema" type="spml:SchemaType"
maxOccurs="unbounded"/>
          <element name="capabilities"
type="spml:CapabilitiesListType" minOccurs="0"/>
        </sequence>
        <attribute name="targetID" type="string" use="optional"/>
        <attribute name="profile" type="anyURI" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ListTargetsRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        </extension>
        <attribute name="profile" type="anyURI" use="optional"/>
      </complexContent>
    </complexType>

  <complexType name="ListTargetsResponseType">
    <complexContent>
      <extension base="spml:ResponseType">
        <sequence>
          <element name="target" type="spml:TargetType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <element name="select" type="spml:SelectionType"/>
  <element name="addRequest" type="spml:AddRequestType"/>
  <element name="addResponse" type="spml:AddResponseType"/>
  <element name="modifyRequest" type="spml:ModifyRequestType"/>
  <element name="modifyResponse" type="spml:ModifyResponseType"/>
  <element name="deleteRequest" type="spml>DeleteRequestType"/>
  <element name="deleteResponse" type="spml:ResponseType"/>
  <element name="lookupRequest" type="spml:LookupRequestType"/>
  <element name="lookupResponse" type="spml:LookupResponseType"/>
  <element name="listTargetsRequest"
type="spml>ListTargetsRequestType"/>
  <element name="listTargetsResponse"
type="spml>ListTargetsResponseType"/>

</schema>

```

## Appendix A. Async Capability XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_SPMLv2_aync_27.xsd -->
<!-- Draft schema for SPML v2.0 asynchronous capabilities. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0:async"
  xmlns:spml="urn:oasis:names:tc:SPML:2:0"
  xmlns:spmlasync="urn:oasis:names:tc:SPML:2:0:async"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <import namespace="urn:oasis:names:tc:SPML:2:0"
    schemaLocation="draft_pstc_SPMLv2_core_27.xsd"/>

  <complexType name="CancelRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <attribute name="asyncRequestID" type="xsd:string"
use="required"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="CancelResponseType">
    <complexContent>
      <extension base="spml:ResponseType">
        <attribute name="asyncRequestID" type="xsd:string"
use="required"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="StatusRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <attribute name="returnResults" type="xsd:boolean"
use="optional" default="false"/>
        <attribute name="asyncRequestID" type="xsd:string"
use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="StatusResponseType">

```

```
<complexContent>
  <extension base="spml:ResponseType">
    <attribute name="asyncRequestID" type="xsd:string"
use="optional"/>
  </extension>
</complexContent>
</complexType>

<element name="cancelRequest" type="spmlasync:CancelRequestType"/>
<element name="cancelResponse" type="spmlasync:CancelResponseType"/>
<element name="statusRequest" type="spmlasync:StatusRequestType"/>
<element name="statusResponse" type="spmlasync:StatusResponseType"/>

</schema>
```

4522

## Appendix B. Batch Capability XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_SPMLv2_batch_27.xsd -->
<!-- -->
<!-- Draft schema for SPML v2.0 batch request capability. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0:batch"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:spml="urn:oasis:names:tc:SPML:2:0"
  xmlns:spmlbatch="urn:oasis:names:tc:SPML:2:0:batch"
  elementFormDefault="qualified">

  <import namespace='urn:oasis:names:tc:SPML:2:0'
    schemaLocation='draft_pstc_SPMLv2_core_27.xsd' />

  <simpleType name="ProcessingType">
    <restriction base="string">
      <enumeration value="sequential"/>
      <enumeration value="parallel"/>
    </restriction>
  </simpleType>

  <simpleType name="OnErrorType">
    <restriction base="string">
      <enumeration value="resume"/>
      <enumeration value="exit"/>
    </restriction>
  </simpleType>

  <complexType name="BatchRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <annotation>
          <documentation>Elements that extend spml:RequestType
</documentation>
        </annotation>
        <attribute name="processing" type="spmlbatch:ProcessingType"
use="optional" default="sequential"/>
        <attribute name="onError" type="spmlbatch:OnErrorType"
use="optional" default="exit"/>
      </extension>
    </complexContent>
  </complexType>

```

```
<complexType name="BatchResponseType">
  <complexContent>
    <extension base="spml:ResponseType">
      <annotation>
        <documentation>Elements that extend spml:ResponseType
</documentation>
      </annotation>
    </extension>
  </complexContent>
</complexType>

  <element name="batchRequest" type="spmlbatch:BatchRequestType"/>
  <element name="batchResponse" type="spmlbatch:BatchResponseType"/>

</schema>
```

4524

## Appendix C. Bulk Capability XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_SPMLv2_bulk_27.xsd -->
<!-- -->
<!-- Draft schema for SPML v2.0 bulk operation capabilities. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0:bulk"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:spml="urn:oasis:names:tc:SPML:2:0"
  xmlns:spmlsearch="urn:oasis:names:tc:SPML:2:0:search"
  xmlns:spmlbulk="urn:oasis:names:tc:SPML:2:0:bulk"
  elementFormDefault="qualified">

  <import namespace='urn:oasis:names:tc:SPML:2:0'
    schemaLocation='draft_pstc_SPMLv2_core_27.xsd' />

  <import namespace='urn:oasis:names:tc:SPML:2:0:search'
    schemaLocation='draft_pstc_SPMLv2_search_27.xsd' />

  <complexType name="BulkModifyRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element ref="spmlsearch:query"/>
          <element name="modification" type="spml:ModificationType"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="BulkDeleteRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element ref="spmlsearch:query"/>
        </sequence>
        <attribute name="recursive" type="boolean" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="bulkModifyRequest"
type="spmlbulk:BulkModifyRequestType"/>

```

```
<element name="bulkModifyResponse" type="spml:ResponseType"/>

  <element name="bulkDeleteRequest"
type="spmlbulk:BulkDeleteRequestType"/>
  <element name="bulkDeleteResponse" type="spml:ResponseType"/>

</schema>
```

4526

## Appendix D. Password Capability XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_SPMLv2_password_27.xsd -->
<!-- -->
<!-- Draft schema for SPML v2.0 password capabilities. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0:password"
  xmlns:pass="urn:oasis:names:tc:SPML:2:0:password"
  xmlns:spml="urn:oasis:names:tc:SPML:2:0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <import namespace="urn:oasis:names:tc:SPML:2:0"
    schemaLocation="draft_pstc_SPMLv2_core_27.xsd"/>

  <complexType name="SetPasswordRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="psoID" type="spml:PSOIdentifierType"/>
          <element name="password" type="string"/>
          <element name="currentPassword" type="string"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ExpirePasswordRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="psoID" type="spml:PSOIdentifierType"/>
        </sequence>
        <attribute name="remainingLogins" type="int" use="optional"
default="1"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ResetPasswordRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="psoID" type="spml:PSOIdentifierType"/>

```

```

        </sequence>
    </extension>
</complexContent>
</complexType>

<complexType name="ResetPasswordResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <sequence>
                <element name="password" type="string" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ValidatePasswordRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="psoID" type="spml:PSOIdentifierType"/>
                <element name="password" type="string"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ValidatePasswordResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <attribute name="valid" type="boolean" use="optional"/>
        </extension>
    </complexContent>
</complexType>

    <element name="setPasswordRequest"
type="pass:SetPasswordRequestType"/>
    <element name="setPasswordResponse" type="spml:ResponseType"/>
    <element name="expirePasswordRequest"
type="pass:ExpirePasswordRequestType"/>
    <element name="expirePasswordResponse" type="spml:ResponseType"/>
    <element name="resetPasswordRequest"
type="pass:ResetPasswordRequestType"/>
    <element name="resetPasswordResponse"
type="pass:ResetPasswordResponseType"/>
    <element name="validatePasswordRequest"
type="pass:ValidatePasswordRequestType"/>
    <element name="validatePasswordResponse"
type="pass:ValidatePasswordResponseType"/>

</schema>

```

## Appendix E. Reference Capability XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_SPMLv2_reference_27.xsd -->
<!-- -->
<!-- Draft schema for SPML v2.0 reference capabilities. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0:reference"
  xmlns:ref="urn:oasis:names:tc:SPML:2:0:reference"
  xmlns:spml="urn:oasis:names:tc:SPML:2:0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <import namespace="urn:oasis:names:tc:SPML:2:0"
    schemaLocation="draft_pstc_SPMLv2_core_27.xsd"/>

  <complexType name="ReferenceType">
    <complexContent>
      <extension base="spml:ExtensibleType">
        <sequence>
          <element name="toPsoID" type="spml:PSOIdentifierType"
minOccurs="0"/>
          <element name="referenceData" type="spml:ExtensibleType"
minOccurs="0"/>
        </sequence>
        <attribute name="typeOfReference" type="string"
use="required"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ReferenceDefinitionType">
    <complexContent>
      <extension base="spml:ExtensibleType">
        <sequence>
          <element name="schemaEntity"
type="spml:SchemaEntityRefType"/>
          <element name="canReferTo" type="spml:SchemaEntityRefType"
minOccurs="0" maxOccurs="unbounded"/>
          <element name="referenceDataType"
type="spml:SchemaEntityRefType" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="typeOfReference" type="string"
use="required"/>
      </extension>
    </complexContent>
  </complexType>

```

```

</complexType>

<complexType name="HasReferenceType">
  <complexContent>
    <extension base="spml:QueryClauseType">
      <sequence>
        <element name="toPsoID" type="spml:PSOIdentifierType"
minOccurs="0" />
        <element name="referenceData" type="spml:ExtensibleType"
minOccurs="0" />
      </sequence>
      <attribute name="typeOfReference" type="string"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

  <element name="hasReference" type="spmlref:HasReferenceType"/>
  <element name="reference" type="spmlref:ReferenceType"/>
  <element name="referenceDefinition"
type="spmlref:ReferenceDefinitionType"/>

</schema>

```

4530

## Appendix F. Search Capability XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_SPMLv2_search_27.xsd -->
<!-- -->
<!-- Draft schema for SPML v2.0 search capabilities. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0:search"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:spml="urn:oasis:names:tc:SPML:2:0"
  xmlns:spmlsearch="urn:oasis:names:tc:SPML:2:0:search"
  elementFormDefault="qualified">

  <import namespace='urn:oasis:names:tc:SPML:2:0'
    schemaLocation='draft_pstc_SPMLv2_core_27.xsd' />

  <simpleType name="ScopeType">
    <restriction base="string">
      <enumeration value="pso"/>
      <enumeration value="oneLevel"/>
      <enumeration value="subTree"/>
    </restriction>
  </simpleType>

  <complexType name="SearchQueryType">
    <complexContent>
      <extension base="spml:QueryClauseType">
        <sequence>
          <annotation>
            <documentation>Open content is one or more instances of
            QueryClauseType (including SelectionType) or
            LogicalOperator.</documentation>
          </annotation>
          <element name="basePsoID" type="spml:PSOIdentifierType"
            minOccurs="0"/>
        </sequence>
        <attribute name="targetID" type="string" use="optional"/>
        <attribute name="scope" type="spmlsearch:ScopeType"
            use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ResultsIteratorType">
    <complexContent>

```

```

        <extension base="spml:ExtensibleType">
            <attribute name="ID" type="xsd:ID"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="SearchRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="query" type="spmlsearch:SearchQueryType"
minOccurs="0"/>
                <element name="includeDataForCapability" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="returnData" type="spml:ReturnDataType"
use="optional" default="everything"/>
            <attribute name="maxSelect" type="xsd:int" use="optional"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="SearchResponseType">
    <complexContent>
        <extension base="spml:ResponseType">
            <sequence>
                <element name="pso" type="spml:PSOType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="iterator"
type="spmlsearch:ResultsIteratorType" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="IterateRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="iterator"
type="spmlsearch:ResultsIteratorType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="CloseIteratorRequestType">
    <complexContent>
        <extension base="spml:RequestType">
            <sequence>
                <element name="iterator"
type="spmlsearch:ResultsIteratorType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

```

```
<complexType name="LogicalOperatorType">
  <complexContent>
    <extension base="spml:QueryClauseType">
    </extension>
  </complexContent>
</complexType>

<element name="query" type="spmlsearch:SearchQueryType"/>
<element name="and" type="spmlsearch:LogicalOperatorType"/>
<element name="or" type="spmlsearch:LogicalOperatorType"/>
<element name="not" type="spmlsearch:LogicalOperatorType"/>
<element name="searchRequest" type="spmlsearch:SearchRequestType"/>
<element name="searchResponse" type="spmlsearch:SearchResponseType"/>
<element name="iterateRequest" type="spmlsearch:IterateRequestType"/>
<element name="iterateResponse" type="spmlsearch:SearchResponseType"/>
<element name="closeIterateRequest"
type="spmlsearch:CloseIteratorRequestType"/>
  <element name="closeIteratorResponse" type="spml:ResponseType"/>

</schema>
```

## Appendix G. Suspend Capability XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_SPMLv2_suspend_27.xsd -->
<!-- -->
<!-- Draft schema for SPML v2.0 suspend capabilities. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0:suspend"
  xmlns:spmlsuspend="urn:oasis:names:tc:SPML:2:0:suspend"
  xmlns:spml="urn:oasis:names:tc:SPML:2:0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <import namespace="urn:oasis:names:tc:SPML:2:0"
    schemaLocation="draft_pstc_SPMLv2_core_27.xsd"/>

  <complexType name="SuspendRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="psoID" type="spml:PSOIdentifierType"/>
        </sequence>
        <attribute name="effectiveDate" type="dateTime" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ResumeRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="psoID" type="spml:PSOIdentifierType"/>
        </sequence>
        <attribute name="effectiveDate" type="dateTime"
use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ActiveRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="psoID" type="spml:PSOIdentifierType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

```

```

    </complexContent>
  </complexType>

  <complexType name="ActiveResponseType">
    <complexContent>
      <extension base="spml:ResponseType">
        <attribute name="active" type="boolean" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="IsActiveType">
    <complexContent>
      <extension base="spml:QueryClauseType">
      </extension>
    </complexContent>
  </complexType>

  <element name="isActive" type="spmlsuspend:IsActiveType"/>
  <element name="suspendRequest" type="spmlsuspend:SuspendRequestType"/>
  <element name="suspendResponse" type="spml:ResponseType"/>
  <element name="resumeRequest" type="spmlsuspend:ResumeRequestType"/>
  <element name="resumeResponse" type="spml:ResponseType"/>
  <element name="activeRequest" type="spmlsuspend:ActiveRequestType"/>
  <element name="activeResponse" type="spmlsuspend:ActiveResponseType"/>

</schema>

```

## Appendix H. Updates Capability XSD

```

<?xml version="1.0" encoding="UTF-8"?>
<!--*****-->
<!-- draft_pstc_spmlv2_updates_27.xsd -->
<!-- Draft schema for SPML v2.0 updates capabilities. -->
<!-- -->
<!-- Editors: -->
<!-- Jeff Bohren (Jeff_Bohren@bmc.com) -->
<!-- -->
<!-- -->
<!-- Copyright (C) The Organization for the Advancement of -->
<!-- Structured Information Standards [OASIS] 2005. All Rights -->
<!-- Reserved. -->
<!--*****-->
<schema targetNamespace="urn:oasis:names:tc:SPML:2:0:updates"
  xmlns:spml="urn:oasis:names:tc:SPML:2:0"
  xmlns:spmlupdates="urn:oasis:names:tc:SPML:2:0:updates"
  xmlns:spmlsearch="urn:oasis:names:tc:SPML:2:0:search"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <import namespace="urn:oasis:names:tc:SPML:2:0"
    schemaLocation="draft_pstc_spmlv2_core_27.xsd"/>

  <import namespace="urn:oasis:names:tc:SPML:2:0:search"
    schemaLocation="draft_pstc_spmlv2_search_27.xsd"/>

  <complexType name="UpdatesRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element ref="spmlsearch:query" minOccurs="0"/>
          <element name="updatedByCapability" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="updatedSince" type="xsd:dateTime"
use="optional"/>
        <attribute name="token" type="xsd:string" use="optional"/>
        <attribute name="maxSelect" type="xsd:int" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <simpleType name="UpdateKindType">
    <restriction base="string">
      <enumeration value="add"/>
      <enumeration value="modify"/>
      <enumeration value="delete"/>
      <enumeration value="capability"/>
    </restriction>
  </simpleType>

  <complexType name="UpdateType">

```

```

    <complexContent>
      <extension base="spml:ExtensibleType">
        <sequence>
          <element name="psoID" type="spml:PSOIdentifierType" />
        </sequence>
        <attribute name="timestamp" type="xsd:dateTime"
use="required"/>
        <attribute name="updateKind"
type="spmlupdates:UpdateKindType" use="required"/>
        <attribute name="wasUpdatedByCapability" type="xsd:string"
use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="ResultsIteratorType">
    <complexContent>
      <extension base="spml:ExtensibleType">
        <attribute name="ID" type="xsd:ID"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="UpdatesResponseType">
    <complexContent>
      <extension base="spml:ResponseType">
        <sequence>
          <element name="update" type="spmlupdates:UpdateType"
minOccurs="0" maxOccurs="unbounded"/>
          <element name="iterator"
type="spmlupdates:ResultsIteratorType" minOccurs="0"/>
        </sequence>
        <attribute name="token" type="xsd:string" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="IterateRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="iterator"
type="spmlupdates:ResultsIteratorType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="CloseIteratorRequestType">
    <complexContent>
      <extension base="spml:RequestType">
        <sequence>
          <element name="iterator"
type="spmlupdates:ResultsIteratorType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

```

```
</complexContent>
</complexType>

<element name="updatesRequest" type="spmlupdates:UpdatesRequestType"/>
<element name="updatesResponse"
type="spmlupdates:UpdatesResponseType"/>
<element name="iterateRequest" type="spmlupdates:IterateRequestType"/>
<element name="iterateResponse"
type="spmlupdates:UpdatesResponseType"/>
<element name="closeIteratorRequest"
type="spmlupdates:CloseIteratatorRequestType"/>
<element name="closeIteratorResponse" type="spml:ResponseType"/>

</schema>
```

---

## Appendix I. Document References

4535

- 4536       **[AES]**                       National Institute of Standards and Technology (NIST), FIPS-197:  
4537                               Advanced Encryption Standard,  
4538                               <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>,  
4539                               National Institute of Standards and Technology (NIST)
- 4540       **[ARCHIVE-1]**               OASIS Provisioning Services Technical Committee, email archive,  
4541                               [http://www.oasis-  
4542                               open.org/apps/org/workgroup/provision/email/archives/index.  
4543                               html](http://www.oasis-open.org/apps/org/workgroup/provision/email/archives/index.html), OASIS PS-TC
- 4544       **[DS]**                           IETF/W3C, *W3C XML Signatures*, <http://www.w3.org/Signature/>,  
4545                               W3C/IETF
- 4546       **[DSML]**                       OASIS Directory Services Markup Standard, *DSML V2.0  
4547                               Specification*, [http://www.oasis-  
4548                               open.org/specs/index.php#dsmlv2](http://www.oasis-open.org/specs/index.php#dsmlv2), OASIS DSML Standard
- 4549       **[GLOSSARY]**               OASIS Provisioning Services TC, *Glossary of Terms*,  
4550                               [http://www.oasis-  
4551                               open.org/apps/org/workgroup/provision/download.php](http://www.oasis-open.org/apps/org/workgroup/provision/download.php), OASIS  
4552                               PS-TC
- 4553       **[RFC 2119]**               S. Bradner., *Key words for use in RFCs to Indicate Requirement  
4554                               Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF
- 4555       **[RFC 2246]**               T. Dierks and C. Allen, *The TLS Protocol*,  
4556                               <http://www.ietf.org/rfc/rfc2246.txt>, IETF
- 4557       **[SAML]**                       OASIS Security Services TC, [http://www.oasis-  
4558                               open.org/committees/tc\\_home.php?wg\\_abbrev=security](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security),  
4559                               OASIS SS-TC
- 4560       **[SOAP]**                       W3C XML Protocol Working Group,  
4561                               <http://www.w3.org/2000/xp/Group/>
- 4562       **[SPML-Bind]**               OASIS Provisioning Services TC, *SPML V1.0 Protocol Bindings*,  
4563                               [http://www.oasis-  
4564                               open.org/apps/org/workgroup/provision/download.php/1816/d  
4565                               raft-pstc-bindings-03.doc](http://www.oasis-open.org/apps/org/workgroup/provision/download.php/1816/draft-pstc-bindings-03.doc), OASIS PS-TC
- 4566       **[SPML-REQ]**               OASIS Provisioning Services Technical Committee, *Requirements*,  
4567                               [http://www.oasis-  
4568                               open.org/apps/org/workgroup/provision/download.php/2277/d  
4569                               raft-pstc-requirements-01.doc](http://www.oasis-open.org/apps/org/workgroup/provision/download.php/2277/draft-pstc-requirements-01.doc), OASIS PS-TC
- 4570       **[SPML-UC]**               OASIS Provisioning Services Technical Committee, *SPML V1.0  
4571                               Use Cases*, [http://www.oasis-  
4572                               open.org/apps/org/workgroup/provision/download.php/988/drf  
4573                               at-spml-use-cases-05.doc](http://www.oasis-open.org/apps/org/workgroup/provision/download.php/988/draft-spml-use-cases-05.doc), OASIS PS-TC
- 4574       **[SPMLv2-Profile-DSML]**   OASIS Provisioning Services Technical Committee, *SPMLv2  
4575                               DSMLv2 Profile*, OASIS PS-TC
- 4576       **[SPMLv2-Profile-XSD]**   OASIS Provisioning Services Technical Committee, *SPML V2 XSD  
4577                               Profile*, OASIS PS-TC

4578 4579	<b>[SPMLv2-REQ]</b>	OASIS Provisioning Services Technical Committee, Requirements, OASIS PS-TC
4580 4581	<b>[SPMLv2-ASYNC]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Async Capability of SPMLv2, OASIS PS-TC
4582 4583	<b>[SPMLv2-BATCH]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Batch Capability of SPMLv2, OASIS PS-TC
4584 4585	<b>[SPMLv2-BULK]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Bulk Capability of SPMLv2, OASIS PS-TC
4586 4587	<b>[SPMLv2-CORE]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Core Operations of SPMLv2, OASIS PS-TC
4588 4589	<b>[SPMLv2-PASS]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Password Capability of SPMLv2, OASIS PS-TC
4590 4591	<b>[SPMLv2-REF]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Reference Capability of SPMLv2, OASIS PS-TC
4592 4593	<b>[SPMLv2-SEARCH]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Search Capability of SPMLv2, OASIS PS-TC
4594 4595	<b>[SPMLv2-SUSPEND]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Suspend Capability of SPMLv2, OASIS PS-TC
4596 4597	<b>[SPMLv2-UPDATES]</b>	OASIS Provisioning Services Technical Committee, XML Schema Definitions for Updates Capability of SPMLv2, OASIS PS-TC
4598 4599	<b>[SPMLv2-UC]</b>	OASIS Provisioning Services Technical Committee., SPML V2.0 Use Cases, OASIS PS-TC
4600 4601 4602	<b>[WSS]</b>	OASIS Web Services Security (WSS) TC, <a href="http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss">http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss</a> , OASIS SS-TC
4603 4604	<b>[X509]</b>	RFC 2459 - Internet X.509 Public Key Infrastructure Certificate and CRL Profile, <a href="http://www.ietf.org/rfc/rfc2459.txt">http://www.ietf.org/rfc/rfc2459.txt</a>
4605 4606	<b>[XSD]</b>	W3C Schema WG ., <i>W3C XML Schema</i> , <a href="http://www.w3.org/TR/xmlschema-1/">http://www.w3.org/TR/xmlschema-1/</a> W3C
4607		

---

4608 **Appendix J. Acknowledgments**

4609 The following individuals were voting members of the Provisioning Services committee at the time  
4610 that this version of the specification was issued:

4611 Jeff Bohren, BMC  
4612 Robert Boucher, CA  
4613 Gary Cole, Sun Microsystems  
4614 Rami Elron, BMC  
4615 Marco Fanti, Thor Technologies  
4616 James Hu, HP  
4617 Martin Raeppele, SAP  
4618 Gavenraj Sodhi, CA  
4619 Kent Spaulding, Sun Microsystems  
4620

4621

---

## Appendix K. Notices

4622 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
4623 that might be claimed to pertain to the implementation or use of the technology described in this  
4624 document or the extent to which any license under such rights might or might not be available;  
4625 neither does it represent that it has made any effort to identify any such rights. Information on  
4626 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
4627 website. Copies of claims of rights made available for publication and any assurances of licenses to  
4628 be made available, or the result of an attempt made to obtain a general license or permission for  
4629 the use of such proprietary rights by implementors or users of this specification, can be obtained  
4630 from the OASIS President.

4631 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
4632 applications, or other proprietary rights which may cover technology that may be required to  
4633 implement this specification. Please address the information to the OASIS President.

4634 Copyright © OASIS Open 2005. *All Rights Reserved.*

4635 This document and translations of it may be copied and furnished to others, and derivative works  
4636 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
4637 published and distributed, in whole or in part, without restriction of any kind, provided that the above  
4638 copyright notice and this paragraph are included on all such copies and derivative works. However,  
4639 this document itself does not be modified in any way, such as by removing the copyright notice or  
4640 references to OASIS, except as needed for the purpose of developing OASIS specifications, in  
4641 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights  
4642 document must be followed, or as required to translate it into languages other than English.

4643 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
4644 successors or assigns.

4645 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
4646 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
4647 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY  
4648 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
4649 PARTICULAR PURPOSE.

4650