



1

2

# OASIS Service Provisioning Markup Language (SPML) Version 2

3

4

## OASIS Standard 2006 April 1

5

6

Document identifier: pstc-spml2-os.pdf

7

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

8

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

9

Editor:

10

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

11

Contributors:

12

Jeff Bohren, BMC

13

Robert Boucher, CA

14

Doron Cohen, BMC

15

Gary Cole, Sun Microsystems

16

Cal Collingham, CA

17

Rami Elron, BMC

18

Marco Fanti, Thor Technologies

19

Ian Glazer, IBM

20

James Hu, HP

21

Ron Jacobsen, CA

22

Jeff Larson, Sun Microsystems

23

Hal Lockhart, BEA

24

Prateek Mishra, Oracle Corporation

25

Martin Raeppe, SAP

26

Darran Rolls, Sun Microsystems

27

Kent Spaulding, Sun Microsystems

28

Gavenraj Sodhi, CA

29

Cory Williams, IBM

30

Gerry Woods, SOA Software

31

Abstract:

32

This specification defines the concepts and operations of an XML-based provisioning

33

request-and-response protocol.

34

Status:

35 This is an OASIS Standard document produced by the Provisioning Services Technical  
36 Committee. It was approved by the OASIS membership on 1 April 2006.

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  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79

## Table of contents

|         |  |    |
|---------|--|----|
| 1       | Introduction.....                                    | 7  |
| 1.1     | Purpose .....  | 7  |
| 1.2     | Organization .....                                   | 7  |
| 1.3     | Audience.....  | 7  |
| 1.4     | Notation .....                                       | 8  |
| 1.4.1   | Normative sections .....                             | 8  |
| 1.4.2   | Normative terms.....                                 | 8  |
| 1.4.3   | Typographical conventions .....                      | 8  |
| 1.4.4   | Namespaces .....                                     | 9  |
| 2       | Concepts .....                                       | 10 |
| 2.1     | Domain Model .....                                   | 10 |
| 2.1.1   | Requestor .....                                      | 10 |
| 2.1.2   | Provider.....  | 11 |
| 2.1.3   | Target.....  | 11 |
| 2.1.3.1 | Target Schema .....                                  | 11 |
| 2.1.3.2 | Supported Schema Entities .....                      | 12 |
| 2.1.3.3 | Capabilities.....                                    | 12 |
| 2.1.4   | Provisioning Service Object (PSO).....               | 13 |
| 2.2     | Core Protocol.....                                   | 13 |
| 2.3     | Profile.....   | 13 |
| 3       | Protocol .....                                       | 14 |
| 3.1     | Request/Response Model .....                         | 14 |
| 3.1.1   | Conversational flow.....                             | 16 |
| 3.1.2   | Status and Error codes .....                         | 16 |
| 3.1.2.1 | Status (normative).....                              | 17 |
| 3.1.2.2 | Error (normative).....                               | 17 |
| 3.1.2.3 | Error Message (normative) .....                      | 18 |
| 3.1.3   | Synchronous and asynchronous operations .....        | 19 |
| 3.1.3.1 | ExecutionMode attribute .....                        | 19 |
| 3.1.3.2 | Async Capability .....                               | 19 |
| 3.1.3.3 | Determining execution mode .....                     | 20 |
| 3.1.3.4 | Results of asynchronous operations (normative) ..... | 22 |
| 3.1.4   | Individual and batch requests .....                  | 22 |
| 3.2     | Identifiers .....                                    | 22 |
| 3.2.1   | Request Identifier (normative) .....                 | 23 |
| 3.2.2   | Target Identifier (normative) .....                  | 23 |
| 3.2.3   | PSO Identifier (normative) .....                     | 24 |
| 3.3     | Selection.....                                       | 26 |

|     |         |  |    |
|-----|---------|--|----|
| 80  | 3.3.1   | QueryClauseType .....                          | 26 |
| 81  | 3.3.2   | Logical Operators .....                        | 26 |
| 82  | 3.3.3   | SelectionType .....                            | 27 |
| 83  | 3.3.3.1 | SelectionType in a Request (normative) .....   | 27 |
| 84  | 3.3.3.2 | SelectionType Processing (normative) .....     | 28 |
| 85  | 3.3.3.3 | SelectionType Errors (normative) .....         | 29 |
| 86  | 3.3.4   | SearchQueryType .....                          | 29 |
| 87  | 3.3.4.1 | SearchQueryType in a Request (normative) ..... | 30 |
| 88  | 3.3.4.2 | SearchQueryType Errors (normative) .....       | 31 |
| 89  | 3.4     | CapabilityData .....                           | 32 |
| 90  | 3.4.1   | CapabilityDataType .....                       | 32 |
| 91  | 3.4.1.1 | CapabilityData in a Request (normative) .....  | 33 |
| 92  | 3.4.1.2 | CapabilityData Processing (normative) .....    | 34 |
| 93  | 3.4.1.3 | CapabilityData Errors (normative) .....        | 37 |
| 94  | 3.4.1.4 | CapabilityData in a Response (normative) ..... | 37 |
| 95  | 3.5     | Transactional Semantics .....                  | 39 |
| 96  | 3.6     | Operations .....                               | 39 |
| 97  | 3.6.1   | Core Operations .....                          | 39 |
| 98  | 3.6.1.1 | listTargets .....                              | 39 |
| 99  | 3.6.1.2 | add .....                                      | 50 |
| 100 | 3.6.1.3 | lookup .....                                   | 56 |
| 101 | 3.6.1.4 | modify .....                                   | 61 |
| 102 | 3.6.1.5 | delete .....                                   | 71 |
| 103 | 3.6.2   | Async Capability .....                         | 74 |
| 104 | 3.6.2.1 | cancel .....                                   | 75 |
| 105 | 3.6.2.2 | status .....                                   | 77 |
| 106 | 3.6.3   | Batch Capability .....                         | 83 |
| 107 | 3.6.3.1 | batch .....                                    | 83 |
| 108 | 3.6.4   | Bulk Capability .....                          | 90 |
| 109 | 3.6.4.1 | bulkModify .....                               | 90 |
| 110 | 3.6.4.2 | bulkDelete .....                               | 92 |
| 111 | 3.6.5   | Password Capability .....                      | 95 |
| 112 | 3.6.5.1 | setPassword .....                              | 95 |
| 113 | 3.6.5.2 | expirePassword .....                           | 97 |
| 114 | 3.6.5.3 | resetPassword .....                            | 98 |

|     |         |  |     |
|-----|---------|--|-----|
| 115 | 3.6.5.4 | validatePassword.....                                    | 100 |
| 116 | 3.6.6   | Reference Capability.....                                | 103 |
| 117 | 3.6.6.1 | Reference Definitions.....                               | 105 |
| 118 | 3.6.6.2 | References.....  | 106 |
| 119 | 3.6.6.3 | Complex References .....                                 | 106 |
| 120 | 3.6.6.4 | Reference CapabilityData in a Request (normative) .....  | 111 |
| 121 | 3.6.6.5 | Reference CapabilityData Processing (normative).....     | 113 |
| 122 | 3.6.6.6 | Reference CapabilityData Errors (normative).....         | 115 |
| 123 | 3.6.6.7 | Reference CapabilityData in a Response (normative) ..... | 115 |
| 124 | 3.6.7   | Search Capability.....                                   | 116 |
| 125 | 3.6.7.1 | search .....   | 117 |
| 126 | 3.6.7.2 | iterate .....  | 123 |
| 127 | 3.6.7.3 | closeIterator .....                                      | 129 |
| 128 | 3.6.8   | Suspend Capability .....                                 | 133 |
| 129 | 3.6.8.1 | suspend.....   | 133 |
| 130 | 3.6.8.2 | resume .....   | 135 |
| 131 | 3.6.8.3 | active.....  | 137 |
| 132 | 3.6.9   | Updates Capability.....                                  | 140 |
| 133 | 3.6.9.1 | updates .....  | 141 |
| 134 | 3.6.9.2 | iterate .....  | 147 |
| 135 | 3.6.9.3 | closeIterator .....                                      | 152 |
| 136 | 3.7     | Custom Capabilities.....                                 | 157 |
| 137 | 4       | Conformance (normative) .....                            | 158 |
| 138 | 4.1     | Core operations and schema are mandatory.....            | 158 |
| 139 | 4.2     | Standard capabilities are optional .....                 | 158 |
| 140 | 4.3     | Custom capabilities must not conflict .....              | 158 |
| 141 | 4.4     | Capability Support is all-or-nothing .....               | 159 |
| 142 | 4.5     | Capability-specific data.....                            | 159 |
| 143 | 5       | Security Considerations .....                            | 160 |
| 144 | 5.1     | Use of SSL 3.0 or TLS 1.0 .....                          | 160 |
| 145 | 5.2     | Authentication.....                                      | 160 |
| 146 | 5.3     | Message Integrity .....                                  | 160 |
| 147 | 5.4     | Message Confidentiality .....                            | 160 |
| 148 |         | Appendix A. Core XSD .....                               | 161 |
| 149 |         | Appendix B. Async Capability XSD .....                   | 169 |
| 150 |         | Appendix C. Batch Capability XSD.....                    | 171 |
| 151 |         | Appendix D. Bulk Capability XSD .....                    | 173 |
| 152 |         | Appendix E. Password Capability XSD .....                | 175 |
| 153 |         | Appendix F. Reference Capability XSD.....                | 177 |
| 154 |         | Appendix G. Search Capability XSD .....                  | 179 |
| 155 |         | Appendix H. Suspend Capability XSD.....                  | 182 |
| 156 |         | Appendix I. Updates Capability XSD .....                 | 184 |

|     |                                       |     |
|-----|---------------------------------------|-----|
| 157 | Appendix J. Document References ..... | 187 |
| 158 | Appendix K. Acknowledgments .....     | 189 |
| 159 | Appendix L. Notices .....             | 190 |

---

# 160 1 Introduction

## 161 1.1 Purpose

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

## 164 1.2 Organization

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

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

173 Each section that describes an operation includes:

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

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

## 185 1.3 Audience

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

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

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

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

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

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

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

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

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

## 197 1.4 Notation

### 198 1.4.1 Normative sections

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

### 201 1.4.2 Normative terms

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

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

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

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

### 212 1.4.3 Typographical conventions

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

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

215 Listings of SPML schemas appear like this.



216

217

Example code listings appear like this.

## 218 1.4.4 Namespaces

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

- 222 • The prefix `dsml`: stands for the Directory Services Markup Language namespace **[DSML]**.
- 223 • The prefix `xsd`: stands for the W3C XML Schema namespace **[XSD]**.
- 224 • The prefix `spml`: stands for the SPMLv2 Core XSD namespace  
225 **[SPMLv2-CORE]**.
- 226 • The prefix `spmlasync`: stands for the SPMLv2 Async Capability XSD namespace.  
227 **[SPMLv2-ASYNC]**.
- 228 • The prefix `spmlbatch`: stands for the SPMLv2 Batch Capability XSD namespace  
229 **[SPMLv2-BATCH]**.
- 230 • The prefix `spmlbulk`: stands for the SPMLv2 Bulk Capability XSD namespace  
231 **[SPMLv2-BULK]**.
- 232 • The prefix `spmlpass`: stands for the SPMLv2 Password Capability XSD namespace  
233 **[SPMLv2-PASS]**.
- 234 • The prefix `spmlref`: stands for the SPMLv2 Reference Capability XSD namespace  
235 **[SPMLv2-REF]**.
- 236 • The prefix `spmlsearch`: stands for the SPMLv2 Search Capability XSD namespace  
237 **[SPMLv2-SEARCH]**.
- 238 • The prefix `spmlsuspend`: stands for the SPMLv2 Suspend Capability XSD namespace  
239 **[SPMLv2-SUSPEND]**.
- 240 • The prefix `spmlupdates`: stands for the SPMLv2 Updates Capability XSD namespace  
241 **[SPMLv2-UPDATES]**.

242

## 2 Concepts

243

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

244

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.

245

246

247

248

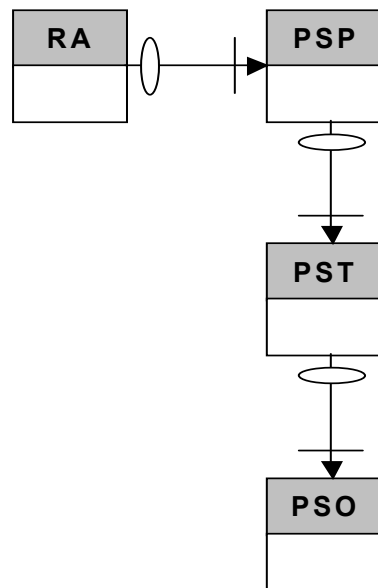
### 2.1 Domain Model

249

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.

250

251



252

253

Figure 1. Domain model elements

254

#### 2.1.1 Requestor

255

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:

256

257

258

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

259

**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

260

261 requestor and its provider have established a trust relationship between them. The details of  
262 establishing and maintaining this trust relationship are beyond the scope of this specification.

## 263 2.1.2 Provider

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

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

## 272 2.1.3 Target

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

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

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

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

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

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

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

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

### 297 2.1.3.1 Target Schema

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

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

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

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

### 308 **2.1.3.2 Supported Schema Entities**

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

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

### 314 **2.1.3.3 Capabilities**

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

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

319 • The OASIS *PSTC* defines each *standard capability* in an SPML namespace.  
320 See the section titled "**Namespaces**".

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

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

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

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

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

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

## 346 2.1.4 Provisioning Service Object (PSO)

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

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

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

## 352 2.2 Core Protocol

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

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

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

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

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

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

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

## 368 2.3 Profile

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

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

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

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

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

379

## 3 Protocol

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

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

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

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

### 3.1 Request/Response Model

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

```

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

- 407
- 408 • the [exchange of requests and responses](#) between requestor and provider
  - 409 • [synchronous and asynchronous execution](#) of operations
  - 409 • [individual and batch requests](#)

### 410 3.1.1 Conversational flow

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

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

### 421 3.1.2 Status and Error codes

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

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

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

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



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

### 440 3.1.2.1 Status (normative)

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

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

### 455 3.1.2.2 Error (normative)

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

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

- 476 - The request may have specified a container that is *may not contain the specified object*.  
 477 The target (or a system or application that underlies the target) may restrict the types of  
 478 objects that the provider can add to the specified container. The target (or a system or  
 479 application that underlies the target) may restrict the containers to which the provider can  
 480 add the specified object.
- 481 • A response that specifies "error='resultSetTooLarge'" indicates that the provider  
 482 cannot return (or cannot queue for subsequent iteration—as in the case of an overlarge search  
 483 result) the entire result of an operation.  
 484  
 485 In this case, the requestor may be able to refine the request so as to produce a smaller result.  
 486 For example, a requestor might break a single search operation into several search requests,  
 487 each of which selects a sub-range of the original (overlarge) search result.
  - 488 • A response that specifies "error='customError'" indicates that the provider has  
 489 encountered an error that none of the standard error code values describes.  
 490 In this case, the provider's response SHOULD provide error information in a format that is  
 491 available to the requestor. SPMLv2 does not specify the format of a custom error.
- 492 Several additional values of {ErrorCode} apply only to certain operations. (For example,  
 493 "error='unsupportedProfile'" applies only to the listTargets operation. Currently,  
 494 "error='invalidIdentifier'" and "error='alreadyExists'" apply only to the add  
 495 operation.) The section that discusses each operation also discusses any value of {ErrorCode}  
 496 that is specific to that operation.

### 497 **3.1.2.3 Error Message (normative)**

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

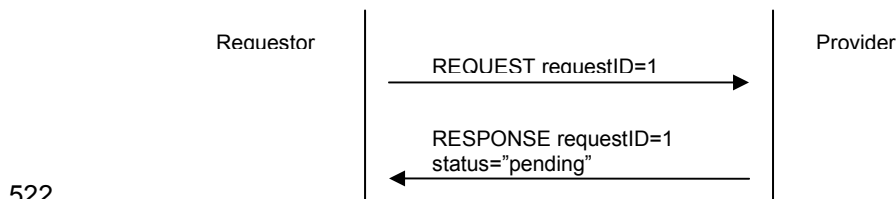
- 501 • A response that specifies "status='failure'" SHOULD contain at least one  
 502 <errorMessage> that describes *each condition that caused the failure*.
- 503 • A response that specifies "status='success'" MAY contain any number of  
 504 <errorMessage> elements that describe *warning* conditions.
- 505 • A response that specifies "status='success'" SHOULD NOT contain an  
 506 <errorMessage> element that describes an *informational* message

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

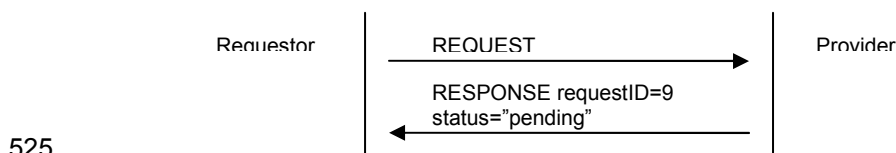
### 509 3.1.3 Synchronous and asynchronous operations

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

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



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



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

#### 530 3.1.3.1 ExecutionMode attribute

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

#### 536 3.1.3.2 Async Capability

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

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

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

### 547 **3.1.3.3 Determining execution mode**

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

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

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

#### 558 **3.1.3.3.1 Requestor specifies synchronous execution (normative)**

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

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

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

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

#### 577 **3.1.3.3.2 Requestor specifies asynchronous execution (normative)**

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

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

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

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

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

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

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

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

### 602 **3.1.3.3 Provider chooses synchronous execution (normative)**

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

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

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

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

### 615 **3.1.3.4 Provider chooses asynchronous execution (normative)**

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

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

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

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

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

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

### 636 3.1.3.4 Results of asynchronous operations (normative)

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

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

### 645 3.1.4 Individual and batch requests

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

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

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

## 654 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>

```

655 SPMLv2 uses several different types of identifiers.

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

### 663 3.2.1 Request Identifier (normative)

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

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

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

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

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

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

### 685 3.2.2 Target Identifier (normative)

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

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

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

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

### 697 **3.2.3 PSO Identifier (normative)**

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

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

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

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

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

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

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

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



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

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

## 738 3.3 Selection

### 739 3.3.1 QueryClauseType

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

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

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

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

### 756 3.3.2 Logical Operators

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

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

### 765 3.3.3 SelectionType

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

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

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

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

#### 783 3.3.3.1 SelectionType in a Request (normative)

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

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

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

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

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

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

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

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

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

### 815 **3.3.3.2 SelectionType Processing (normative)**

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

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

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

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

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

### 834 3.3.3.3 SelectionType Errors (normative)

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

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

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

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

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

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

### 860 3.3.4 SearchQueryType

861 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"/>

```

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

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

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

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

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

### 872 3.3.4.1 SearchQueryType in a Request (normative)

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

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

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

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

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

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

- 891 object.)  
892 See the section titled "[SearchQueryType Errors \(normative\)](#)" below.
- 893 • A requestor that wants the provider to search *only direct descendants* of the target or (that  
894 wants to search only direct descendants) of the object specified by <basePsoID> MUST  
895 specify "scope='oneLevel'".
  - 896 • A requestor that wants the provider to search *any direct or indirect descendant* of the target or  
897 (that wants to search any direct or indirect descendant) of the object specified by  
898 <basePsoID> MUST specify "scope='subTree'".
- 899 **Open content.** An instance of {SearchQueryType} MUST contain (as open content) exactly  
900 one instance of a type that extends {QueryClauseType}.
- 901 • Any capability may define elements of (a type that extends) {QueryClauseType}. These  
902 elements allow a requestor to select objects based on capability-defined data.  
903 See the section titled "[QueryClauseType](#)" above.
  - 904 • A <select> element is an instance of {SelectionType}, which extends  
905 {QueryClauseType} to filter objects based on schema-defined content.  
906 See the section titled "[SelectionType in a Request \(normative\)](#)".
  - 907 • Logical Operators such as <and>, <or> and <not> combine individual selection criteria.  
908 A logical operator MUST contain at least one instance of a type that extends  
909 {QueryClauseType} or a (logical operator MUST contain at least one) logical operator.  
910 See the section titled "[Logical Operators](#)" above.

#### 911 **3.3.4.2 SearchQueryType Errors (normative)**

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



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

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

### 933 **3.4 CapabilityData**

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

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

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

#### 952 **3.4.1 CapabilityDataType**

953 SPMLv2 defines a {CapabilityDataType} that may occur in a request or in a response. Each  
954 instance of {CapabilityDataType} contains all of the data that is *associated with a particular*  
955 *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>
```

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

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

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

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

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

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

### 978 **3.4.1.1 CapabilityData in a Request (normative)**

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

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

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

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

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

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

1007 The section titled "[CapabilityData Processing \(normative\)](#)" describes the default handling for  
1008 capability-specific data. Any capability for which the default handling is inappropriate MUST specify  
1009 how operations should handle data specific to that capability. The section titled "[Reference](#)  
1010 [CapabilityData Processing \(normative\)](#)" specifies handling of data specific to the Reference  
1011 Capability.

1012 **Capability defines structure.** Any capability that implies capability-specific data SHOULD specify  
1013 the structure of that data. (That is, the capability to which the "capabilityURI" attribute of an  
1014 instance of {CapabilityDataType} refers SHOULD specify the structure of data that the  
1015 instance of {CapabilityDataType} contains.) Furthermore, any capability that implies  
1016 capability-specific data and for which the default processing of capability-specific data is  
1017 inappropriate MUST specify the structure of that capability-specific data and MUST specify how  
1018 operations handle that capability-specific data. See the discussion of "[Capability-specific data](#)"  
1019 within the section titled "[Conformance](#)".

1020 Of the capabilities that SPMLv2 defines, only the Reference Capability implies that capability-  
1021 specific data may be associated with an object. The Reference Capability specifies that an  
1022 instance of {CapabilityDataType} that refers to the Reference Capability  
1023 (e.g., a <capabilityData> element that specifies  
1024 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'"  
1025 MUST contain at least one reference to another object. The Reference Capability defines the  
1026 structure of a <reference> element as {ReferenceType}.) The Reference Capability also  
1027 specifies that each <reference> must match a supported <referenceDefinition>.  
1028 See the section titled "[Reference CapabilityData in a Request \(normative\)](#)".

### 1029 **3.4.1.2 CapabilityData Processing (normative)**

1030 **capabilityURI.** An instance of {CapabilityDataType} MUST specify a value of  
1031 "capabilityURI" that identifies a *supported capability*. That is, the (value of the)  
1032 "capabilityURI" attribute for an instance of {CapabilityDataType} MUST match the (value  
1033 of the) "namespaceURI" attribute of a <capability> the provider supports *for the target* (that  
1034 contains the object to be manipulated) and (that the provider supports on that target) *for the*  
1035 *schema entity* of which the object to be manipulated is an instance.

1036 For normative specifics of supported capabilities,  
1037 see the section titled "[listTargetsResponse \(normative\)](#)".

1038 **mustUnderstand.** The "mustUnderstand" attribute tells a provider whether the default  
1039 processing of capability-specific data is sufficient for the content of an instance of

1040 {CapabilityDataType}. (The next topic within this section describes the default processing of  
1041 capability-specific data.)

- 1042 • If an instance of {CapabilityDataType} specifies "mustUnderstand='true'", then  
1043 the provider MUST handle the data (that the instance of {CapabilityDataType} contains)  
1044 in the manner that the corresponding capability specifies.  
1045

1046 If the provider cannot handle the data (that the instance of {CapabilityDataType} contains)  
1047 in the manner that the corresponding capability specifies,  
1048 then the provider's response MUST specify "status='failure'".  
1049 See the section titled "[CapabilityData Errors \(normative\)](#)" below.

- 1050 • If an instance of {CapabilityDataType} specifies "mustUnderstand='false'"  
1051 or an instance of {CapabilityDataType} omits "mustUnderstand",  
1052 then a provider MAY handle the data (that the instance of {CapabilityDataType} contains)  
1053 according to the default processing that is described below.

- 1054 - If the provider knows that the corresponding capability (e.g., the Reference Capability)  
1055 specifies special handling, then the provider SHOULD process the data (that the instance  
1056 of {CapabilityDataType} contains) in the manner that the corresponding capability  
1057 specifies.

- 1058 - If the provider knows that the corresponding capability (e.g., the Reference Capability)  
1059 specifies special handling but the provider *cannot provide the special handling* that the  
1060 corresponding capability specifies, then the provider MUST handle the data (that the  
1061 instance of {CapabilityDataType} contains) according to the default processing  
1062 that is described below.

- 1063 - If the provider does not know whether the corresponding capability specifies special  
1064 handling, then the provider MUST handle the data (that the instance of  
1065 {CapabilityDataType} contains) according to the default processing  
1066 that is described below.

1067 **Default processing.** By default, a provider treats the set of data specific to each capability as if it  
1068 were *opaque*. That is, a provider processes the content of an instance of  
1069 {CapabilityDataType} *exactly as it is* --without manipulating that content in any way.

1070 (The provider needs to perform capability-specific processing only if the instance of  
1071 {CapabilityDataType} specifies "mustUnderstand='true'" or if the instance of  
1072 {CapabilityDataType} refers to the Reference Capability. See the topic named  
1073 "mustUnderstand" immediately above within this section.)

- 1074 • If an <addRequest> contains an instance of {CapabilityDataType},  
1075 then the provider MUST associate the instance of {CapabilityDataType} *exactly as it is*  
1076 (i.e., without manipulating its content in any way) with the newly created object.

- 1077 • If a <modification> contains an instance of {CapabilityDataType},  
1078 then the default handling depends on the "modificationMode" of that <modification>  
1079 and also depends on whether an instance of {CapabilityDataType} that specifies the  
1080 same "capabilityURI" is already associated with the object to be modified.

- 1081 - If a <modification> that specifies "modificationMode='add'"  
1082 contains an instance of {CapabilityDataType},  
1083 then the provider MUST *append the content* of the instance of {CapabilityDataType}  
1084 that the <modification> contains *exactly as it is* to (the content of) any instance of  
1085 {CapabilityDataType} that is already associated with the object to be modified

1086 and that specifies the same "capabilityURI".  
1087  
1088 If no instance of {CapabilityDataType} that specifies the same "capabilityURI"  
1089 (as the instance of {CapabilityDataType} that the <modification> contains)  
1090 is already associated with the object to be modified,  
1091 then the provider MUST the associate with the modified object the <capabilityData>  
1092 (that the <modification> contains) *exactly as it is* .

1093 - If a <modification> that specifies "modificationMode='replace'"  
1094 contains an instance of {CapabilityDataType},  
1095 then the provider MUST *replace entirely* any instance of {CapabilityDataType}  
1096 that is already associated with the object to be modified  
1097 and that specifies the same "capabilityURI"  
1098 with the instance of {CapabilityDataType} that the <modification> contains  
1099 *exactly as it is*.

1100  
1101 If no instance of {CapabilityDataType} that specifies the same "capabilityURI"  
1102 (as the instance of {CapabilityDataType} that the <modification> contains)  
1103 is already associated with the object to be modified,  
1104 then the provider MUST the associate with the modified object the <capabilityData>  
1105 (that the <modification> contains) *exactly as it is* .

1106 - If a <modification> that specifies "modificationMode='delete'"  
1107 contains an instance of {CapabilityDataType},  
1108 then the provider MUST *delete entirely* any instance of {CapabilityDataType}  
1109 that is already associated with the object to be modified  
1110 and that specifies the same "capabilityURI"

1111  
1112 If no instance of {CapabilityDataType} that specifies the same "capabilityURI"  
1113 (as the instance of {CapabilityDataType} that the <modification> contains)  
1114 is already associated with the object to be modified, then the provider MUST do nothing.  
1115 In this case, the provider's response MUST NOT specify "status='failure'"  
1116 unless there is some other reason to do so.

1117 **Capability-specific handling.** Any capability that implies capability-specific data and for which the  
1118 default processing of capability-specific data is inappropriate MUST specify how (at least the core)  
1119 operations should process that data. (That is, the capability to which the "capabilityURI"  
1120 attribute of an instance of {CapabilityDataType} refers MUST specify how operations should  
1121 process the data that the instance of {CapabilityDataType} contains if the default processing  
1122 for capability-specific data is inappropriate.)

1123 See the discussion of "[Capability-specific data](#)" within the section titled "[Conformance](#)".

1124 Of the standard capabilities that SPMLv2 defines, only the Reference Capability implies that  
1125 capability-specific data may be associated with an object. The Reference Capability specifies how  
1126 operations should process the content of an instance of {CapabilityDataType} that specifies  
1127 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'".

1128 See the section titled "[Reference CapabilityData Processing \(normative\)](#)".

### 1129 3.4.1.3 CapabilityData Errors (normative)

1130 A provider's response to a request that contains an instance of {CapabilityDataType}  
1131 MUST specify an error if any of the following is true:

- 1132 • The instance of {CapabilityDataType} specifies "mustUnderstand='true'"  
1133 and the provider does not recognize the value of the "capabilityURI" attribute  
1134 as identifying a capability that the provider supports *for the target* that contains the object to be  
1135 manipulated *and* that the provider supports *for the schema entity* of which the object to be  
1136 manipulated is an instance.
- 1137 • The instance of {CapabilityDataType} specifies "mustUnderstand='true'"  
1138 and the capability to which its "capabilityURI" refers does not specify the structure of data  
1139 specific to that capability.
- 1140 • The instance of {CapabilityDataType} specifies "mustUnderstand='true'" and the  
1141 capability to which its "capabilityURI" refers does not specify how operations should  
1142 process data specific to that capability.
- 1143 • The request contains two or more instances of {CapabilityDataType} that specify the  
1144 same value of "capabilityURI".

1145 In addition, a provider's response to a request that contains an instance of  
1146 {CapabilityDataType} MAY specify an error if any of the following is true:

- 1147 • The provider does not recognize the value of the "capabilityURI" (that the instance of  
1148 {CapabilityDataType} specifies) as identifying a capability that the provider supports *for*  
1149 *the target* that contains the object to be manipulated *and* that the provider supports *for the*  
1150 *schema entity* of which the object to be manipulated is an instance.  
1151  
1152 Alternatively, the provider MAY perform the default handling as described above  
1153 in the section titled "[CapabilityData Processing \(normative\)](#)".

1154 A provider's response to a request that contains an instance of {CapabilityDataType}  
1155 SHOULD contain an <errorMessage> for each instance of {CapabilityDataType} that the  
1156 provider could not process.

1157 **Capability-specific errors.** Any capability that implies capability-specific data MAY specify  
1158 additional errors related to that data. (That is, the capability to which the "capabilityURI"  
1159 attribute of an instance of {CapabilityDataType} refers MAY specify additional errors related to  
1160 that instance of {CapabilityDataType}.)

1161 Of the capabilities that SPMLv2 defines, only the Reference Capability implies that capability-  
1162 specific data may be associated with an object. The Reference Capability specifies additional  
1163 errors related to any instance of {CapabilityDataType} that refers to the Reference Capability  
1164 See the section titled "[Reference CapabilityData Errors \(normative\)](#)".

### 1165 3.4.1.4 CapabilityData in a Response (normative)

1166 **capabilityURI.** An instance of {CapabilityDataType} MUST specify a value of  
1167 "capabilityURI" that identifies a *supported capability*. That is, the (value of the)  
1168 "capabilityURI" attribute for an instance of {CapabilityDataType} MUST match the (value  
1169 of the) "namespaceURI" attribute of a <capability> the provider supports *for the target* (that  
1170 contains the object to be manipulated) and (that the provider supports on that target) *for the*

1171 *schema entity* of which the object to be manipulated is an instance.  
1172 See the section titled "[listTargetsResponse \(normative\)](#)".

1173 **One per capability.** No more than one instance of {CapabilityDataType} within a response  
1174 may refer to a given capability. That is, a response MUST NOT contain two (and a request MUST  
1175 NOT contain more than two) instances of {CapabilityDataType} that specify the same value of  
1176 "capabilityURI".

1177 This implies that an instance of {CapabilityDataType} that refers to a certain capability MUST  
1178 contain *all the data* within that response *that is specific to that capability* and that is associated with  
1179 a particular object.

1180 **mustUnderstand.** An instance of {CapabilityDataType} within a response MAY specify  
1181 "mustUnderstand". A provider SHOULD preserve any "mustUnderstand" attribute of an  
1182 instance of {CapabilityDataType}. See the discussions of the "mustUnderstand" attribute  
1183 within the sections titled "[CapabilityData in a Request \(normative\)](#)" and "[CapabilityData Processing \(normative\)](#)" above.  
1184

1185 **Capability defines structure.** Any capability that implies capability-specific data MUST specify the  
1186 structure of that data. (That is, the capability to which the "capabilityURI" attribute of an  
1187 instance of {CapabilityDataType} refers MUST specify the structure of data that the instance  
1188 of {CapabilityDataType} contains.) See the discussion of "[Custom Capabilities](#)" within the  
1189 section titled "[Conformance](#)".

1190 Of the capabilities that SPMLv2 defines, only the Reference Capability implies that capability-  
1191 specific data may be associated with an object. The Reference Capability specifies that an  
1192 instance of {CapabilityDataType} that refers to the Reference Capability MUST contain at  
1193 least one reference to another object. The Reference Capability defines the structure of a  
1194 <reference> element as {ReferenceType}.) The Reference Capability also specifies that  
1195 each <reference> must match a supported <referenceDefinition>.  
1196 See the section titled "[Reference CapabilityData in a Response \(normative\)](#)".



## 1197 **3.5 Transactional Semantics**

1198 SPMLv2 specifies no transactional semantics. This specification defines no operation that implies  
1199 atomicity. That is, no core operation defines (and no operation that is part of one of SPMLv2's  
1200 standard capabilities defines) a logical unit of work that must be committed or rolled back as a unit.

1201 Provisioning operations are notoriously difficult to undo and redo. For security reasons, many  
1202 systems and applications will not allow certain identity management operations to be fully reversed  
1203 or repeated. (More generally, support for transactional semantics suggests participation in  
1204 externally managed transactions. Such participation is beyond the scope of this specification.)

1205 Any transactional semantics should be defined as a capability (or possibly as more than one  
1206 capability). See the section titled "[Custom Capabilities](#)". A transactional capability would define  
1207 operations that imply atomicity or (would define operations) that allow a requestor to specify  
1208 atomicity.

1209 Any provider that is able to support transactional semantics should then declare its support for such  
1210 a capability as part of the provider's response to the listTargets operation (as the provider would  
1211 declare its support for any other capability).

## 1212 **3.6 Operations**

1213 The first subsection discusses the required [Core Operations](#).

1214 Subsequent subsections discuss any optional operation that is associated with each of the standard  
1215 capabilities:

- 1216 • [Async Capability](#)
- 1217 • [Batch Capability](#)
- 1218 • [Bulk Capability](#)
- 1219 • [Password Capability](#)
- 1220 • [Reference Capability](#)
- 1221 • [Search Capability](#)
- 1222 • [Suspend Capability](#)
- 1223 • [Updates Capability](#)

### 1224 **3.6.1 Core Operations**

1225 Schema syntax for the SPMLv2 core operations is defined in a schema associated with the  
1226 following XML namespace: `urn:oasis:names:tc:SPML:2:0` [**SPMLv2-CORE**]. The Core XSD  
1227 is included as Appendix A to this document.

1228 A conformant provider must implement all the operations defined in the Core XSD. For more  
1229 information, see the section entitled "[Conformance](#)".

1230 The SPMLv2 core operations include:

- 1231 • a *discovery* operation ([listTargets](#)) on the provider
- 1232 • several *basic* operations ([add](#), [lookup](#), [modify](#), [delete](#)) that *apply to objects on a target*

#### 1233 **3.6.1.1 listTargets**

1234 The listTargets operation enables a requestor to determine the set of targets that a provider makes  
1235 available for provisioning and (the listTargets operation also enables a requestor) to determine the  
1236 set of capabilities that the provider supports for each target.

1237 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"/>

```

1238 **ListTargets must be synchronous.** Because the requestor cannot know (at the time the requestor  
1239 asks to listTargets) whether the provider supports asynchronous execution, the listTargets  
1240 operation must be synchronous.

1241 **ListTargets is not batchable.** Because the requestor cannot know (at the time the requestor asks  
1242 the provider to listTargets) whether the provider supports the batch capability, a requestor must not  
1243 nest a listTargets request in a [batch](#) request.

#### 1244 **3.6.1.1.1 listTargetsRequest (normative)**

1245 A requestor **MUST** send a <listTargetsRequest> to a provider in order to ask the provider to  
1246 declare the set of targets that the provider exposes for provisioning operations.

1247 **Execution.** A <listTargetsRequest> **MUST NOT** specify  
1248 "executionMode='asynchronous' ". A <listTargetsRequest> **MUST** specify  
1249 "executionMode='synchronous' " or (a <listTargetsRequest> **MUST**) omit  
1250 "executionMode".

1251 This is because a requestor SHOULD examine each target definition to see whether the target  
1252 supports the Async Capability *before* making a request that specifies  
1253 "executionMode='asynchronous'" (rather than *assuming* that the provider supports  
1254 asynchronous execution of requested operations). Since a requestor typically must perform the  
1255 listTargets operation only once at the beginning of a session, this restriction should not be too  
1256 onerous.

1257 For more information, see the section titled "[Determining execution mode](#)".

1258 **Profile.** a <listTargetsRequest> MAY specify "profile".

1259 Any profile value MUST be a URI (e.g., of an XML namespace) that identifies an SPML profile.

1260 **No required content.** A <listTargetsRequest> requires no sub-element or XML content.

### 1261 [3.6.1.1.2 listTargetsResponse \(normative\)](#)

1262 A provider that receives a <listTargetsRequest> from a requestor that it trusts  
1263 MUST examine the request and (if the request is valid) return to the requestor a list of the targets  
1264 that the provider exposes for provisioning operations.

- 1265 • If a <listTargetsRequest> does not specify a "profile",  
1266 then the <listTargetsResponse> MUST contain every instance of <target>  
1267 that the provider exposes for provisioning operations *regardless of the profile* or profiles  
1268 for (which the provider supports) that target.
- 1269 • If a <listTargetsRequest> specifies a "profile" that the provider supports,  
1270 then the <listTargetsResponse> MUST contain only instances of <target>  
1271 for which the provider supports the specified profile.
- 1272 • If a <listTargetsRequest> specifies a "profile" that the provider *does not support*,  
1273 then the <listTargetsResponse> MUST specify "status='failure'".  
1274 See the topic named "Error" below within this section.

1275 **Execution.** A provider MUST execute a listTargets operation synchronously. This is because a  
1276 provider must allow the requestor to examine each target definition to see whether the target  
1277 supports the Async Capability (and thus whether the provider might choose to execute a requested  
1278 operation asynchronously) *before* the provider chooses to execute a requested operation  
1279 asynchronously. Since a requestor typically must perform the listTargets operation only once at the  
1280 beginning of a session, this restriction should not be too onerous.

1281 If a requestor specifies "executionMode='asynchronous'", a provider MUST fail the  
1282 operation with "error='unsupportedExecutionMode'".

1283 For more information, see the section titled "[Determining execution mode](#)".

1284 **Status.** A <listTargetsResponse> MUST have a "status" attribute that indicates whether  
1285 the provider successfully processed the request. See the section titled "[Status \(normative\)](#)".

1286 **Error.** If the provider cannot return a list of its targets, then the <listTargetsResponse> MUST  
1287 contain an error attribute that characterizes the failure.

1288 See the general section titled "[Error \(normative\)](#)".

1289 In addition, the <listTargetsResponse> MUST specify an appropriate value of "error" if any  
1290 of the following is true:

1291 • The `<listTargetsRequest>` specifies a "profile" and the provider cannot return at least  
1292 one `<target>` that supports the specified profile. In this case, the  
1293 `<listTargetsResponse>` SHOULD specify "error='unsupportedProfile'".

1294 **Target.** A `<listTargetsResponse>` that specifies "status='success'" MUST contain at  
1295 least one `<target>` element. Each `<target>` SHOULD specify "targetID".

1296 • If the `<listTargetsResponse>` contains only one `<target>`  
1297 then the `<target>` MAY omit "targetID".

1298 • If the `<listTargetsResponse>` contains more than one `<target>`  
1299 then each `<target>` MUST specify "targetID".

1300 Any value of "targetID" MUST identify each target uniquely within the namespace of the  
1301 provider.

1302 **Target profile.** Any `<target>` MAY specify "profile". Any "profile" value MUST be a URI  
1303 (e.g., of an XML namespace) that identifies a specific SPML [profile](#).

1304 If a `<target>` specifies a "profile", then the provider MUST support for that target  
1305 (and for any objects on that target) the behavior that the SPML profile specifies.  
1306 Refer to the documentation of each profile for normative specifics.

1307 **Schema.** A `<target>` MUST contain at least one `<schema>` element. Each `<schema>` element  
1308 MUST contain (or each `<schema>` element MUST refer to) some form of XML Schema that defines  
1309 the structure of XML objects on that target.

1310 **Schema content.** Each `<spml:schema>` element MAY include any number of `<xsd:schema>`  
1311 elements.

1312 • If an `<spml:schema>` element contains no `<xsd:schema>` element,  
1313 then that `<spml:schema>` element MUST have a valid "ref" attribute (see below).

1314 • If an `<spml:schema>` element contains at least one `<xsd:schema>` element,  
1315 then this takes precedence over the value of any "ref" attribute of that `<spml:schema>`.  
1316 In this case, the requestor SHOULD ignore the value of any "ref" attribute.

1317 Each `<xsd:schema>` element (that an `<spml:schema>` element contains)  
1318 MUST include the XML namespace of the schema.

1319 **Schema ref.** Each `<spml:schema>` MAY have a "ref" attribute.  
1320 If an `<spml:schema>` has a "ref" attribute, then:

1321 • The "ref" value MUST be a URI that uniquely *identifies* the schema.  
1322 • The "ref" value MAY be a *location* of a schema document  
1323 (e.g. the physical URL of an XSD file).

1324 A requestor should ignore any "ref" attribute of an `<spml:schema>` element that contains an  
1325 `<xsd:schema>`. (See the topic named "Schema content" immediately above.)

1326 **Supported Schema Entities.** A target MAY declare as part of its `<spml:schema>` the set of  
1327 schema entities for which the target supports the basic SPML operations (i.e., [add](#), [lookup](#), [modify](#)  
1328 and [delete](#)). The target `<spml:schema>` MAY contain any number of  
1329 `<supportedSchemaEntity>` elements. Each `<supportedSchemaEntity>` MUST refer to an  
1330 entity in the target schema. (See the topics named "SupportedSchemaEntity entityName" and  
1331 "SupportedSchemaEntity targetID" below within this section.)

1332 A provider that *explicitly* declares a set of schema entities that a target supports has *implicitly*  
1333 declared that the target supports *only* those schema entities. If a target schema contains at least  
1334 one <supportedSchemaEntity>, then the provider **MUST** support the basic SPML operations  
1335 for (objects on that target that are instances of) any target schema entity to which a  
1336 <supportedSchemaEntity> refers.

1337 A provider that does **not** *explicitly* declare as part of a target at least one schema entity that the  
1338 target supports has *implicitly* declared that the target supports *every* schema entity. If a target  
1339 schema contains no <supportedSchemaEntity>, then the provider **MUST** support the basic  
1340 SPML operations for (objects on that target that are instances of) *any* top-level entity in the target  
1341 schema.

1342 A provider **SHOULD** explicitly declare the set of schema entities that each target supports. In  
1343 general, the syntactic convenience of omitting the declaration of supported schema entities (and  
1344 thereby implicitly declaring that the provider supports all schema entities) does not justify the  
1345 burden that this imposes on each requestor. When a provider omits the declaration of supported  
1346 schema entities, each requestor must determine the set of schema entities that the target supports.  
1347 This process is especially laborious for a requestor that functions without prior knowledge.

1348 **SupportedSchemaEntity entityName.** Each <supportedSchemaEntity> **MUST** refer to an  
1349 entity in the schema (of the target that contains the <supportedSchemaEntity>):

- 1350 • In the XSD Profile [**SPMLv2-Profile-XSD**], each <supportedSchemaEntity> **MUST** specify  
1351 a QName (as the value of its "entityName" attribute).
- 1352 • In the DSMLv2 Profile [**SPMLv2-Profile-DSML**], each <supportedSchemaEntity> **MUST**  
1353 specify the name of an `objectclass` (as the value of its "entityName" attribute).

1354 **SupportedSchemaEntity targetID.** A <supportedSchemaEntity> **SHOULD** specify a  
1355 "targetID".

- 1356 • A provider **MAY** omit "targetID" in any <supportedSchemaEntity>.  
1357 (That is, a provider **MAY** omit the optional "targetID" attribute of  
1358 {SchemaEntityRefType} in a <supportedSchemaEntity> element.)
- 1359 • Any "targetID" in a <supportedSchemaEntity> **MUST** refer to the containing target.  
1360 (That is, the value of any "targetID" attribute that a <supportedSchemaEntity> specifies  
1361 **MUST** match the value of the "targetID" attribute of the <target> element that contains  
1362 the <supportedSchemaEntity> element.)

1363 **SupportedSchemaEntity isContainer.** A <supportedSchemaEntity> **MAY** have an  
1364 "isContainer" attribute that specifies whether an (object that is an) instance of the supported  
1365 schema entity may contain other objects.

- 1366 • If a <supportedSchemaEntity> specifies "isContainer='true'", then a provider  
1367 **MUST** allow a requestor to add an object beneath any instance of the schema entity.
- 1368 • If a <supportedSchemaEntity> specifies "isContainer='false'"  
1369 (or if a <supportedSchemaEntity> does not specify "isContainer"), then a provider  
1370 **MUST NOT** allow a requestor to add an object beneath any instance of the schema entity.

1371 **Capabilities.** A target may also declare a set of capabilities that it supports. Each capability defines  
1372 optional operations or semantics. For general information, see the subsection titled "[Capabilities](#)"  
1373 within the "[Concepts](#)" section.

1374 A <target> element **MAY** contain at most one <capabilities> element. A <capabilities>  
1375 element **MAY** contain any number of <capability> elements.

1376 **Capability.** Each <capability> declares support for exactly one capability:

- 1377 • Each <capability> element MUST specify (as the value of its "namespaceURI" attribute)  
1378 an XML namespace that *identifies* the capability.
- 1379 • Each <capability> element MAY specify (as the value of its "location" attribute) the URL  
1380 of an XML schema that defines any structure that is associated with the capability  
1381 (e.g., an SPML request/response pair that defines an operation—see below).
- 1382 **Capability operations.** An XML schema document that a capability "location" attribute  
1383 specifies MAY define operations. An XML schema document for a capability MUST define any  
1384 operation as a paired request and response such that both of the following are true:
- 1385 • The (XSD type of the) request (directly or indirectly) extends {RequestType}  
1386 • The (XSD type of the) response (directly or indirectly) extends {ResponseType}
- 1387 **Capability appliesTo.** A target may support a capability for *all* of the target's supported schema  
1388 entities or only for *a specific subset* of the target's supported schema entities. Each [capability](#)  
1389 element may specify any number of supported schema entities to which it applies. A capability that  
1390 does not specify a supported schema entity to which it applies must apply to every supported  
1391 schema entity.
- 1392 A <capability> element MAY contain any number of <appliesTo> elements.
- 1393 A <capability> element that contains no <appliesTo> element MUST apply to every schema  
1394 entity that the target supports. If the XML schema for the capability defines an operation, the  
1395 provider MUST support the capability-defined operation for (any object that is instance of) any  
1396 schema entity that the target supports. If the capability implies semantic meaning, then the provider  
1397 MUST apply that semantic meaning to (every object that is an instance of) any schema entity that  
1398 the target supports.
- 1399 **Capability appliesTo entityName.** Each <appliesTo> element MUST have an "entityName"  
1400 attribute that refers to a supported schema entity of the containing target. (See the topic named  
1401 "Supported Schema Entities entityName" earlier in this section.)
- 1402 • In the XSD Profile, each <appliesTo> element MUST specify a QName  
1403 (as the value of its "entityName" attribute).
- 1404 • In the DSMLv2 Profile [**SPMLv2-Profile-DSML**], each <appliesTo> element MUST specify  
1405 the name of an `objectclass` (as the value of its "entityName" attribute).
- 1406 An <appliesTo> element MAY have a "targetID" attribute.
- 1407 • A provider MAY omit "targetID" in any <appliesTo>.  
1408 (That is, a provider MAY omit the optional "targetID" attribute of  
1409 {SchemaEntityRefType} in an <appliesTo> element.)
- 1410 • Any "targetID" MUST refer to the containing target.  
1411 (That is, any "targetID" attribute of an <appliesTo> element  
1412 MUST contain the same value as the "targetID" attribute  
1413 of the <target> element that contains the <appliesTo> element.)
- 1414 **Capability content.** SPMLv2 specifies only the optional <appliesTo> element as content for  
1415 most capability elements. However, a declaration of support for the reference capability is special.
- 1416 **Reference Capability content.** A <capability> element that refers to the Reference Capability  
1417 (i.e., any <capability> element that specifies  
1418 "namespaceURI='urn:oasis:names:tc:SPML:2.0:reference'")  
1419 MUST contain (as open content) at least one <referenceDefinition> element.  
1420 (For normative specifics, please see the topic named "Reference Definition" immediately below.

1421 For background and for general information, please see the section titled "[Reference Capability](#)".  
1422 For Reference Capability XSD, please see Appendix F.)

1423 **ReferenceDefinition.** Each <referenceDefinition> element MUST be an instance of  
1424 {spmlref:ReferenceDefinitionType}. Each reference definition names a type of reference,  
1425 specifies a "from" schema entity and specifies a set of "to" schema entities. Any instance of the  
1426 "from" schema entity may refer to any instance of any "to" schema entity using the type of reference  
1427 that the reference definition names.

1428 **ReferenceDefinition typeOfReference.** Each <referenceDefinition> element MUST have a  
1429 "typeOfReference" attribute *that names the type of reference*.

1430 **ReferenceDefinition schemaEntity.** Each <referenceDefinition> element MUST contain  
1431 exactly one <schemaEntity> sub-element that specifies a "*from*" schema entity for that type of  
1432 reference.

- 1433 • The <schemaEntity> MUST have an "entityName" attribute that refers to a supported  
1434 schema entity of the containing target. (See topic named the "Supported Schema Entities"  
1435 earlier in this section.)
- 1436 • The <schemaEntity> MAY have a "targetID" attribute. Any "targetID" that the  
1437 <schemaEntity> specifies MUST refer to the containing target.  
1438 (That is, any "targetID" value that a <schemaEntity> specifies  
1439 MUST match the value of the "targetID" attribute of the <target> element  
1440 that contains the <referenceDefinition>.)

1441 **ReferenceDefinition canReferTo.** Each <referenceDefinition> element MAY contain any  
1442 number of <canReferTo> sub-elements, each of which specifies a valid "*to*" schema entity. A  
1443 <referenceDefinition> element that contains no <canReferTo> element implicitly declares  
1444 that *any instance of any schema entity on any target* is a valid "to" schema entity.

- 1445 • A <canReferTo> element MUST have an "entityName" attribute that refers to a supported  
1446 schema entity. The value of the "entityName" attribute MUST be the name of a top-level  
1447 entity that is valid in the schema.
- 1448 • A <canReferTo> element SHOULD have a "targetID" attribute.
  - 1449 - If the <listTargetsResponse> contains only one <target>,  
1450 then any <canReferTo> element MAY omit "targetID".
  - 1451 - If the <listTargetsResponse> contains more than one <target>,  
1452 then any <canReferTo> element MUST specify "targetID".
  - 1453 - If the <canReferTo> element specifies "targetID",  
1454 then the "entityName" attribute (of the <canReferTo> element)  
1455 MUST refer to a supported schema entity of the specified target  
1456 (i.e., the <target> whose "targetID" value matches  
1457 the "targetID" value that the <canReferTo> element specifies).
  - 1458 - If the <canReferTo> element does not specify "targetID",  
1459 then the "entityName" attribute (of the <canReferTo> element)  
1460 MUST refer to a supported schema entity of the containing target  
1461 (i.e., the <target> that contains the <referenceDefinition>).

1462 **ReferenceDefinition referenceDataType.** Each <referenceDefinition> element MAY  
1463 contain any number of <referenceDataType> sub-elements, each of which specifies a *schema*  
1464 *entity that is a valid structure for reference data*. A <referenceDefinition> element that



- 1465 contains no <referenceDataType> element implicitly declares that an instance of that type of  
1466 reference will never contain reference data.
- 1467 • A <referenceDataType> element MUST have an "entityName" attribute that refers to a  
1468 supported schema entity. The value of the "entityName" attribute MUST be the name of a  
1469 top-level entity that is valid in the schema.
  - 1470 • A <referenceDataType> element SHOULD have a "targetID" attribute.
    - 1471 - If the <listTargetsResponse> contains only one <target>,  
1472 then any <referenceDataType> element MAY omit "targetID".
    - 1473 - If the <listTargetsResponse> contains more than one <target>,  
1474 then any <referenceDataType> element MUST specify "targetID".
    - 1475 - If the <referenceDataType> element specifies "targetID",  
1476 then the "entityName" attribute (of the <canReferTo> element)  
1477 MUST refer to a supported schema entity of the specified target  
1478 (i.e., the <target> whose "targetID" value matches  
1479 the "targetID" value that the <referenceDataType> element specifies).
    - 1480 - If the <referenceDataType> element does not specify "targetID",  
1481 then the "entityName" attribute (of the <canReferTo> element)  
1482 MUST refer to a supported schema entity of the containing target  
1483 (i.e., the <target> that contains the <referenceDefinition>).

### 1484 3.6.1.1.3 *listTargets Examples (non-normative)*

1485 In the following example, a requestor asks a provider to list the [targets](#) that the provider exposes for  
1486 provisioning operations.

```
<listTargetsRequest/>
```

1487 The provider returns a <listTargetsResponse>. The "status" attribute of the  
1488 <listTargetsResponse> element indicates that the listTargets request was successfully  
1489 processed. The <listTargetsResponse> contains two <target> elements. Each <target>  
1490 describes an endpoint that is available for provisioning operations.

1491 The requestor did not specify a profile, but both targets specify the XSD profile **[SPMLv2-Profile-  
1492 XSD]**. The requestor must observe the conventions that the XSD profile specifies in order to  
1493 manipulate an object on either target.

1494 If the requestor had specified the DSML profile, then the response would have contained a different  
1495 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>

```

1496 This example <listTargetsResponse> contains two instances of <target> that are named  
1497 target1 and target2. Each of these targets contains a simple schema.

1498 The schema for target1 defines two entities: Account and Group. The schema for target1  
1499 declares each of these entities as a supported schema entity. The provider declares that target1  
1500 supports the Bulk capability and Search capability for both Account and Group. The provider also  
1501 declares that target1 supports the Password, Suspend, and Reference capabilities for Account.

1502 The schema for target2 defines three entities: Person, Organization and  
1503 OrganizationalUnit. The schema for target2 declares each of these entities as a supported  
1504 schema entity. The provider declares that target2 supports the Bulk capability and Search  
1505 capability for all three schema entities. The provider also declares that target2 supports the  
1506 Password, Suspend, and Reference capabilities for instances of Person (but not for instances of  
1507 Organization or OrganizationalUnit).

1508 **Reference Definitions.** Within target1's declaration of the Reference Capability for Account,  
1509 the provider also declares two types of references: owner and memberOf. The provider declares  
1510 that an instance of Account on target1 may refer to an instance of Person on target2 as its  
1511 owner. An instance of Account on target1 may also use a memberOf type of reference to refer  
1512 to an instance of Group on target1.

1513 Within `target2`'s declaration of the Reference Capability for `Person`, the provider declares that a  
1514 `Person` on `target2` may own an `Account` on `target1`. (That is, an instance of `Person` on  
1515 `target2` may use an "owns" type of reference to refer to an instance of `Account` on `target1`.)  
1516 Note that the "owns" type of reference *may be* (but is not necessarily) an inverse of the "owner"  
1517 type of reference. For more information, please see the section titled "Reference Capability".

1518 **NOTE:** Subsequent examples within this section will build on this example, using the target  
1519 definitions returned in this example. Examples will also build upon each other. An object that is  
1520 created in the example of the add operation will be modified or deleted in later examples.

### 1521 3.6.1.2 add

1522 The add operation enables a requestor to create a new object on a target and (optionally) to bind  
1523 the object beneath a specified parent object (thus forming a hierarchy of containment).

1524 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"/>

```

### 1525 3.6.1.2.1 *addRequest* (normative)

1526 A requestor **MUST** send an `<addRequest>` to a provider in order to (ask the provider to) create a  
1527 new object.

1528 **Execution.** A `<addRequest>` **MAY** specify "executionMode".  
1529 See the section titled "[Determining execution mode](#)".

1530 **TargetID.** An `<addRequest>` **SHOULD** specify "targetID".

- 1531 • If the provider exposes only one target in its `<listTargetsResponse>`,
- 1532 then a requestor **MAY** omit the "targetID" attribute of an `<addRequest>`.
- 1533 • If the provider exposes more than one target in its `<listTargetsResponse>`,
- 1534 then a requestor **MUST** specify the "targetID" attribute of an `<addRequest>`.
- 1535 Any "targetID" value must specify a valid target. (That is, the value of any "targetID" in
- 1536 an `<addRequest>` **MUST** match the "targetID" of a `<target>` that is contained in the
- 1537 provider's `<listTargetsResponse>`.)

1538 **psoID.** An `<addRequest>` **MAY** contain a `<psoID>`. (A requestor supplies `<psoID>` in order to  
1539 specify an identifier for the new object. See the section titled "[PSO Identifier \(normative\)](#)".)

1540 **ContainerID.** An `<addRequest>` **MAY** contain a `<containerID>`. (A requestor supplies  
1541 `<containerID>` in order to specify an existing object under which the new object should be  
1542 bound.)

- 1543 • A requestor that wants to bind a new object *in the top-level namespace of a target*  
 1544 MUST NOT supply <containerID>.
- 1545 • A requestor that wants to bind a new object *beneath a specific object on a target*  
 1546 MUST supply <containerID>. Any <containerID> must identify an existing object.  
 1547 (That is, the content of <containerID> in an <addRequest> must match the <psoid> of an  
 1548 object that already exists on the target.)
- 1549 **Data.** An <addRequest> MUST contain a <data> element that supplies initial content for the new  
 1550 object. A <data> element MUST contain only elements and attributes defined by the target  
 1551 schema as valid for the schema entity of which the object to be added is an instance.
- 1552 **CapabilityData.** An <addRequest> element MAY contain any number of <capabilityData>  
 1553 elements. (Each <capabilityData> element contains data specific to a single capability. Each  
 1554 <capabilityData> element may contain any number of items of capability-specific data.  
 1555 Capability-specific data need not be defined by the target schema as valid for schema entity of  
 1556 which the object to be added is an instance.  
 1557 See the section titled "[CapabilityData in a Request \(normative\)](#)".
- 1558 **ReturnData.** An <addRequest> MAY have a "returnData" attribute that tells the provider  
 1559 which types of data to include in the provider's response.
- 1560 • A requestor that wants the provider to return *nothing* of the added object  
 1561 MUST specify "returnData='nothing'".
- 1562 • A requestor that wants the provider to return *only the identifier* of the added object  
 1563 MUST specify "returnData='identifier'".
- 1564 • A requestor that wants the provider to return the identifier of the added object  
 1565 *plus the XML representation of the object (as defined in the schema of the target)*  
 1566 MUST specify "returnData='data'".
- 1567 • A requestor that wants the provider to return the identifier of the added object  
 1568 *plus the XML representation of the object (as defined in the schema of the target)*  
 1569 *plus any capability-specific data that is associated with the object*  
 1570 MAY specify "returnData='everything'" or MAY omit the "returnData" attribute  
 1571 (since "returnData='everything'" is the default).

### 1572 [3.6.1.2 addResponse \(normative\)](#)

- 1573 A provider that receives an <addRequest> from a requestor that the provider trusts MUST  
 1574 examine the content of the <addRequest>. If the request is valid, the provider MUST create the  
 1575 requested object under the specified parent (i.e., target or container object) if it is possible to do so.
- 1576 **PSO Identifier.** The provider MUST create the object with any <psoid> that the <addRequest>  
 1577 supplies. If the provider cannot create the object with the specified <psoid> (e.g., because the  
 1578 <psoid> is not valid or because an object that already exists has that <psoid>), then the provider  
 1579 must fail the request. See the topic named "Error" below within this section.
- 1580 **Data.** The provider MUST create the object with any XML element or attribute contained by the  
 1581 <data> element in the <addRequest>.
- 1582 **CapabilityData.** The provider SHOULD associate with the created object the content of each  
 1583 <capabilityData> that the <addRequest> contains. The "mustUnderstand" attribute of  
 1584 each <capabilityData> indicates whether the provider MUST process the content of the  
 1585 <capabilityData> *as the corresponding capability specifies*. See the sections titled  
 1586 "[CapabilityData in a Request \(normative\)](#)" and "[CapabilityData Processing \(normative\)](#)".

1587 Also see the section titled "[CapabilityData Errors \(normative\)](#)".

1588 **Execution.** If an `<addRequest>` does not specify a type of execution, a provider MUST choose a  
1589 type of execution for the requested operation.  
1590 See the section titled "[Determining execution mode](#)".

1591 **Response.** The provider must return to the requestor an `<addResponse>`.

1592 **Status.** The `<addResponse>` MUST have a `"status"` attribute that indicates whether the  
1593 provider successfully created the requested object. See the section titled "[Status \(normative\)](#)".

1594 **PSO and ReturnData.** If the provider successfully created the requested object, the  
1595 `<addResponse>` MUST contain an `<pso>` element that contains the (XML representation of the)  
1596 newly created object.

- 1597 • A `<pso>` element MUST contain a `<psoID>` element.  
1598 The `<psoID>` element MUST contain the identifier of the newly created object.  
1599 See the section titled "[PSO Identifier \(normative\)](#)".
- 1600 - If the `<addRequest>` supplies a `<psoID>`, then `<psoID>` of the newly created object  
1601 MUST match the `<psoID>` supplied by the `<addRequest>`.  
1602 (See the topic named "PSO Identifier" above within this section.)
- 1603 - If the `<addRequest>` does not supply `<psoID>`, the provider must generate a `<psoID>`  
1604 that uniquely identifies the newly created object.
- 1605 • A `<pso>` element MAY contain a `<data>` element.
  - 1606 - If the `<addRequest>` specified `"returnData=' identifier' "`  
1607 then the `<pso>` MUST NOT contain a `<data>` element.
  - 1608 - Otherwise, if the `<addRequest>` specified `"returnData=' data' "`  
1609 or (if the `<addRequest>` specified) `"returnData=' everything' "`  
1610 or (if the `<addRequest>`) omitted the `"returnData"` attribute,  
1611 then the `<pso>` MUST contain exactly one `<data>` element that contains the XML  
1612 representation of the object.  
1613 This XML must be valid according to the schema of the target for the schema entity of  
1614 which the newly created object is an instance.
- 1615 • A `<pso>` element MAY contain any number of `<capabilityData>` elements. Each  
1616 `<capabilityData>` element contains a set of *capability-specific data* that is associated with  
1617 the newly created object (for example, a *reference* to another object).  
1618 See the section titled "[CapabilityData in a Response \(normative\)](#)".
  - 1619 - If the `<addRequest>` `"returnData=' identifier' "`  
1620 or (if the `<addRequest>` specified) `"returnData=' data' "`  
1621 then the `<addResponse>` MUST NOT contain a `<capabilityData>` element.
  - 1622 - Otherwise, if the `<addRequest>` specified `"returnData=' everything' "`  
1623 or (if the `<addRequest>`) omitted the `"returnData"` attribute  
1624 then the `<addResponse>` MUST contain a `<capabilityData>` element for each set of  
1625 capability-specific data that is associated with the newly created object.

1626 **Error.** If the provider cannot create the requested object, the `<addResponse>` MUST contain an  
1627 `"error"` attribute that characterizes the failure. See the general section titled "[Error \(normative\)](#)".

1628 In addition, the `<addResponse>` MUST specify an appropriate value of `"error"` if any of the  
1629 following is true:

- 1630 • An <addRequest> specifies "targetID" but the value of "targetID" does not identify a  
1631 target that the provider supports.  
1632 In this case, the <addResponse> SHOULD specify "error='noSuchIdentifier'".
  - 1633 • An <addRequest> specifies "targetID" and (the <addRequest> also) contains  
1634 <containerID> but the value of the "targetID" attribute in the <addRequest> does not  
1635 match the value of the "targetID" attribute in the <containerID>.  
1636 In this case, the <addResponse> SHOULD specify "error='malformedRequest'".
  - 1637 • An <addRequest> contains <containerID> but the content of <containerID> does not  
1638 identify an object that exists. (That is, <containerID> does not match the <psoid> of an  
1639 object that exists.)  
1640 In this case, the <addResponse> SHOULD specify "error='noSuchIdentifier'".
  - 1641 • An <addRequest> contains <containerID> but the <supportedSchemaEntity> (of  
1642 which <containerID> identifies an instance) does not specify "isContainer='true'".  
1643 In this case, the <addResponse> SHOULD specify "error='invalidContainment'".
  - 1644 • An <addRequest> contains <containerID> but the target does not allow the specified  
1645 parent object to contain the object to be created.  
1646 In this case, the <addResponse> SHOULD specify "error='invalidContainment'".
  - 1647 • An <addRequest> supplies <psoid> but the <psoid> element is not valid.  
1648 In this case, the <addResponse> SHOULD specify "error='invalidIdentifier'".
  - 1649 • An <addRequest> supplies <psoid> but an object with that <psoid> already exists.  
1650 In this case, the <addResponse> SHOULD specify "error='alreadyExists'".
  - 1651 • The <data> element is missing an element or attribute that is required (according to the  
1652 schema of the target) for the object to be added.
  - 1653 • A <capabilityData> element specifies "mustUnderstand='true'" and the provider  
1654 cannot associate the content of the <capabilityData> with the object to be created.
- 1655 The provider MAY return an error if:
- 1656 • The <data> element contains data that the provider does not recognize as *valid according to*  
1657 *the target schema* for the type of object to be created.
  - 1658 • The provider does not recognize the content of a <capabilityData> element as specific to  
1659 any capability that the target supports (for the schema entity of which the object to be created is  
1660 an instance).
- 1661 Also see the section titled "[CapabilityData Errors \(normative\)](#)".

### 1662 3.6.1.2.3 add Examples (non-normative)

1663 In the following example, a requestor asks a provider to add a new person. The requestor specifies  
1664 the attributes required for the `Person` schema entity (`cn`, `firstName`, `lastName` and `fullName`).  
1665 The requestor also supplies an optional `email` address for the person. This example assumes that  
1666 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>
```

1667 The provider returns an <addResponse> element. The "status" attribute of the  
1668 <addResponse> element indicates that the add request was successfully processed. The  
1669 <addResponse> contains a <pso>. The <pso> contains a <psoID> that identifies the newly  
1670 created object. The <pso> also contains a <data> element that contains the schema-defined XML  
1671 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>
```

1672 Next, the requestor asks a provider to add a new account. The requestor specifies a name for the  
1673 account. The requestor also specifies references to a Group that resides on target1 and to a  
1674 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>
```

1675 The provider returns an <addResponse> element. The "status" attribute of the  
1676 <addResponse> element indicates that the add operation was successfully processed. The  
1677 <addResponse> contains a <pso> that contains a <psoID> that identifies the newly created  
1678 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>

```

1679 **3.6.1.3 lookup**

1680 The lookup operation enables a requestor to *obtain the XML that represents an object* on a target.

1681 The lookup operation also obtains any *capability-specific data* that is associated with the object.

1682 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"/>

```

### 1683 3.6.1.3.1 *lookupRequest (normative)*

1684 A requestor MUST send a <lookupRequest> to a provider in order to (ask the provider to) return  
1685 (the XML that represents) an existing object.

1686 **Execution.** A <lookupRequest> MAY specify "executionMode".  
1687 See the section titled "[Determining execution mode](#)".

1688 In general, a requestor SHOULD NOT specify "executionMode='asynchronous' ". The  
1689 reason for this is that the result of a lookup should reflect the current state of a target object. If a  
1690 lookup operation is executed asynchronously then other operations are more likely to intervene.

1691 **PsoID.** A <lookupRequest> MUST contain exactly one <psoID> that identifies the object to  
1692 lookup (i.e., the object for which the provider should return the XML representation). The <psoID>  
1693 MUST identify an object that exists on a target.

1694 **ReturnData.** A <lookupRequest> MAY have a "returnData" attribute that tells the provider  
1695 which subset of (the XML representation of) a <pso> to include in the provider's response.

- 1696 • A requestor that wants the provider to return *nothing* of a requested object  
1697 MUST specify "returnData='nothing' ".
- 1698 • A requestor that wants the provider to return *only the identifier* of a requested object  
1699 MUST specify "returnData='identifier' ".
- 1700 • A requestor that wants the provider to return the identifier of a requested object  
1701 *plus the XML representation of the object (as defined in the schema of the target)*  
1702 MUST specify "returnData='data' ".
- 1703 • A requestor that wants the provider to return the identifier of a requested object  
1704 *plus the XML representation of the object (as defined in the schema of the target)*  
1705 *plus any capability-specific data that is associated with the object*  
1706 MAY specify "returnData='everything' " or MAY omit the "returnData" attribute  
1707 (since "returnData='everything' " is the default).

### 1708 3.6.1.3.2 *lookupResponse (normative)*

1709 A provider that receives a <lookupRequest> from a requestor that the provider trusts MUST  
1710 examine the content of the <lookupRequest>. If the request is valid, the provider MUST return  
1711 (the XML that represents) the requested object if it is possible to do so.

1712 **Execution.** If an <lookupRequest> does not specify "executionMode", the provider MUST  
1713 choose a type of execution for the requested operation.  
1714 See the section titled "[Determining execution mode](#)".

1715 A provider SHOULD execute a lookup operation synchronously if it is possible to do so. The reason  
1716 for this is that the result of a lookup should reflect the current state of a target object. If a lookup  
1717 operation is executed asynchronously then other operations are more likely to intervene.

1718 **Response.** The provider must return to the requestor a <lookupResponse>.

1719 **Status.** The <lookupResponse> must have a "status" that indicates whether the provider  
1720 successfully returned each requested object. See the section titled "[Status \(normative\)](#)".

1721 **PSO and ReturnData.** If the provider successfully returned the requested object, the  
1722 <lookupResponse> MUST contain an <psso> element for the requested object. Each <psso>  
1723 contains the subset of (the XML representation of) a requested object that the "returnData"  
1724 attribute of the <lookupRequest> specified. By default, each <psso> contains the entire (XML  
1725 representation of an) object.

1726 • A <psso> element MUST contain a <pssoID> element.  
1727 The <pssoID> element MUST contain the identifier of the requested object.  
1728 See the section titled "[PSO Identifier \(normative\)](#)".

1729 • A <psso> element MAY contain a <data> element.

1730 - If the <lookupRequest> specified "returnData=' identifier' ",  
1731 then the <psso> MUST NOT contain a <data> element.

1732 - Otherwise, if the <lookupRequest> specified "returnData=' data' "  
1733 or (if the <lookupRequest> specified) "returnData=' everything' "  
1734 or (if the <lookupRequest>) omitted the "returnData" attribute  
1735 then the <data> element MUST contain the XML representation of the object.  
1736 This XML must be valid according to the schema of the target for the schema entity of  
1737 which the newly created object is an instance.

1738 • A <psso> element MAY contain any number of <capabilityData> elements.  
1739 Each <capabilityData> element MUST contain all the data (that are associated with the  
1740 object and) that are specific to the capability that the <capabilityData> specifies as  
1741 "capabilityURI". For example, a <capabilityData> that refers to the Reference  
1742 Capability (i.e., a <capabilityData> that specifies  
1743 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference' ")  
1744 must contain at least one *reference* to another object.  
1745 See the section titled "[CapabilityData in a Response \(normative\)](#)".

1746 - If the <lookupRequest> specified "returnData=' identifier' "  
1747 or (if the <lookupRequest> specified) "returnData=' data' "  
1748 then the <psso> MUST NOT contain a <capabilityData> element.

1749 - Otherwise, if the <lookupRequest> specified "returnData=' everything' "  
1750 or (if the <lookupRequest>) omitted the "returnData" attribute,  
1751 then the <psso> MUST contain a <capabilityData> element  
1752 for each set of capability-specific data that is associated with the requested object

1753 (and that is specific to a capability that the target supports for the schema entity  
1754 of which the requested object is an instance).

1755 **Error.** If the provider cannot return the requested object, the `<lookupResponse>` must have an  
1756 "error" attribute that characterizes the failure. See the general section titled "[Error \(normative\)](#)".

1757 In addition, the `<lookupResponse>` MUST specify an appropriate value of "error" if any of the  
1758 following is true:

- 1759 • A `<lookupRequest>` contains no `<psoID>`.
- 1760 • A `<lookupRequest>` contains a `<psoID>` that does not identify an object that exists on a  
1761 target.

1762 The provider MAY return an error if:

- 1763 • A `<psoID>` contains data that the provider does not recognize.

### 1764 [3.6.1.3.3 lookup Examples \(non-normative\)](#)

1765 In the following example, a requestor asks a provider to return the `Person` object from the [add](#)  
1766 [examples](#) above. The requestor specifies the `<psoID>` for the `Person` object.

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

1767 The provider returns a `<lookupResponse>` element. The "status" attribute of the  
1768 `<lookupResponse>` element indicates that the lookup request was successfully processed. The  
1769 `<lookupResponse>` contains a `<pso>` element that contains the requested object.

1770 The `<pso>` element contains a `<psoID>` element that contains the PSO Identifier. The `<pso>` also  
1771 contains a `<data>` element that contains the XML representation of the object (according to the  
1772 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>
```

1773 Next, the requestor asks a provider to return the `Account` object from the [add examples](#) above.  
1774 The requestor specifies a `<psoID>` for the `Account` object.

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

1775 The provider returns a `<lookupResponse>` element. The "status" attribute of the  
1776 `<lookupResponse>` element indicates that the lookup request was successfully processed. The  
1777 `<lookupResponse>` contains a `<pso>` element that contains the requested object.

1778 The <ps> element contains a <psID> element that uniquely identifies the object. The <ps>  
1779 also contains a <data> element that contains the XML representation of the object (according to  
1780 the schema of the target).

1781 In this example, the <ps> element also contains a <capabilityData> element. The  
1782 <capabilityData> element in turn contains two <reference> elements. The lookup operation  
1783 automatically includes capability-specific data (such as these two reference elements) if the  
1784 schema for the target declares that it supports the reference capability (for the schema entity of  
1785 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>
```

1786 To illustrate the effect of the "returnData" attribute, let's reissue the previous request and  
1787 specify a value of "returnData" other than the default (which is  
1788 "returnData='everything'"). First, assume that the requestor specifies  
1789 "returnData='identifier'".

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

1790 The response specifies "status='success'" which indicates that the lookup operation  
1791 succeeded and that the requested object exists. Since the request specifies  
1792 "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>
```

1793 Next assume that the requestor specifies "returnData='data'".

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

1794 Since the request specifies "return='data'", the <ps> in the response contains the <psID>  
1795 and <data> but no <capabilityData> element. Specifying "return='data'" returns the  
1796 XML representation of the object as defined in the schema for the target but *suppresses capability-*  
1797 *specific data*.

1798 Specifying "return='data'" is advantageous if the requestor is not interested in capability-  
1799 specific data. Omitting capability-specific data may reduce the amount of work that the provider

1800 must do in order to build the <lookupResponse>. Reducing the size of the response should also  
1801 reduce the network traffic that is required in order to transmit the response. Omitting capability-  
1802 specific data may also reduce the amount of XML parsing work that the requestor must perform in  
1803 order to process the response.

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

#### 1804 **3.6.1.4 modify**

1805 The modify operation enables a requestor to *change an object* on a target. The modify operation  
1806 can change the *schema-defined component* of an object, any *capability-specific data* that is  
1807 associated with the object, or *both*.

1808 **Modify can change PSO Identifier.** One important subtlety is that a modify operation may change  
1809 the identifier of the modified object. For example, assume that a provider exposes the  
1810 Distinguished Name (DN) as the identifier of each object on a target that represents a directory  
1811 service. In this case, modifying the object's Common Name (CN) or moving the object beneath a  
1812 different Organizational Unit (OU) would change the object's DN and therefore its PSO-ID.

1813 A provider should expose an immutable identifier as the PSO-ID of each object. In the case of a  
1814 target that represents a directory service, an immutable identifier could be a Globally Unique  
1815 Identifier (GUID) that is managed by the directory service or it could be any form of unique identifier  
1816 that is managed by the provider.

1817 For normative specifics, please see the section titled "[PSO Identifier \(normative\)](#)".

1818 **Modifying capability-specific data.** Any capability may imply capability-specific data (where the  
1819 target supports that capability for the schema entity of which the object is an instance). However,  
1820 many capabilities do not. Of the standard capabilities that SPMLv2 defines, only the [Reference](#)  
1821 [Capability](#) implies capability-specific data.

1822 The default processing for capability-specific data is to treat the content of each  
1823 <capabilityData> as opaque. See the section titled "[CapabilityData](#)".

1824 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"/>

```

#### 1825 **3.6.1.4.1** *modifyRequest (normative)*

1826 A requestor **MUST** send a <modifyRequest> to a provider in order to (ask the provider to) modify  
1827 an existing object.

1828 **Execution.** A <modifyRequest> **MAY** specify "executionMode".

1829 See the section titled "[Determining execution mode](#)".

1830 **PsoID.** A <modifyRequest> MUST contain exactly one <psoID>. A <psoID> MUST identify an  
1831 object that exists on a target that is exposed by the provider.

1832 **ReturnData.** A <modifyRequest> MAY have a "returnData" attribute that tells the provider  
1833 which subset of (the XML representation of) each modified <pso> to include in the provider's  
1834 response.

- 1835 • A requestor that wants the provider to return *nothing* of the modified object  
1836 MUST specify "returnData='nothing'".
- 1837 • A requestor that wants the provider to return *only the identifier* of the modified object  
1838 MUST specify "returnData='identifier'".
- 1839 • A requestor that wants the provider to return the identifier of the modified object  
1840 *plus the XML representation of the object (as defined in the schema of the target)*  
1841 MUST specify "returnData='data'".
- 1842 • A requestor that wants the provider to return the identifier of the modified object  
1843 *plus the XML representation of the object (as defined in the schema of the target)*  
1844 *plus any capability-specific data that is associated with the object*  
1845 MAY specify "returnData='everything'" or MAY omit the "returnData" attribute  
1846 (since "returnData='everything'" is the default).

1847 **Modification.** A <modifyRequest> MUST contain at least one <modification>. A  
1848 <modification> describes a set of changes to be applied (to the object that the <psoID>  
1849 identifies). A <modification> MUST have a "modificationMode" that specifies the type of  
1850 change as one of 'add', 'replace' or 'delete'.

1851 A requestor MAY specify a change to a schema-defined element or attribute of the object to be  
1852 modified. A requestor MAY specify any number of changes to capability-specific data associated  
1853 with the object to be modified.

1854 A requestor MUST use a <component> element to specify a schema-defined element or attribute  
1855 of the object to be modified. A requestor MUST use a <capabilityData> element to describe  
1856 each change to a capability-specific data element that is associated with the object to be modified.

1857 A <modification> element MUST contain a <component> element or (the <modification>  
1858 MUST contain) at least one <capabilityData> element. A <modification> element MAY  
1859 contain a <component> element *as well as* one or more <capabilityData> elements.

1860 **Modification component.** The <component> sub-element of a <modification> specifies a  
1861 schema-defined element or attribute of the object that is to be modified. This is an instance of  
1862 {SelectionType}, which occurs in several contexts within SPMLv2.  
1863 See the section titled "[SelectionType in a Request \(normative\)](#)".

1864 **Modification data.** A requestor MUST specify as the content of the <data> sub-element of a  
1865 <modification> any content or *value* that is to be added to, replaced within, or deleted from the  
1866 element or attribute that the <component> (sub-element of the <modification>) specifies.

1867 **Modification capabilityData.** A requestor MAY specify any number of <capabilityData>  
1868 elements within a <modification> element. Each <capabilityData> element specifies  
1869 *capability-specific data* (for example, *references* to other objects) for the object to be modified.  
1870 Because the {CapabilityDataType} is an {ExtensibleType}, a <capabilityData>  
1871 element may validly contain any XML element or attribute. The <capabilityData> element  
1872 SHOULD contain elements that the provider will recognize as specific to a capability that the target  
1873 supports (for the schema entity of which the object to be modified is an instance).  
1874 See the section titled "[CapabilityData in a Request \(normative\)](#)".



1875 **3.6.1.4.2 modifyResponse (normative)**

1876 A provider that receives a <modifyRequest> from a requestor that the provider trusts MUST  
1877 examine the content of the <modifyRequest>. If the request is valid, the provider MUST apply  
1878 each requested <modification> (to the object that is identified by the <psoid> of the  
1879 <modifyRequest>) if it is possible to do so.

1880 For normative specifics related to processing any <capabilityData> within a  
1881 <modification>, please see the section titled "[CapabilityData Processing \(normative\)](#)".

1882 **Execution.** If a <modifyRequest> does not specify "executionMode", the provider MUST  
1883 choose a type of execution for the requested operation.  
1884 See the section titled "[Determining execution mode](#)".

1885 **Response.** The provider must return to the requestor a <modifyResponse>.

1886 **Status.** The <modifyResponse> must have a "status" attribute that indicates whether the  
1887 provider successfully applied the requested modifications to each identified object.  
1888 See the section titled "[Status \(normative\)](#)".

1889 **PSO and ReturnData.** If the provider successfully modified the requested object, the  
1890 <modifyResponse> MUST contain an <psoid> element. The <psoid> contains the subset of (the  
1891 XML representation of) a requested object that the "returnData" attribute of the  
1892 <lookupRequest> specified. By default, the <psoid> contains the entire (XML representation of  
1893 the) modified object.

- 1894 • A <psoid> element MUST contain a <psoid> element.  
1895 The <psoid> element MUST contain the identifier of the requested object.  
1896 See the section titled "[PSO Identifier \(normative\)](#)".
- 1897 • A <psoid> element MAY contain a <data> element.
  - 1898 - If the <modifyRequest> specified "returnData=' identifier' ",  
1899 then the <psoid> MUST NOT contain a <data> element.
  - 1900 - Otherwise, if the <modifyRequest> specified "returnData=' data' "  
1901 or (if the <modifyRequest> specified) "returnData=' everything' "  
1902 or (if the <modifyRequest>) omitted the "returnData" attribute  
1903 then the <data> element MUST contain the XML representation of the object.  
1904 This XML must be valid according to the schema of the target for the schema entity of  
1905 which the newly created object is an instance.
- 1906 • A <psoid> element MAY contain any number of <capabilityData> elements. Each  
1907 <capabilityData> element contains a set of *capability-specific data* that is associated with  
1908 the newly created object (for example, a *reference* to another object).  
1909 See the section titled "[CapabilityData in a Response \(normative\)](#)".
  - 1910 - If the <modifyRequest> specified "returnData=' identifier' "  
1911 or (if the <modifyRequest> specified) "returnData=' data' "  
1912 then the <modifyResponse> MUST NOT contain a <capabilityData> element.
  - 1913 - Otherwise, if the <modifyRequest> specified "returnData=' everything' "  
1914 or (if the <modifyRequest>) omitted the "returnData" attribute,  
1915 then the <modifyResponse> MUST contain a <capabilityData> element for each set  
1916 of capability-specific data that is associated with the requested object  
1917 (and that is specific to a capability that the target supports for the schema entity of which  
1918 the requested object is an instance).

1919 **Error.** If the provider cannot modify the requested object, the `<modifyResponse>` must have an  
1920 "error" attribute that characterizes the failure. See the general section titled "[Error \(normative\)](#)".

1921 In addition, a `<modifyResponse>` MUST specify an appropriate value of "error" if any of the  
1922 following is true:

- 1923 • The `<modifyRequest>` contains a `<modification>` for which there is no corresponding  
1924 `<psoID>`.
- 1925 • A `<modification>` contains neither a `<component>` nor a `<capabilityData>`.
- 1926 • A `<component>` is empty (that is, a `<component>` element has no content).
- 1927 • A `<component>` specifies an element or attribute that is not valid (according to the schema of  
1928 the target) for the type of object to be modified.

1929 The provider MAY return an error if:

- 1930 • A `<component>` contains data that the provider does not recognize as specifying an XML  
1931 element or attribute that is *valid according to the target schema* for the type of object to be  
1932 modified.
- 1933 • A `<capabilityData>` element contains data that the provider does not recognize as specific  
1934 to the capability that its "capabilityURI" attribute identifies.

1935 In addition, see the section titled "[SelectionType Errors \(normative\)](#)" as well as the section titled  
1936 "[CapabilityData Errors \(normative\)](#)".

### 1937 [3.6.1.4.3 modify Examples \(non-normative\)](#)

1938 In the following example, a requestor asks a provider to modify the email address for an existing  
1939 `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>
```

1940 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
1941 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
1942 `<pso>` element of the `<modifyResponse>` contains the XML representation of the modified  
1943 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>
```

1944 In the following example, a requestor asks a provider to modify the same `Person` object, adding a  
1945 reference to an `Account` that the `Person` owns. (Since the request is to add capability-specific  
1946 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>
```

1947 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
1948 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
1949 `<pso>` element of the `<modifyResponse>` shows that the provider has added (the  
1950 `<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>
```

1951

1952 **Modifying capabilityData.** Of the standard capabilities defined by SPMLv2, only the [Reference](#)  
1953 [Capability](#) associates capability-specific data with an object. We must therefore imagine a custom  
1954 capability "foo" in order to illustrate the *default processing* of capability data. (We illustrate the  
1955 handling of references further below.)

1956 In this example, the requestor wishes to replace any existing data foo-specific data that is  
1957 associated with a specific `Account` with a new `<foo>` element. The fact that each  
1958 `<capabilityData>` omits the "mustUnderstand" flag indicates that the requestor will accept  
1959 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>

```

1960 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
 1961 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
 1962 `<pso>` element of the `<modifyResponse>` shows that any capability data that is specific to the  
 1963 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>

```

1964 The requestor next adds another `<foo>` element to the set of foo-specific data that is associated  
 1965 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>

```

1966 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
 1967 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
 1968 `<pso>` element of the `<modifyResponse>` shows that the content of the foo-specific  
 1969 `<capabilityData>` in the `<modification>` has been appended to the previous content of the  
 1970 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>

```

- 1971 Finally, our requestor deletes any foo-specific capability data from the Account. The  
1972 <capabilityData> element does not need any content. The content of <capabilityData> is  
1973 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>

```

- 1974 The provider returns a <modifyResponse> element. The "status" attribute of the  
1975 <modifyResponse> element indicates that the modify request was successfully processed. The  
1976 <pso> element of the <modifyResponse> shows that the foo-specific <capabilityData> has  
1977 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>

```

1978

1979 **Modifying a reference.** The previous topic illustrates the default processing of capability data. The  
1980 Reference Capability specifies enhanced behavior for the modify operation.  
1981 See the section titled "[Reference CapabilityData Processing \(normative\)](#)".

1982 In this example, the requestor wishes to change the owner of an `Account` from "2244" (which is  
1983 the `<psoID>` of "Person:joebob") to "2245" (which is the `<psoID>` of "Person:billybob").

1984 Since SPMLv2 does not specify any mechanism to define the cardinality of a type of reference, a  
1985 requestor should not assume that a provider enforces any specific cardinality for any type of  
1986 reference. For a general discussion of the issues surrounding references, see the section titled  
1987 "[Reference Capability](#)".

1988 Assume that each account should have at most one owner. If the requestor could trust the provider  
1989 to enforce this, and if the requestor could trust that no other requestor has changed the value of  
1990 "owner", the requestor could simply ask the provider to replace the owner value 2244 with 2245.  
1991 However, since our requestor is both cautious and general, the requestor instead nests two  
1992 `<modification>` elements within a single `<modifyRequest>`:  
1993 - one `<modification>` to *delete any current values* of "owner" and  
1994 - one `<modification>` to *add the desired value* of "owner".

1995 The `<modification>` that specifies "modificationMode='delete'" contains a  
1996 `<capabilityData>` that specifies "mustUnderstand='true'". This means that the provider  
1997 must process the content of that `<capabilityData>` as the Reference Capability specifies. (If  
1998 the provider cannot do that, the provider must fail the request.)

1999 The `<capabilityData>` contains a `<reference>` that specifies only  
2000 "typeOfReference='owner'". The `<reference>` contains no `<toPsoID>` and (the  
2001 `<reference>` contains) no `<referenceData>` element. The Reference Capability specifies that  
2002 this *incomplete reference acts as a wildcard*. In this context, this `<reference>` that specifies only  
2003 "typeOfReference" matches every `<reference>` that is associated with the object and that  
2004 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>
```

2005 The provider returns a `<modifyResponse>` element. The "status" attribute of the  
2006 `<modifyResponse>` element indicates that the modify request was successfully processed. The  
2007 `<pso>` element of the `<modifyResponse>` shows that the `<reference>` that specifies  
2008 "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>

```

### 2009 **3.6.1.5 delete**

2010 The delete operation enables a requestor to *remove an object* from a target. The delete operation  
2011 automatically removes any *capability-specific data* that is associated with the object.

2012 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"/>

```

#### 2013 **3.6.1.5.1 deleteRequest (normative)**

2014 A requestor **MUST** send a `<deleteRequest>` to a provider in order to (ask the provider to)  
2015 remove an existing object.

2016 **Execution.** A `<deleteRequest>` **MAY** specify "executionMode".  
2017 See the section titled "[Determining execution mode](#)".

2018 **PsoID.** A `<deleteRequest>` **MUST** contain a `<psoID>` element that identifies the object to  
2019 delete.

2020 **Recursive.** A `<deleteRequest>` **MAY** have a "recursive" attribute that specifies whether the  
2021 provider should delete (along with the specified object) any object that the specified object (either  
2022 directly or indirectly) contains.

- 2023 • A requestor that wants the provider to *delete any object that the specified object contains*  
 2024 (along with the specified object) MUST specify "recursive='true'".
- 2025 • A requestor that wants the provider to delete the specified object *only if the specified object*  
 2026 *contains no other object* MUST NOT specify "recursive='true' ". Such a requestor MAY  
 2027 specify "recursive='false' " or (such a requestor MAY) omit the "recursive" attribute  
 2028 (since "recursive='false' " is the default).

### 2029 3.6.1.5.2 deleteResponse (normative)

2030 A provider that receives a <deleteRequest> from a requestor that the provider trusts MUST  
 2031 examine the content of the request. If the request is valid, the provider MUST delete the object  
 2032 (that is specified by the <psoID> sub-element of the <deleteRequest>) if it is possible to do so.

2033 **Execution.** If an <deleteRequest> does not specify "executionMode", the provider MUST  
 2034 choose a type of execution for the requested operation.  
 2035 See the section titled "Determining execution mode".

2036 **Recursive.** A provider MUST NOT delete an object that contains another object unless the  
 2037 <deleteRequest> specifies "recursive='true' ". If the <deleteRequest> specifies  
 2038 "recursive='true' " then the provider MUST delete the specified object along with any object  
 2039 that the specified object (directly or indirectly) contains.

2040 **Response.** The provider must return to the requestor a <deleteResponse>.

2041 **Status.** A <deleteResponse> must contain a "status" attribute that indicates whether the  
 2042 provider successfully deleted the specified object. See the section titled "Status (normative)".

2043 **Error.** If the provider cannot delete the specified object, the <deleteResponse> must contain an  
 2044 "error" attribute that characterizes the failure. See the general section titled "Error (normative)".

2045 In addition, the <deleteResponse> MUST specify an appropriate value of "error" if any of the  
 2046 following is true:

- 2047 • The <psoID> sub-element of the <deleteRequest> is empty (that is, the identifier  
 2048 element has no content). In this case, the <deleteResponse> SHOULD specify  
 2049 "error='noSuchIdentifier'".
- 2050 • The <psoID> sub-element of the <deleteRequest> contains invalid data. In this case the  
 2051 provider SHOULD return "error='unsupportedIdentifierType'".
- 2052 • The <psoID> sub-element of the <deleteRequest> does not specify an object that exists.  
 2053 In this case the <deleteResponse> MUST specify "error='noSuchIdentifier'".
- 2054 • The <psoID> sub-element of the <deleteRequest> specifies an object that contains another  
 2055 object and the <deleteRequest> does not specify "recursive='true' ". In such a case  
 2056 the provider should return "error='containerNotEmpty'".

### 2057 3.6.1.5.3 delete Examples (non-normative)

2058 In the following example, a requestor asks a provider to delete an existing Person object.

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



2059 The provider returns a <deleteResponse> element. The "status" attribute of the  
2060 <deleteResponse> element indicates that the delete request was successfully processed. The  
2061 <deleteResponse> contains no other data.

---

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

2062

2063

2064

## 2065 **3.6.2 Async Capability**

2066 The Async Capability is defined in a schema associated with the following XML namespace:  
2067 `urn:oasis:names:tc:SPML:2:0:async`. The Async Capability XSD is included as Appendix B  
2068 to this document.

2069 A provider that supports asynchronous execution of requested operations for a target SHOULD  
2070 declare that the target supports the Async Capability. A provider that does not support  
2071 asynchronous execution of requested operations for a target MUST NOT declare that the target  
2072 supports the Async Capability.

2073 **IMPORTANT:** The Async Capability does NOT define an operation specific to requesting  
2074 asynchronous execution. A provider that supports the Async Capability (for a schema entity of  
2075 which each object that the requestor desires to manipulate is an instance):

- 2076 1) MUST allow a requestor to specify `"executionMode='asynchronous'"`.  
2077 The provider MUST NOT fail such a request with  
2078 `"error='unsupportedExecutionMode'"`.  
2079 The provider MUST execute the requested operation asynchronously  
2080 (if the provider executes the requested operation at all).  
2081 See the section titled "[Requestor specifies asynchronous execution \(normative\)](#)".
- 2082 2) MAY choose to execute a requested operation asynchronously  
2083 when the request does not specify the `"executionMode"` attribute.  
2084 See the section titled "[Provider chooses asynchronous execution \(normative\)](#)".

2085 The Async Capability also defines two operations that a requestor may use to manage another  
2086 operation that a provider is executing asynchronously:  
2087 • A status operation allows a requestor to check the status (and possibly results) of an operation.  
2088 • A cancel operation asks the provider to stop executing an operation.

2089 **Status.** When a provider is executing SPML operations asynchronously, the requestor needs a way  
2090 to check the status of requests. The [status](#) operation allows a requestor to determine whether an  
2091 asynchronous operation has succeeded or has failed or is still pending. The [status](#) operation also  
2092 allows a requestor to obtain the output of an asynchronous operation.

2093 **Cancel.** A requestor may also need to cancel an asynchronous operation. The cancel operation  
2094 allows a requestor to ask a provider to [stop executing](#) an asynchronous operation.

2095 **Synchronous.** Both the status and cancel operations must be executed synchronously. Because  
2096 both cancel and status operate on other operations that a provider is executing asynchronously, it  
2097 would be confusing to execute cancel or status asynchronously. For example, what would it mean  
2098 to get the status of a status operation? Describing the expected behavior (or interpreting the result)  
2099 of canceling a cancel operation would be difficult, and the chain (e.g., canceling a request to cancel  
2100 a cancelRequest) could become even longer if status or cancel were supported asynchronously.

2101 **Resource considerations.** A provider must limit the size and duration of its asynchronous  
2102 operation results (or that provider will exhaust available resources). A provider must decide:

- 2103 • *How many resources* the provider will devote to storing the results of operations  
2104 that are executed asynchronously (so that the requestor may obtain the results).
- 2105 • *For how long a time* the provider will store the results of each operation  
2106 that is executed asynchronously.

- 2107 These decisions may be governed by the provider's implementation, by its configuration, or by  
2108 runtime computation.
- 2109 A provider that wishes to *never to store the results of operations* SHOULD NOT declare that it  
2110 supports the Async Capability. (Such a provider may *internally* execute requested operations  
2111 asynchronously, but must respond to each request exactly as if the request had been processed  
2112 synchronously.)
- 2113 A provider that wishes to support the asynchronous execution of requested operations MUST store  
2114 the results of an asynchronous operation *for a reasonable period of time* in order to allow the  
2115 requestor to obtain those results. SPMLv2 does not specify a minimum length of time.
- 2116 As a practical matter, a provider cannot queue the results of asynchronous operations forever. The  
2117 provider must eventually release the resources associated with asynchronous operation results.  
2118 (Put differently, a provider must eventually discard the results of an operation that the provider  
2119 executes asynchronously.) Otherwise, the provider may run out of resources.
- 2120 Providers should carefully manage the resources associated with operation results. For example:
- 2121 • A provider may define a *timeout interval* that specifies the maximum time between status  
2122 requests. If a requestor does not request the status of asynchronous operation within this  
2123 interval, the provider will release the results of the asynchronous operation.  
2124 (Any subsequent request for status on this asynchronous operation will receive a response  
2125 that specifies "error='noSuchRequest'".)
  - 2126 • A provider may also define an overall *result lifetime* that specifies the maximum length of time  
2127 to retain the results of an asynchronous operation. After this amount of time has passed, the  
2128 provider will release the results of the operation.
  - 2129 • A provider may also wish to enforce an *overall limit* on the resources available to store the  
2130 results of asynchronous operations, and may wish to adjust its behavior (or even to refuse  
2131 requests for asynchronous execution) accordingly.
  - 2132 • To prevent denial of service attacks, the provider should not allocate any resource on behalf of  
2133 a requestor until that requestor is properly authenticated.

### 2134 3.6.2.1 cancel

2135 The cancel operation enables a requestor to stop the execution of an asynchronous operation. (The  
2136 cancel operation itself must be synchronous.)

2137 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"/>

```

2138 **Cancel must be synchronous.** Because cancel operates on another operation that a provider is  
 2139 executing asynchronously, the cancel operation itself must be synchronous.

2140 **Cancel is not batchable.** Because the cancel operation must be synchronous, a requestor must  
 2141 not nest a cancel request in a [batch](#) request.

### 2142 [3.6.2.1.1](#) *cancelRequest (normative)*

2143 A requestor **MUST** send a `<cancelRequest>` to a provider in order to (ask the provider to) cancel  
 2144 a requested operation that the provider is executing asynchronously.

2145 **Execution.** A `<cancelRequest>` **MUST NOT** specify "executionMode='asynchronous'".  
 2146 A `<cancelRequest>` **MUST** specify "executionMode='synchronous' "  
 2147 or (a `<cancelRequest>` **MUST**) omit the "executionMode" attribute.  
 2148 See the section titled "[Determining execution mode](#)".

2149 **AsyncRequestID.** A `<cancelRequest>` **MUST** have an "asyncRequestID" attribute that  
 2150 specifies the operation to cancel.

### 2151 [3.6.2.1.2](#) *cancelResponse (normative)*

2152 A provider that receives a `<cancelRequest>` from a requestor that the provider trusts **MUST**  
 2153 examine the content of the request. If the request is valid, the provider **MUST** stop the execution of  
 2154 the operation (that the "asyncRequestID" attribute of the `<cancelRequest>` specifies) if it is  
 2155 possible for the provider to do so.

- 2156 • If the provider is already executing the specified operation asynchronously,  
 2157 then the provider **MUST** *terminate execution* of the specified operation.
- 2158 • If the provider plans to execute the specified operation asynchronously  
 2159 but has not yet begun to execute the specified operation,  
 2160 then the provider **MUST** *prevent execution* of the specified operation.

2161 **Execution.** The provider **MUST** execute the cancel operation synchronously (if the provider  
 2162 executes the cancel operation at all). See the section titled "[Determining execution mode](#)".

2163 **Response.** The provider must return to the requestor a `<cancelResponse>`.

2164 **Status.** A `<cancelResponse>` must have a "status" attribute that indicates whether the  
 2165 provider successfully processed the request to cancel the specified operation.  
 2166 See the section titled "[Status \(normative\)](#)".

2167 Since the provider must execute a cancel operation synchronously, the `<cancelResponse>`  
 2168 **MUST NOT** specify "status='pending'". The `<cancelResponse>` **MUST** specify  
 2169 "status='success'" or (the `<cancelResponse>` **MUST** specify) "status='failure'".

2170 If the provider successfully canceled the specified operation, the `<cancelResponse>` MUST  
2171 specify "status=' success' ". If the provider failed to cancel the specified operation, the  
2172 `<cancelResponse>` MUST specify "status=' failure' ".

2173 **Error.** If the provider cannot cancel the specified operation, the `<cancelResponse>` MUST  
2174 contain an "error" attribute that characterizes the failure.  
2175 See the general section titled "[Error \(normative\)](#)".

2176 In addition, the `<cancelResponse>` MUST specify an appropriate value of "error" if any of the  
2177 following is true:

- 2178 • The "asyncRequestID" attribute of the `<cancelRequest>` has no value. In this case, the  
2179 `<cancelResponse>` SHOULD specify "error=' invalidIdentifier' ".
- 2180 • The "asyncRequestID" attribute of the `<cancelRequest>` does not specify an operation  
2181 that exists. In this case the provider SHOULD return "error=' noSuchRequest' ".

### 2182 [3.6.2.1.3 cancel Examples \(non-normative\)](#)

2183 In order to illustrate the cancel operation, we must first execute an operation asynchronously. In the  
2184 following example, a requestor first asks a provider to delete a `Person` asynchronously.

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

2185 The provider returns a `<deleteResponse>` element. The "status" attribute of the  
2186 `<deleteResponse>` element indicates that the provider has chosen to execute the delete  
2187 operation asynchronously. The `<deleteResponse>` also returns a "requestID".

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

2188 Next, the same requestor asks the provider to cancel the delete operation. The requestor specifies  
2189 the value of "requestID" from the `<deleteResponse>` as the value of "asyncRequestID" in  
2190 the `<cancelRequest>`.

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

2191 The provider returns a `<cancelResponse>`. The "status" attribute of the `<cancelResponse>`  
2192 indicates that the provider successfully canceled the delete operation.

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

### 2193 [3.6.2.2 status](#)

2194 The status operation enables a requestor to determine whether an asynchronous operation has  
2195 completed successfully or has failed or is still executing. The status operation also (optionally)  
2196 enables a requestor to obtain results of an asynchronous operation. (The status operation itself  
2197 must be synchronous.)

2198 The subset of the Async Capability XSD that is most relevant to the status operation is shown  
2199 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"/>

```

2200 **Status must be synchronous.** The status operation acts on other operations that a provider is  
2201 executing asynchronously. The status operation itself therefore must be synchronous.

2202 **Status is not batchable.** Because the status operation must be synchronous, a requestor must not  
2203 nest a status request in a [batch](#) request.

#### 2204 **3.6.2.2.1 *statusRequest (normative)***

2205 A requestor **MUST** send a `<statusRequest>` to a provider in order to obtain the status or results  
2206 of a requested operation that the provider is executing asynchronously.

2207 **Execution.** A `<statusRequest>` **MUST NOT** specify "executionMode='asynchronous' ". A  
2208 `<statusRequest>` **MUST** specify "executionMode='synchronous' " or (a  
2209 `<statusRequest>` **MUST**) omit "executionMode".  
2210 See the section titled "[Determining execution mode](#)".

2211 **AsyncRequestID.** A `<statusRequest>` **MAY** have an "asyncRequestID" attribute that  
2212 specifies one operation for which to return status or results. A `<statusRequest>` that omits  
2213 "asyncRequestID" implicitly requests the status of *all* operations that the provider has executed  
2214 asynchronously on behalf of the requestor (and for which operations the provider still retains status  
2215 and results).

2216 **returnResults.** A `<statusRequest>` **MAY** have a "returnResults" attribute that specifies  
2217 whether the requestor wants the provider to return any results (or output) of the operation that is  
2218 executing asynchronously. If a `<statusRequest>` does not specify "returnResults", the  
2219 requestor has implicitly asked that the provider return only the "status" of the operation that is  
2220 executing asynchronously.

### 2221 3.6.2.2.2 *statusResponse (normative)*

2222 A provider that receives a <statusRequest> from a requestor that the provider trusts MUST  
2223 examine the content of the request. If the request is valid, the provider MUST return the status  
2224 (and, if requested, any result) of the operation (that the "asyncRequestID" attribute of the  
2225 <statusRequest> specifies) if it is possible for the provider to do so.

2226 **Execution.** The provider MUST execute the status operation synchronously (if the provider  
2227 executes the status operation at all). See the section titled "[Determining execution mode](#)".

2228 **ReturnResults.** A <statusRequest> MAY have a "returnResults" attribute that indicates  
2229 whether the requestor wants the provider to return in each nested response (in addition to status,  
2230 which is always returned) *any results* of (i.e., output or XML content of the response element for)  
2231 the operation that is executing asynchronously.

2232 • If a <statusRequest> specifies "returnResults='true'", then the provider MUST also  
2233 return in the <statusResponse> any results (or output) of each operation.

2234 • If a <statusRequest> specifies "returnResults='false'", then the provider MUST  
2235 return in the <statusResponse> only the "status" of the each operation.

2236 • If the <statusRequest> does not specify a value for "returnResults", the provider MUST  
2237 assume that the requestor wants only the "status" (and the provider MUST NOT return in  
2238 the <statusResponse> any result) of the operation that is executing asynchronously.

2239 **Response.** The provider must return to the requestor a <statusResponse>.

2240 **Status.** A <statusResponse> must have a "status" attribute that indicates whether the  
2241 provider successfully obtained the status of the specified operation (and obtained any results of the  
2242 specified operation if the <statusRequest> specifies "returnResults='true'").  
2243 See the section titled "[Status \(normative\)](#)".

2244 Since the provider must execute a status operation synchronously, the <statusResponse>  
2245 MUST NOT specify "status='pending'". The <statusResponse> MUST specify  
2246 "status='success'" or (the <statusResponse> MUST specify) "status='failure'".

2247 • If the provider successfully obtained the status of the specified operation (and successfully  
2248 obtained any output of the specified operation if the <statusRequest> specifies  
2249 "returnOutput='true'"), the <statusResponse> MUST specify "status='success'".

2250 • If the provider failed to obtain the status of the specified operation (or failed to obtain any output  
2251 of the specified operation if the <statusRequest> specifies "returnOutput='true'"), the  
2252 <statusResponse> MUST specify "status='failure'".

2253 **Nested Responses.** A <statusResponse> MAY contain any number of responses. Each  
2254 response is an instance of a type that extends {ResponseType}. Each response represents an  
2255 operation that the provider is executing asynchronously.

2256 • A <statusResponse> that specifies "status='failure'" MUST NOT contain an  
2257 embedded response. Since the status operation failed, the response should not contain data.

2258 • A <statusResponse> that specifies "status='success'" MAY contain any number of  
2259 responses.

2260 - If the <statusRequest> specifies "asyncRequestID",  
2261 then a successful <statusResponse> MUST contain *exactly one nested response*  
2262 that represents the operation that "asyncRequestID" specifies.



2263 - If the `<statusRequest>` omits "asyncRequestID",  
2264 then a successful `<statusResponse>` MUST contain a *nested response for each*  
2265 *operation* that the provider has executed asynchronously as the result of a request from  
2266 that requestor (and for which operation the provider still retains status and results).

2267 **Nested Response RequestID.** Each nested response MUST have a "requestID" attribute that  
2268 identifies the corresponding operation (within the namespace of the provider).

2269 **Nested Response Status.** Each nested response MUST have a "status" attribute that  
2270 specifies the current state of the corresponding operation.

2271 • A nested response that represents an operation that failed  
2272 MUST specify "status='failure'".

2273 • A nested response that represents an operation that succeeded  
2274 MUST specify "status='success'".

2275 • A nested response that represents an operation that the provider is still executing  
2276 MUST specify "status='pending'".

2277 **Nested Response and ReturnResults.** If a `<statusRequest>` specifies  
2278 "returnResults='true'", then each response that is nested in the `<statusResponse>`  
2279 MUST contain any output *thus far produced* by the corresponding operation.

2280 • A nested response that specifies "status='success'" MUST contain *all* of the output that  
2281 would have been contained in a synchronous response for the operation if the provider had  
2282 executed the specified operation synchronously.

2283 • A nested response that specifies "status='pending'" MUST contain *an initial subset* of the  
2284 output that would have been contained in a synchronous response for the operation if the  
2285 provider had executed the specified operation synchronously.

2286 **Error.** If the provider cannot obtain the status of the specified operation, the `<statusResponse>`  
2287 MUST contain an "error" attribute that characterizes the failure.  
2288 See the general section titled "[Error \(normative\)](#)".

2289 In addition, a `<statusResponse>` MUST specify an appropriate value of "error" if any of the  
2290 following is true:

2291 • The "asyncRequestID" attribute of the `<statusRequest>` has no value. In this case, the  
2292 `<statusResponse>` SHOULD specify "error='invalidIdentifier'".

2293 • The "asyncRequestID" attribute of the `<statusRequest>` has a value, but does not  
2294 identify an operation for which the provider retains status and results.  
2295 In this case the provider SHOULD return "error='noSuchRequest'".

### 2296 **3.6.2.2.3 status Examples (non-normative)**

2297 In order to illustrate the status operation, we must first execute an operation asynchronously. In this  
2298 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>
```

2299 The provider returns an `<addResponse>`. The "status" attribute of the `<addResponse>`  
2300 indicates that provider will execute the delete operation asynchronously. The `<addResponse>` also  
2301 has a "requestID" attribute (even though the original `<addRequest>` did not specify  
2302 "requestID").

2303 If the original `<addRequest>` had specified a "requestID", then the `<addResponse>` would  
2304 specify the same "requestID" value.

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

2305 The same requestor then asks the provider to obtain the status of the add operation. The requestor  
2306 does not ask the provider to include any output of the add operation.

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

2307 The provider returns a `<statusResponse>`. The "status" attribute of the `<statusResponse>`  
2308 indicates that the provider successfully obtained the status of the add operation.

2309 The `<statusResponse>` also contains a nested `<addResponse>` that represents the add  
2310 operation. The `<addResponse>` specifies "status='pending'", which indicates that the add  
2311 operation has not completed executing.

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

2312 Next, the same requestor asks the provider to obtain the status of the add operation. This time the  
2313 requestor asks the provider to include any results of the add operation.

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

2314 The provider again returns a `<statusResponse>`. The "status" attribute of the  
2315 `<statusResponse>` again indicates that the provider successfully obtained the status of the add  
2316 operation.

2317 The `<statusResponse>` again contains a nested `<addResponse>` that represents the add  
2318 operation. The `<addResponse>` specifies "status='pending'", which indicates that the add  
2319 operation still has not completed executing.

2320 Because the statusRequest specified "returnOutput='true'", the `<addResponse>` contains  
2321 an initial subset of the output that the add operation will eventually produce if the add operation  
2322 successfully completes. The `<ps>` element already contains the `Person` data that was supplied in  
2323 the `<addRequest>` but the `<ps>` element does not yet contain the `<psID>` element that will be  
2324 generated when the add operation is complete.

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

2325 Finally, the same requestor asks the provider to obtain the status of the add operation. The  
2326 requestor again asks the provider to include any output of the add operation.

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

2327 The provider again returns a <statusResponse>. The "status" attribute of the  
2328 <statusResponse> again indicates that the provider successfully obtained the status of the add  
2329 operation.

2330 The <statusResponse> again contains a nested <addResponse> that represents the add  
2331 operation. The <addResponse> specifies "status=' success'", which indicates that the add  
2332 operation completed successfully.

2333 Because the <statusRequest> specified "returnResults=' true'" and because the  
2334 <addResponse> specifies "status=' success'", the <addResponse> now contains all of the  
2335 output of the add operation. The <pso> element contains the <Person> data that was supplied in  
2336 the <addRequest> and the <pso> element also contains the <psoID> element that was missing  
2337 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>
```

2338

2339

### 2340 **3.6.3 Batch Capability**

2341 The Batch Capability is defined in a schema associated with the following XML namespace:  
2342 urn:oasis:names:tc:SPML:2:0:batch. The Batch Capability XSD is included as Appendix C  
2343 to this document.

2344 A provider that supports batch execution of requested operations for a target SHOULD declare that  
2345 the target supports the Batch Capability. A provider that does not support batch execution of  
2346 requested operations MUST NOT declare that the target supports the Batch Capability.

2347 The Batch Capability defines one operation: batch.

#### 2348 **3.6.3.1 batch**

2349 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"/>
```

2350 The batch operation combines any number of individual requests into a single request.

2351 **No transactional semantics.** Using a batch operation to combine individual requests does not  
2352 imply atomicity (i.e., “all-or-nothing” semantics) for the group of batched requests. A requestor must  
2353 not assume that the failure of a nested request will undo a nested request that has already  
2354 completed. (See the section titled “[Transactional Semantics](#)”.)

2355 Note that this does not *preclude* a batch operation having transactional semantics—this is merely  
2356 unspecified. A provider (or some higher-level service) with the ability to undo specific operations  
2357 could support rolling back an entire batch if an operation nested within the batch fails.

2358 **Nested Requests.** The Core XSD defines `{RequestType}` as the base type for any SPML  
2359 request. A requestor may group into a `<batchRequest>` any number of requests that derive from  
2360 `{RequestType}`. However, there are some exceptions. See the topics named “Batch is not  
2361 batchable” and “Some operations are not batchable” immediately below.

2362 **Batch is not batchable.** A requestor must not nest a batch request within another batch request.  
2363 (To support nested batches would impose on each provider a burden of complexity that the benefits  
2364 of nested batches do not justify.)

2365 **Some operations are not batchable.** For various reasons, a requestor must not nest certain  
2366 types of requests within a batch request. For example, a request to [listTargets](#) must not be batched  
2367 (because a requestor cannot know until the requestor examines the response from [listTargets](#)  
2368 whether the provider supports the batch capability). Requests to [search](#) for objects (and requests  
2369 to [iterate](#) the results of a search) must not be batched for reasons of scale. Batching requests to  
2370 [cancel](#) and obtain the [status](#) of asynchronous operations would introduce timing problems.

2371 **Positional correspondence.** The provider’s `<batchResponse>` contains an individual response  
2372 for each individual request that the requestor’s `<batchRequest>` contained. Each individual  
2373 response occupies the same position within the `<batchResponse>` that the corresponding  
2374 individual request occupied within the `<batchRequest>`.

2375 **Processing.** A requestor can specify whether the provider executes the individual requests *one-by-*  
2376 *one in the order that they occur* within a `<batchRequest>`. The “processing” attribute of a  
2377 `<batchRequest>` controls this behavior.

2378 • When a `<batchRequest>` specifies “processing=’sequential’”, the provider must  
2379 execute each requested operation *one at a time and in the exact order* that it occurs within the  
2380 `<batchRequest>`.

2381 • When a `<batchRequest>` specifies “processing=’parallel’”, the provider may execute  
2382 the requested operations within the `<batchRequest>` *in any order*.

2383 **Individual errors.** The “onError” attribute of a `<batchRequest>` specifies whether the provider  
2384 quits at the first error it encounters (in processing individual requests within a `<batchRequest>`) or  
2385 continues despite any number of such errors.

2386 • When a `<batchRequest>` specifies “onError=’exit’”, the provider stops executing  
2387 individual operations within the batch as soon as the provider encounters an error.  
2388 Any operation that produces an error is marked as failed.  
2389 Any operation that the provider does not execute is also marked as failed.

- 2390 • When a <batchRequest> specifies “onError=’ resume’”, the provider handles any error
- 2391 that occurs in processing an individual operation within that <batchRequest>.
- 2392 No error that occurs in processing an individual operation prevents execution of any other
- 2393 individual operation in the batch.
- 2394 Any operation that produces an error is marked as failed.

2395 (Note that a requestor can guarantee pre-requisite processing in batch operations by specifying  
2396 both “processing=’sequential’” and “onError=’exit’”.)

2397 **Overall error.** When a requestor issues a <batchRequest> with “onError=’ resume’” and one  
2398 or more of the requests in that batch fails, then the provider will return a <batchResponse> with  
2399 “status=’ failure’” (even if some of the requests in that batch succeed). The requestor must  
2400 examine every individual response within the overall <batchResponse> to determine which  
2401 requests succeeded and which requests failed.

### 2402 3.6.3.1.1 *batchRequest (normative)*

2403 A requestor MUST send a <batchRequest> to a provider in order to (ask the provider to) execute  
2404 multiple requests as a set.

2405 **Nested Requests.** A <batchRequest> MUST contain at least one element that extends  
2406 {RequestType}.

2407 A <batchRequest> MUST NOT contain as a nested request an element that is of any the  
2408 following types:

- 2409 • {spml:ListTargetsRequestType}
- 2410 • {spmlbatch:BatchRequestType}
- 2411 • {spmlsearch:SearchRequestType}
- 2412 • {spmlsearch:IterateRequestType}
- 2413 • {spmlsearch:CloseIteratorRequestType}
- 2414 • {spmlasync:CancelRequestType}
- 2415 • {spmlasync:StatusRequestType}
- 2416 • {spmlupdates:UpdatesRequestType}
- 2417 • {spmlupdates:IterateRequestType}
- 2418 • {spmlupdates:CloseIteratorRequestType}

2419 **Processing.** A <batchRequest> MAY specify “processing”. The value of any “processing”  
2420 attribute MUST be either ‘sequential’ or ‘parallel’.

2421 • A requestor who wants the provider to process the nested requests *concurrently with one*  
2422 *another* MUST specify “processing=’ parallel’”.

2423 • A requestor who wants the provider to process the nested requests one-by-one and in the  
2424 order that they appear MAY specify “processing=’ sequential’”.

2425 • A requestor who does not specify “processing” is *implicitly* asking the provider to process  
2426 the nested requests *sequentially*.

2427 **onError.** A <batchRequest> MAY specify “onError”. The value of any “onError” attribute  
2428 MUST be either ‘exit’ or ‘resume’.

2429 • A requestor who wants the provider to *continue processing* nested requests whenever  
2430 processing one of the nested requests produces in an error MUST specify  
2431 “onError=’ resume’”.

- 2432 • A requestor who wants the provider to *cease processing* nested requests as soon as  
2433 processing any of the nested requests produces an error MAY specify “onError=’ exit’”.
- 2434 • A requestor who does not specify an “onError” attribute *implicitly* asks the provider to cease  
2435 processing nested requests as soon as processing any of the nested requests produces an  
2436 error.

### 2437 3.6.3.1.2 *batchResponse (normative)*

2438 The provider must examine the content of the <batchRequest>. If the request is valid, the  
2439 provider MUST process each nested request (according to the effective “processing” and  
2440 “onError” settings) if the provider possibly can.

2441 **processing.** If a <batchRequest> specifies “processing=’ parallel’”, the provider SHOULD  
2442 begin executing each of the nested requests as soon as possible. (Ideally, the provider would begin  
2443 executing all of the nested requests immediately and concurrently.) If the provider cannot begin  
2444 executing all of the nested requests at the same time, then the provider SHOULD begin executing  
2445 *as many as possible* of the nested requests *as soon as possible*.

2446 If a <batchRequest> specifies (or defaults to) “processing=’ sequential’”, the provider  
2447 MUST execute each of the nested requests one-by-one and in the order that each appears within  
2448 the <batchRequest>. The provider MUST complete execution of each nested request before the  
2449 provider begins to execute the next nested request.

2450 **onError.** The effect (on the provider’s behavior) of the “onError” attribute of a <batchRequest>  
2451 depends on the “processing” attribute of the <batchRequest>.

- 2452 • If a <batchRequest> specifies (or defaults to) “onError=’ exit’” and (the  
2453 <batchRequest> specifies or defaults to) “processing=’ sequential’” then the provider  
2454 MUST NOT execute any (operation that is described by a) nested request that is subsequent to  
2455 the first nested request that produces an error.  
2456

2457 If the provider encounters an error in executing (the operation that is described by) a nested  
2458 request, the provider MUST report the error in the nested response that corresponds to the  
2459 nested request and then (the provider MUST) specify “status=’ failure’” in every nested  
2460 response that corresponds to a subsequent nested request within the same  
2461 <batchRequest>. The provider MUST also specify “status=’ failure’” in the overall  
2462 <batchResponse>.

- 2463 • If a <batchRequest> specifies (or defaults to) “onError=’ exit’” and (the  
2464 <batchRequest> specifies) “processing=’ parallel’” then the provider’s behavior once  
2465 an error occurs (in processing an operation that is described by a nested request) is *not fully*  
2466 *specified*.  
2467

2468 If the provider encounters an error in executing (the operation that is described by) a nested  
2469 request, the provider MUST report the error in the nested response that corresponds to the  
2470 nested request. The provider MUST also specify “status=’ failure’” in the overall  
2471 <batchResponse>. The provider MUST also specify “status=’ failure’” in the nested  
2472 response that corresponds to any operation the provider has not yet begun to execute.  
2473 However, the provider’s behavior with respect to any operation that has already begun to  
2474 execute but that is not yet complete is not fully specified.  
2475

2476 The provider MAY stop executing any (operation that is described by a) nested request that has  
2477 not yet completed or (the provider MAY) choose to complete the execution of any (operation  
2478 that corresponds to a) nested request (within the same <batchRequest> and) for which the

2479 provider has already begun execution. The provider SHOULD NOT begin to execute any  
2480 operation (that corresponds to a nested request within the same <batchRequest> and) for  
2481 which the provider has not yet begun execution.

2482 • If a <batchRequest> specifies “onError=’ resume’ ” and (the <batchRequest> specifies)  
2483 “processing=’ parallel’ ”, then the provider MUST execute every (operation that is  
2484 described by a) nested request within the <batchRequest>. If the provider encounters an  
2485 error in executing any (operation that is described by a) nested request, the provider MUST  
2486 report the error in the nested response that corresponds to the nested request and then (the  
2487 provider MUST) specify “status=’ failure’ ” in the overall <batchResponse>.

2488 • If a <batchRequest> specifies “onError=’ resume’ ” and (the <batchRequest> specifies  
2489 or defaults to) “processing=’ sequential’ ”, then the provider MUST execute every  
2490 (operation that is described by a) nested request within the <batchRequest>. If the provider  
2491 encounters an error in executing any (operation that is described by a) nested request, the  
2492 provider MUST report the error in the nested response that corresponds to the nested request  
2493 and then (the provider MUST) specify “status=’ failure’ ” in the overall  
2494 <batchResponse>.

2495 **Response.** The provider MUST return to the requestor a <batchResponse>.

2496 **Status.** The <batchResponse> must contain a “status” attribute that indicates whether the  
2497 provider successfully processed every nested request.  
2498 See the section titled “[Status \(normative\)](#)”.

2499 • If the provider successfully executed every (operation described by a) nested request,  
2500 then the <batchResponse> MUST specify “status=’ success’ ”.

2501 • If the provider encountered an error in processing (the operation described by) any nested  
2502 request, the <batchResponse> MUST specify “status=’ failure’ ”.

2503 **nested Responses.** The <batchResponse> MUST contain a nested response for each nested  
2504 request that the <batchRequest> contains. Each nested response within the <batchResponse>  
2505 *corresponds positionally* to a nested request within the <batchRequest>. That is, each nested  
2506 response MUST appear in the same position within the <batchResponse> that the nested request  
2507 (to which the nested response corresponds) originally appeared within the corresponding  
2508 <batchRequest>.

2509 The content of each nested response depends on whether the provider actually executed the  
2510 nested operation that corresponds to the nested response.

2511 • Each nested response that corresponds to a nested request *that the provider did not process*  
2512 MUST specify “status=’ failed’ ”. (A provider might not process a nested request, for  
2513 example, if the provider encountered an error processing an earlier nested request and the  
2514 requestor specified both “processing=’ sequential’ ” and “onError=’ exit’ ”.)

2515 • Each nested response that corresponds to a nested request for an operation *that the provider*  
2516 *actually executed* MUST contain the same data that the provider would have returned (in the  
2517 response for the corresponding operation) *if the corresponding operation had been requested*  
2518 *individually* (rather than as part of a batch operation).

2519 **Error.** If something (other than the behavior specified by the “onError” setting with respect to  
2520 errors that occur in processing nested requests) prevents the provider from processing one or more  
2521 of the (operations described by the) nested requests within a <batchRequest>, then the  
2522 <batchResponse> MUST have an “error” attribute that characterizes the failure.  
2523 See the general section titled “[Error \(normative\)](#)”.



2524 **3.6.3.1.3 batch Examples (non-normative)**

2525 In the following example, a requestor asks a provider to perform a series of operations. The  
2526 requestor asks the provider first to add a `Person` object to one target and then to add an `Account`  
2527 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>
```

2528 The provider returns an `<batchResponse>` element. The "status" of the `<batchResponse>`  
2529 indicates that all of the nested requests were processed successfully. The `<batchResponse>`  
2530 contains an `<addResponse>` for each `<addRequest>` that the `<batchRequest>` contained.  
2531 Each `<addResponse>` contains the same data that it would have contained if the corresponding  
2532 `<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>
```

2533

2534

### 2535 **3.6.4 Bulk Capability**

2536 The Bulk Capability is defined in a schema associated with the following XML namespace:  
2537 urn:oasis:names:tc:SPML:2:0:bulk. This document includes the Bulk Capability XSD as  
2538 Appendix D.

2539 The Bulk Capability defines two operations: bulkModify and bulkDelete.

2540 A provider that supports the bulkModify and bulkDelete operations for a target SHOULD declare  
2541 that the target supports the Bulk Capability. A provider that does not support both bulkModify and  
2542 bulkDelete MUST NOT declare that the target supports the Bulk Capability.

#### 2543 **3.6.4.1 bulkModify**

2544 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"/>
```

2545 The bulkModify operation applies a specified modification to every object that matches the specified  
2546 query.

2547 • The <modification> is the same type of element that is specified as part of a  
2548 <modifyRequest>.

2549 • The <query> is the same type of element that is specified as part of a <searchRequest>.

2550 **Does not return modified PSO Identifiers.** A bulkModify operation does *not* return a <psoid> for  
2551 each object that it changes, even though a bulkModify operation can change the <psoid> for every  
2552 object that it modifies. By contrast, a modify operation does return the <psoid> of any object that it  
2553 changes.

2554 The difference is that the requestor of a bulkModify operation specifies a *query* that selects objects  
2555 to be modified. The requestor of a modify operation specifies the <psoid> of the object to be  
2556 modified. The modify operation therefore must return the <psoid> to make sure that the requestor  
2557 still has the correct <psoid>.

2558 A bulkModify operation does not return a <psoid> for each object that it changes because:

- 2559 • The requestor does not specify a <psoid> as input. (Therefore, a changed <psoid> does not  
2560 necessarily interest the requestor).
- 2561 • Returning PSO Identifiers for modified objects would cause the bulkModify operation to scale  
2562 poorly (which would defeat the purpose of the bulkModify operation).

### 2563 **3.6.4.1.1** *bulkModifyRequest (normative)*

2564 A requestor MUST send a <bulkModifyRequest> to a provider in order to (ask the provider to)  
2565 make the same set of modifications to every object that matches specified selection criteria.

2566 **Execution.** A <bulkModifyRequest> MAY specify "executionMode".  
2567 See the section titled "[Determining execution mode](#)".

2568 **query.** A <bulkModifyRequest> MUST contain exactly one <query> element.  
2569 A <query> describes criteria that (the provider must use to) select objects on a target.  
2570 See the section titled "[SearchQueryType in a Request \(normative\)](#)".

2571 **Modification.** A <bulkModifyRequest> MUST contain at least one <modification>. Each  
2572 <modification> describes a set of changes to be applied (to every object that matches the  
2573 <query>). A requestor MUST specify each <modification> for a <bulkModifyRequest> in  
2574 the same way as for a <modifyRequest>.  
2575 See the topic named "Modification" within the section titled "[modifyRequest \(normative\)](#)".

### 2576 **3.6.4.1.2** *bulkModifyResponse (normative)*

2577 A provider that receives a <bulkModifyRequest> from a requestor that the provider trusts MUST  
2578 examine the content of the <bulkModifyRequest>. If the request is valid, the provider MUST  
2579 apply the (set of changes described by each of the) specified <modification> elements to every  
2580 object that matches the specified <query> (if the provider can possibly do so).  
2581 The section titled "[modifyResponse \(normative\)](#)" describes how the provider should apply each  
2582 <modification> to an object.

2583 **Response.** The provider MUST return to the requestor a <bulkModifyResponse>.

2584 **Status.** The <bulkModifyResponse> must contain a "status" attribute that indicates whether  
2585 the provider successfully applied every specified modification to every object that matched the  
2586 specified query. See the section titled "[Status \(normative\)](#)".

- 2587 • If the provider successfully applied every specified modification to every object that matched  
2588 the specified query, then the <bulkModifyResponse> MUST specify "status='success'".
- 2589 • If the provider encountered an error in selecting any object that matched the specified query or  
2590 (if the provider encountered an error) in applying any specified modification to any of the  
2591 selected objects, then the <bulkModifyResponse> MUST specify "status='failure'".

2592 **Error.** If the provider was unable to apply the specified modification to every object that matched  
2593 the specified query, then the <bulkModifyResponse> MUST have an "error" attribute that  
2594 characterizes the failure. See the general section titled "[Error \(normative\)](#)".

2595 In addition, the section titled "[SearchQueryType Errors \(normative\)](#)" describes errors specific to a  
2596 request that contains a <query>.

2597 **3.6.4.1.3** *bulkModify Examples (non-normative)*

2598 In the following example, a requestor asks a provider to change every `Person` with an email  
2599 address matching `'jbbbriggs@example.com'` to have instead an email address of  
2600 `'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>
```

2601 The provider returns a `<bulkModifyResponse>`. The `"status"` attribute of the  
2602 `<bulkModifyResponse>` indicates that the provider successfully executed the `bulkModify`  
2603 operation.

```
<bulkModifyResponse status="success"/>
```

2604 In the following example, a requestor asks a provider to remove the "owner" of any account that is  
2605 currently owned by "joebob". The requestor uses as a selection criterion the `<hasReference>`  
2606 query clause that the `Reference Capability` defines.

2607 NOTE: The logic required to modify a reference may depend on the cardinality that is defined for  
2608 that type of reference. See the section titled "[Reference Capability](#)". Also see the topic named  
2609 "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>
```

2610 The provider returns a `<bulkModifyResponse>`. The `"status"` attribute of the  
2611 `<bulkModifyResponse>` indicates that the provider successfully executed the `bulkModify`  
2612 operation.

```
<bulkModifyResponse status="success"/>
```

2613 **3.6.4.2** `bulkDelete`

2614 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"/>

```

2615 The bulkDelete operation deletes every object that matches the specified query.

2616 • The <query> is the same element that is specified as part of a <searchRequest>.

### 2617 **3.6.4.2.1** *bulkDeleteRequest (normative)*

2618 A requestor **MUST** send a <bulkDeleteRequest> to a provider in order to (ask the provider to)  
 2619 delete every object that matches specified selection criteria.

2620 **Execution.** A <bulkDeleteRequest> **MAY** specify "executionMode".  
 2621 See the section titled "[Determining execution mode](#)".

2622 **query.** A <bulkDeleteRequest> **MUST** contain exactly one <query> element.  
 2623 A <query> describes criteria that (the provider must use to) select objects on a target.  
 2624 See the section titled "[SearchQueryType in a Request \(normative\)](#)".

2625 **recursive.** A <bulkDeleteRequest> **MAY** have a "recursive" attribute that indicates  
 2626 whether the provider should delete the specified object *along with any other object it contains*.  
 2627 (Unless the <bulkDeleteRequest> specifies "recursive='true'", a provider will not delete  
 2628 an object that contains other objects.)

### 2629 **3.6.4.2.2** *bulkDeleteResponse (normative)*

2630 A provider that receives a <bulkDeleteRequest> from a requestor that the provider trusts must  
 2631 examine the content of the <bulkDeleteRequest>. If the request is valid, the provider **MUST**  
 2632 delete every object that matches the specified <query> (if the provider can possibly do so).

2633 **recursive.** A provider **MUST NOT** delete any object that contains other objects unless the  
 2634 <bulkDeleteRequest> specifies "recursive='true'".

2635 • If the <bulkDeleteRequest> specifies "recursive='true'",  
 2636 then the provider **MUST** delete every object that matches the specified query  
 2637 *along with any object that a matching object (directly or indirectly) contains*.

2638 • If the <bulkDeleteRequest> specifies "recursive='false'"  
 2639 (or if the <bulkDeleteRequest> omits the "recursive" attribute")  
 2640 and at least one object that matches the specified query contains another object,  
 2641 then the provider **MUST NOT** delete any of the objects that match the specified query.  
 2642 In this case, the provider's response must return an error (see below).

2643 **Response.** The provider MUST return to the requestor a `<bulkDeleteResponse>`.

2644 **Status.** The `<bulkDeleteResponse>` must contain a "status" attribute that indicates whether  
 2645 the provider successfully deleted every object that matched the specified query.  
 2646 See the section titled "[Status \(normative\)](#)".

- 2647 • If the provider successfully deleted every object that matched the specified query, the  
 2648 `<bulkDeleteResponse>` MUST specify "status='success'".
- 2649 • If the provider encountered an error in selecting any object that matched the specified query or  
 2650 (if the provider encountered an error) in deleting any of the selected objects, the  
 2651 `<bulkDeleteResponse>` MUST specify "status='failure'".

2652 **Error.** If the provider was unable to delete every object that matched the specified query, then the  
 2653 `<bulkDeleteResponse>` MUST have an "error" attribute that characterizes the failure.  
 2654 See the general section titled "[Error \(normative\)](#)".

2655 In addition, the section titled "[SearchQueryType Errors \(normative\)](#)" describes errors specific to a  
 2656 request that contains a `<query>`. Also see the section titled "[SelectionType Errors \(normative\)](#)".

2657 If at least one object that matches the specified query contains another object  
 2658 and the `<bulkDeleteRequest>` does NOT specify "recursive='true'",  
 2659 then the provider's response should specify "error='invalidContainment'".

### 2660 [3.6.4.2.3 bulkDelete Examples \(non-normative\)](#)

2661 In the following example, a requestor asks a provider to delete every `Person` with an email address  
 2662 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>
```

2663 The provider returns a `<bulkDeleteResponse>`. The "status" attribute of the  
 2664 `<bulkDeleteResponse>` indicates that the provider successfully executed the `bulkDelete`  
 2665 operation.

```
<bulkDeleteResponse status="success"/>
```

2666 In the following example, a requestor asks a provider to delete any `Account` that is currently  
 2667 owned by "joebob". The requestor uses as a selection criterion the `<hasReference>` query clause  
 2668 that the Reference Capability defines.

```
<bulkDeleteRequest>
  <query scope="subtree" targetID="target2" >
    <hasReference typeOfReference="owner">
      <toPsoID ID="2244" targetID="target2"/>
    </hasReference>
  </query>
</bulkDeleteRequest>
```

2669 The provider returns a `<bulkDeleteResponse>`. The "status" attribute of the  
 2670 `<bulkDeleteResponse>` indicates that the provider successfully executed the `bulkDelete`  
 2671 operation.

```
<bulkDeleteResponse status="success"/>
```

## 2672 3.6.5 Password Capability

2673 The Password Capability is defined in a schema that is associated with the following XML  
2674 namespace: `urn:oasis:names:tc:SPML:2:0:password`. This document includes the  
2675 Password Capability XSD as Appendix E.

2676 The Password Capability defines four operations: `setPassword`, `expirePassword`, `resetPassword`  
2677 and `validatePassword`.

- 2678 • The `setPassword` operation *changes to a specified value* the password that is associated with a  
2679 specified object. The `setPassword` operation also allows a requestor to supply the current  
2680 password (in case the target system or application requires it).
- 2681 • The `expirePassword` operation *marks as no longer valid* the password that is associated with a  
2682 specified object. (Most systems or applications will require a user to change an expired  
2683 password on the next login.)
- 2684 • The `resetPassword` operation *changes to an unspecified value* the password that is associated  
2685 with a specified object. The `resetPassword` operation returns the new password.
- 2686 • The `validatePassword` operation *tests whether a specified value would be valid* as the  
2687 password for a specified object. (The `validatePassword` operation allows a requestor to test a  
2688 password value against the password policy for a system or application.)

2689 A provider that supports the `setPassword`, `expirePassword`, `resetPassword` and `validatePassword`  
2690 operations for a target SHOULD declare that the target supports the Password Capability. A  
2691 provider that does not support all of the `setPassword`, `expirePassword`, `resetPassword` and  
2692 `validatePassword` operations MUST NOT declare that the target supports the Password Capability.

### 2693 3.6.5.1 setPassword

2694 The `setPassword` operation enables a requestor to *specify a new password* for an object.

2695 The subset of the Password Capability XSD that is most relevant to the `setPassword` operation  
2696 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"/>
```



2697 **3.6.5.1.1 setPasswordRequest (normative)**

2698 A requestor MUST send a <setPasswordRequest> to a provider in order to (ask the provider to)  
2699 change to a specified value the password that is associated an existing object.

2700 **Execution.** A <setPasswordRequest> MAY specify "executionMode".  
2701 See the section titled "Determining execution mode".

2702 **psoid.** A <setPasswordRequest> MUST contain exactly one <psoid> element. The <psoid>  
2703 MUST identify an object that exists on a target (that is supported by the provider).  
2704 See the section titled "PSO Identifier (normative)".

2705 **password.** A <setPasswordRequest> MUST contain exactly one <password> element. A  
2706 <password> element MUST contain a string value.

2707 **currentPassword.** A <setPasswordRequest> MAY contain at most one <currentPassword>  
2708 element. A <currentPassword> element MUST contain a string value.

2709 **3.6.5.1.2 setPasswordResponse (normative)**

2710 A provider that receives a <setPasswordRequest> from a requestor that the provider trusts  
2711 MUST examine the content of the <setPasswordRequest>. If the request is valid and if the  
2712 specified object exists, then the provider MUST change (to the value that the <password> element  
2713 contains) the password that is associated with the object that is specified by the <psoid>.

2714 **Execution.** If a <setPasswordRequest> does not specify "executionMode", the provider  
2715 MUST choose a type of execution for the requested operation.  
2716 See the section titled "Determining execution mode".

2717 **Response.** The provider must return to the requestor a <setPasswordResponse>. The  
2718 <setPasswordResponse> must have a "status" attribute that indicates whether the provider  
2719 successfully changed (to the value that the <password> element contains) the password that is  
2720 associated with the specified object. See the section titled "Status (normative)".

2721 **Error.** If the provider cannot change (to the value that the <password> element contains) the  
2722 password that is associated with the requested object, the <setPasswordResponse> must  
2723 contain an "error" attribute that characterizes the failure.  
2724 See the general section titled "Error (normative)".

2725 In addition, a <setPasswordResponse> MUST specify an error if any of the following is true:

- 2726 • The <setPasswordRequest> contains a <psoid> for an object that does not exist.
- 2727 • The target system or application will not accept (as the new password) the value that a  
2728 <setPasswordRequest> supplies as the content of the <password> element.
- 2729 • The target system or application *requires the current password* in order to change the password  
2730 and a <setPasswordRequest> supplies no content for <currentPassword>.
- 2731 • The target system or application *requires the current password* in order to change the password  
2732 and the target system or application will not accept (as the current password) the value that a  
2733 <setPasswordRequest> supplies as the content of <currentPassword>.
- 2734 • The target system or application *returns an error (or throws an exception)* when the provider  
2735 tries to set the password.



2736 **3.6.5.1.3 setPassword Examples (non-normative)**

2737 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>
```

2738 The provider returns a `<setPasswordResponse>` element. The "status" of the  
2739 `<setPasswordResponse>` indicates that the provider successfully changed the password.

```
<setPasswordResponse requestID="133" status="success"/>
```

2740 **3.6.5.2 expirePassword**

2741 The `expirePassword` operation *marks as invalid the current password* for an object.

2742 The subset of the Password Capability XSD that is most relevant to the `expirePassword` operation  
2743 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"/>
```

2744 **3.6.5.2.1 expirePasswordRequest (normative)**

2745 A requestor **MUST** send a `<expirePasswordRequest>` to a provider in order to (ask the provider  
2746 to) mark as no longer valid the password that is associated with an existing object.

2747 **Execution.** A `<expirePasswordRequest>` **MAY** specify "executionMode".

2748 See the section titled "[Determining execution mode](#)".

2749 **psoID.** A `<expirePasswordRequest>` **MUST** contain exactly one `<psoID>` element. The

2750 `<psoID>` **MUST** identify an object that exists on a target (that is supported by the provider).

2751 See the section titled "[PSO Identifier \(normative\)](#)".

2752 **remainingLogins.** A `<expirePasswordRequest>` **MAY** have a "remainingLogins" attribute  
2753 that specifies a number of grace logins that the target system or application should permit.

2754 **3.6.5.2.2** *expirePasswordResponse (normative)*

2755 A provider that receives a <expirePasswordRequest> from a requestor that the provider trusts  
2756 MUST examine the content of the <expirePasswordRequest>. If the request is valid and if the  
2757 specified object exists, then the provider MUST mark as no longer valid the password that is  
2758 associated with the object that the <psoID> specifies.

2759 **Execution.** If an <expirePasswordRequest> does not specify "executionMode", the provider  
2760 MUST choose a type of execution for the requested operation.  
2761 See the section titled "[Determining execution mode](#)".

2762 **Response.** The provider must return to the requestor an <expirePasswordResponse>. The  
2763 <expirePasswordResponse> must have a "status" attribute that indicates whether the  
2764 provider successfully marked as no longer valid the password that is associated with the specified  
2765 object. See the section titled "[Status \(normative\)](#)" for values of this attribute.

2766 **Error.** If the provider cannot mark as invalid the password that is associated with the requested  
2767 object, the <expirePasswordResponse> must contain an "error" attribute that characterizes  
2768 the failure. See the general section titled "[Error \(normative\)](#)".

2769 In addition, an <expirePasswordResponse> MUST specify an error if any of the following is  
2770 true:

- 2771 • The <expirePasswordRequest> contains a <psoID> for an object that does not exist.
- 2772 • The target system or application will not accept (as the number of grace logins to permit) the  
2773 value that a <expirePasswordRequest> specifies for the "remainingLogins" attribute.
- 2774 • The target system or application *returns an error (or throws an exception)* when the provider  
2775 tries to mark as no longer valid the password that is associated with the specified object.

2776 **3.6.5.2.3** *expirePassword Examples (non-normative)*

2777 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>
```

2778 The provider returns an <expirePasswordResponse> element. The "status" attribute of the  
2779 <expirePasswordResponse> element indicates that the provider successfully expired the  
2780 password.

```
<expirePasswordResponse requestID="134" status="success"/>
```

2781 **3.6.5.3** *resetPassword*

2782 The resetPassword operation enables a requestor to *change (to an unspecified value)* the  
2783 password for an object and to obtain that newly generated password value.

2784 The subset of the Password Capability XSD that is most relevant to the resetPassword operation  
2785 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"/>

```

#### 2786 **3.6.5.3.1** *resetPasswordRequest (normative)*

2787 A requestor **MUST** send a `<resetPasswordRequest>` to a provider in order to (ask the provider  
2788 to) change the password that is associated an existing object and to (ask the provider to) return to  
2789 the requestor the new password value.

2790 **Execution.** A `<resetPasswordRequest>` **MAY** specify "executionMode".  
2791 See the section titled "[Determining execution mode](#)".

2792 **psoID.** A `<resetPasswordRequest>` **MUST** contain exactly one `<psoID>` element. The  
2793 `<psoID>` **MUST** identify an object that exists on a target (that is supported by the provider).  
2794 See the section titled "[PSO Identifier \(normative\)](#)".

#### 2795 **3.6.5.3.2** *resetPasswordResponse (normative)*

2796 A provider that receives a `<resetPasswordRequest>` from a requestor that the provider trusts  
2797 **MUST** examine the content of the `<resetPasswordRequest>`. If the request is valid and if the  
2798 specified object exists, then the provider **MUST** change the password that is associated with the  
2799 object that is specified by the `<psoID>` and must return to the requestor the new password value.

2800 **Execution.** If an `<resetPasswordRequest>` does not specify "executionMode", the provider  
2801 **MUST** choose a type of execution for the requested operation.  
2802 See the section titled "[Determining execution mode](#)".

2803 **Response.** The provider must return to the requestor a `<resetPasswordResponse>`. The  
2804 `<resetPasswordResponse>` must have a "status" attribute that indicates whether the provider  
2805 successfully changed the password that is associated with the specified object and successfully  
2806 returned to the requestor the new password value. See the section titled "[Status \(normative\)](#)".

2807 If the provider knows that the provider will not be able to return to the requestor the new password  
2808 value, then the provider **MUST NOT** change the password that is associated with the specified  
2809 object. (To do so would create a state that requires manual administrator intervention, and this  
2810 defeats the purpose of the resetPassword operation.)

2811 **password.** The <resetPasswordResponse> MAY contain a <password> element. If the  
2812 <resetPasswordResponse> contains a <password> element, the <password> element MUST  
2813 contain the newly changed password value that is associated with the specified object.

2814 **Error.** If the provider cannot change the password that is associated with the specified object, or if  
2815 the provider cannot return the new password attribute value to the requestor, then the  
2816 <resetPasswordResponse> MUST specify an "error" that characterizes the failure.  
2817 See the general section titled "[Error \(normative\)](#)".

2818 In addition, a <resetPasswordResponse> MUST specify an error if any of the following is true:

- 2819 • The <resetPasswordRequest> contains a <psoID> for an object that does not exist.
- 2820 • The target system or application will not allow the provider to return to the requestor the new  
2821 password value. (If the provider knows this to be the case, then the provider MUST NOT  
2822 change the password that is associated with the specified object. See above.)
- 2823 • The target system or application *returns an error (or throws an exception)* when the provider  
2824 tries to change the password that is associated with the specified object or (when the provider)  
2825 tries to obtain the new password value.

### 2826 [3.6.5.3 resetPassword Examples \(non-normative\)](#)

2827 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>
```

2828 The provider returns an <resetPasswordResponse> element. The "status" attribute of the  
2829 <resetPasswordResponse> indicates that the provider successfully reset the password.

```
<resetPasswordResponse requestID="135" status="success">  
  <password>gener8ed</password>  
</resetPasswordResponse>
```

### 2830 [3.6.5.4 validatePassword](#)

2831 The validatePassword operation enables a requestor to *determine whether a specified value would*  
2832 *be valid* as the password for a specified object.

2833 The subset of the Password Capability XSD that is most relevant to the validatePassword operation  
2834 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"/>

```

#### 2835 **3.6.5.4.1** *validatePasswordRequest (normative)*

2836 A requestor **MUST** send a `<validatePasswordRequest>` to a provider in order to (ask the  
2837 provider to) test whether a specified value would be valid as the password that is associated with  
2838 an existing object.

2839 **Execution.** A `<validatePasswordRequest>` **MAY** specify "executionMode".  
2840 See the section titled "[Determining execution mode](#)".

2841 **psoid.** A `<validatePasswordRequest>` **MUST** contain exactly one `<psoid>` element. The  
2842 `<psoid>` **MUST** identify an object that exists on a target (that is supported by the provider).  
2843 See the section titled "[PSO Identifier \(normative\)](#)".

2844 **password.** A `<validatePasswordRequest>` **MUST** contain exactly one `<password>` element.  
2845 The `<password>` element **MUST** contain a string value.

#### 2846 **3.6.5.4.2** *validatePasswordResponse (normative)*

2847 A provider that receives a `<validatePasswordRequest>` from a requestor that the provider  
2848 trusts **MUST** examine the content of the `<validatePasswordRequest>`. If the request is valid  
2849 and if the specified object exists, then the provider **MUST** test whether the specified value would be  
2850 valid as the password that is associated with the object that the `<psoid>` identifies.

2851 **Execution.** If an `<validatePasswordRequest>` does not specify "executionMode", the  
2852 provider **MUST** choose a type of execution for the requested operation.  
2853 See the section titled "[Determining execution mode](#)".

2854 **Response.** The provider must return to the requestor a `<validatePasswordResponse>`. The  
2855 `<validatePasswordResponse>` **MUST** have a "status" attribute that indicates whether the  
2856 provider successfully tested whether the supplied value would be valid as the password that is  
2857 associated with the specified object. See the section titled "[Status \(normative\)](#)".

2858 **valid.** The `<validatePasswordResponse>` **MUST** have a "valid" attribute that indicates  
2859 whether the `<password>` (content that was specified in the `<validatePasswordRequest>`)  
2860 would be valid as the password that is associated with the specified object.

2861 **Error.** If the provider cannot determine whether the specified value would be valid as the password  
2862 that is associated with the specified object, then the `<validatePasswordResponse>` **MUST**  
2863 specify an "error" value that characterizes the failure.  
2864 See the general section titled "[Error \(normative\)](#)".

2865 In addition, a `<validatePasswordResponse>` **MUST** specify an appropriate value of "error" if  
2866 any of the following is true:

- 2867
- The `<validatePasswordRequest>` contains a `<psoid>` for an object that does not exist.
- 2868
- The target system or application *returns an error (or throws an exception)* when the provider
- 2869
- tries to determine whether the specified value would be valid as the password that is
- 2870
- associated with the specified object.

2871 **3.6.5.4.3** *validatePassword Examples (non-normative)*

2872 In the following example, a requestor asks a provider to validate a value as a password for a  
2873 `Person` object.

```
<validatePasswordRequest requestID="136">  
  <psoid ID="2244" targetID="target2"/>  
  <password>y0baby</password>  
</validatePasswordRequest>
```

2874 The provider returns an `<validatePasswordResponse>` element. The "status" attribute of  
2875 the `<validatePasswordResponse>` indicates that the provider successfully tested whether the  
2876 `<password>` value specified in the request would be valid as the password that is associated with  
2877 the specified object. The `<validatePasswordResponse>` specifies "valid='true'", which  
2878 indicates that the specified value *would be valid* as the password that is associated with the  
2879 specified object.

```
<validatePasswordResponse requestID="136" status="success" valid="true"/>
```

2880

### 2881 3.6.6 Reference Capability

2882 The Reference Capability is defined in a schema that is associated with the following XML  
2883 namespace: urn:oasis:names:tc:SPML:2:0:reference. This document includes the  
2884 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"/>
```

2885 The Reference Capability defines no operation. Instead, the Reference Capability allows a provider  
2886 to declare, as part of each target, which types of objects support *references* to which other types of  
2887 objects. The XML representations of *references flow through the core operations as capability-*  
2888 *specific data*.

2889 • In order to *create an object with references*, a requestor specifies capability-specific data to the  
2890 add operation.

2891 • In order to *add, remove or replace references* to an object, a requestor specifies capability-  
2892 specific data to the modify operation.

2893 • In order to *obtain references* for an object, a requestor examines capability-specific data  
2894 returned as output by the add, lookup and search operations.

2895 **Motivation.** Defining a standard capability for references is important for several reasons.

2896 • Managing references to other objects can be an important part of managing objects.

2897 • Object references to other objects present a *scalability* problem.

2898 • Object references to other objects present an *integrity* problem.

2899 Provisioning systems must often list, create, and delete connections between objects  
2900 in order to manage the objects themselves. In some cases, a provisioning system  
2901 must manage data that is part a specific connection (e.g., in order to specify  
2902 the expiration of a user's membership in a group) – see the topic named “Reference Data” below.  
2903 Because connections to other objects can be very important, it is important to be able to represent  
2904 such connections *generically* (rather than as something specific to each target schema).

2905 The reference capability enables a requestor to manage an object's references independent of the  
2906 object's schema. This is particularly important in the cases where a provider allows references to  
2907 span targets. For example, a provisioning system must often maintain knowledge about which  
2908 people own which accounts. In such cases, an `Account` object (that is contained by one target)  
2909 may refer to a `Person` object (that is contained by another target) as its owner.

2910 Scale is another significant aspect of references. The *number of connections* between objects may  
2911 be an order of magnitude greater than the number of objects themselves. Unconditionally including  
2912 reference information in the XML representation of each object could greatly increase the size of  
2913 each object's XML representation. Imagine, for example, that each `Account` may refer to multiple  
2914 `Groups` (or that a `Group` might refer to each of its members).

2915 Defining reference as an optional capability (and allowing references to be omitted from each  
2916 object's schema) does two things. First, this allows a requestor to exclude an object's references  
2917 from the XML representation of each object (since a requestor can control which capability-specific  
2918 data are included). Second, this allows providers to manage references separately from schema-  
2919 defined attributes (which may help a provider cope with the scale of connections).

2920 The ability to manage references separately from schema-defined data may also help providers to  
2921 maintain the integrity of references. In the systems and applications that underlie many  
2922 provisioning target, deleting an object A may not delete another object B's reference to object A.  
2923 Allowing a provider to manage references separately allows the provider to control such behavior  
2924 (and perhaps even to prevent the deletion of object A when another object B still refers to object A).



### 2925 3.6.6.1 Reference Definitions

2926 **Reference Definitions.** A provider declares each type of reference that a particular target supports  
2927 (or declares each type of reference *that a particular supported schema entity* on a target supports)  
2928 as an instance of {ReferenceDefinitionType}.

2929 A provider's <listTargetsResponse> contains a list of targets that the provider exposes for  
2930 provisioning operations. Part of each target declaration is the set of capabilities that the target  
2931 supports. Each capability refers (by means of its "namespaceURI" attribute) to a specific  
2932 capability. Any <capability> element that refers to the Reference Capability may contain (as  
2933 open content) any number of <referenceDefinition> elements.

2934 Each reference definition names a specific type of reference and also specifies the following:  
2935 • *which schema entity* (on the <target> that contains the <capability> that contains the  
2936 <referenceDefinition>) *can refer...*  
2937 • *...to which schema entity* or schema entities (on which targets).

2938 For normative specifics, see the topic named "Reference Capability content" within the section titled  
2939 "[listTargetsResponse \(normative\)](#)".

2940 **Overlap.** Any number of reference definitions may declare different "from- and to-" entity pairs for  
2941 the same type of reference. For example, a reference definition may declare that an Account may  
2942 refer to a Person as its "owner". Another reference definition may declare that an  
2943 OrganizationalUnit may refer to a Person as its "owner". SPMLv2 specifies the mechanism-  
2944 *-but does not define the semantics--*of reference.

2945 **Direction.** Each reference definition specifies the *direction* of reference. A reference is always  
2946 from an object (that is an instance of the schema entity that <schemaEntity> specifies) to  
2947 another object (that is an instance of a schema entity that <canReferTo> specifies).

2948 **No Inverse.** A standard SPMLv2 reference definition specifies nothing about an inverse  
2949 relationship. For example, a reference definition that says an Account may refer to a Person as  
2950 its "owner" does NOT imply that a Person may refer to Account.

2951 Nothing prevents a provider from declaring (by means of a reference definition) that Person may  
2952 refer to Account in a type of reference called "owns", but nothing (at the level of this specification)  
2953 associates these two types of references to say that "owns" is the inverse of "owner".

2954 **No Cardinality.** A reference definition specifies no restrictions on the number of objects to which an  
2955 object may refer (by means of that defined type of reference). Thus, for example, an Account may  
2956 refer to multiple instances of Person as its "owner". This may be logically incorrect, or this may  
2957 not be the desired behavior, but SPMLv2 does not require a provider to support restrictions on the  
2958 cardinality of a particular type of reference.

2959 In general, a requestor must assume that each defined type of reference is optional and many-to-  
2960 many. This is particularly relevant when a requestor wishes to modify references. A requestor  
2961 SHOULD NOT assume that a reference that the requestor wishes to modify is the object's only  
2962 reference of that type. A requestor also SHOULD NOT assume that a reference from one object to  
2963 another object that the requestor wishes to modify is the *only* reference between the two objects.  
2964 The only restriction that SPMLv2 imposes is that an object A may have no more than one reference  
2965 of the same type to another object B. See the topic named "No duplicates" in the section titled  
2966 "[References](#)".

2967 **ReferenceDataType.** A reference definition may be *complex*, which means that an instance of that  
2968 type of reference may have reference data associated with it.  
2969 See the section titled "[Complex References](#)" below.

2970 The definition of a type of reference that is complex must contain a `<referenceDataType>` for  
2971 each possible structure of reference data. Each `<referenceDataType>` element refers to a  
2972 specific entity in a target schema. A `<referenceData>` element (within any instance of that type  
2973 of reference) may contain one element of any of these types (to which a `<referenceDataType>`  
2974 refers).

2975 A reference definition that contains no `<referenceDataType>` sub-element indicates that the  
2976 type of reference it defines *does not support reference data*.

2977 For a normative description, see the topic named "ReferenceDefinition referenceDataType" within  
2978 the section titled "[listTargetsResponse \(normative\)](#)".

### 2979 **3.6.6.2 References**

2980 **Must contain toPsoID.** Any `<reference>` MUST specify its "toObject". That is, any instance of  
2981 `{ReferenceType}` MUST contain a valid `<toPsoID>`. The only exception is a `<reference>`  
2982 that is used as a wildcard within a `<modification>` that specifies  
2983 "modificationMode='delete' ". In this case (and only in this case), the `<reference>` MUST  
2984 specify a valid "typeOfReference" but (the `<reference>`) MAY omit `<toPsoID>`.  
2985 See the section titled "[Reference CapabilityData Processing \(normative\)](#)".

2986 **No duplicates.** Within the set of references that is associated with an object, at most one  
2987 `<reference>` of a specific "typeOfReference" may refer to a particular object. That is, an  
2988 instance of `{CapabilityDataType}` MUST NOT contain two (and MUST NOT contain more than  
2989 two) instances of `<reference>` that specify the same value of "typeOfReference" and that  
2990 contain `<toPsoID>` elements that identify the same object. See the section titled "[Reference  
2991 CapabilityData in a Request \(normative\)](#)".

2992 **Reference Data.** SPMLv2 allows each reference (i.e., each instance of `{ReferenceType}`) to  
2993 contain additional reference data. Most references between objects require no additional data, but  
2994 allowing references to contain additional data supports cases in which a reference from one object  
2995 to another may carry additional information "on the arrow" of the relationship. For example, a  
2996 RACF user's membership in a particular RACF group carries with it the additional information of  
2997 whether that user has the ADMINISTRATOR or SPECIAL privilege within that group. Several other  
2998 forms of group membership carry with them additional information about the member's expiration.  
2999 See the section titled "[Complex References](#)" below.

3000 **Search.** A requestor can *search for objects based on reference values* using the  
3001 `<hasReference>` query clause. The `{HasReferenceType}` extends `{QueryClauseType}`,  
3002 which indicates that an instance of `{HasReferenceType}` can be used to select objects. A  
3003 `<hasReference>` clause matches an object if and only if the object has a reference that *matches*  
3004 *every specified component* (i.e., element or attribute) of the `<hasReference>` element.  
3005 See the section titled "[search Examples](#)".

### 3006 **3.6.6.3 Complex References**

3007 The vast majority of reference types are simple: that is, one object's reference to another object  
3008 carries no additional information. However certain types of references may support additional  
3009 information that is specific to a particular reference. For example, when a user is assigned to one  
3010 or more Entrust GetAccess Roles, each role assignment has a start date and an end date. We  
3011 describe a *reference that contains additional data* (where that data is specific to the reference) as a  
3012 "complex" reference.

3013 **Example: RACF Group Membership** is another example of a complex type of reference. Each  
3014 RACF group membership carries with it additional data about whether the user has the SPECIAL,  
3015 AUDITOR, or OPERATIONS privileges in that group.

- 3016 • Group-SPECIAL gives a group administrator control over all profiles within the group
- 3017 • Group-AUDITOR allows a user to monitor the use of the group's resources
- 3018 • Group-OPERATIONS allows a user to perform maintenance operations  
3019 on the group's resources

3020 For purposes of this example, let us represent these three group-specific privileges as attributes of  
3021 an XML type called "RacfGroupMembershipType". Suppose that the XML Schema for such a type  
3022 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"/>
```

3023

3024 The following subsections describe several different ways to model RACF Group Membership. The  
3025 fictional `<xsd:schema>` is the same in all of the examples. In each subsection, however, the  
3026 provider's `<target>` definition varies with the approach.

### 3027 3.6.6.3.1 Using Reference Data

3028 The simplest way to model a complex reference such as RACF Group membership is to represent  
3029 the additional information as arbitrary *reference data*. The `<referenceData>` element within a  
3030 `<reference>` may contain any data.

3031 The following example shows how a provider's `listTargetsResponse` might reflect this approach.  
3032 The sample schema for the "RACF" target is very simple (for the sake of brevity). The provider  
3033 defines a type of reference called "memberOfGroup". Within a `<reference>` of this type, the  
3034 `<referenceData>` element must contain exactly one `<racfGroupMembership>` element (and  
3035 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>

```

3036 **Manipulating Reference Data.** The only way to manipulate the reference data associated with a  
 3037 complex reference is by using the [modify](#) operation that is part of the Core XSD. A requestor may  
 3038 add, replace or delete any capability-specific data that is associated with an object.

3039 **Capabilities Do Not Apply.** SPML specifies no way to apply a capability-specific operation to a  
 3040 reference. Thus, for example, one can neither suspend nor resume a reference. This is because a  
 3041 *reference is not a provisioning service object*. A reference is instead *capability-specific data that is*  
 3042 *associated with an object*.

3043 You can think of an object's references (or any set of capability-specific data that is associated with  
 3044 an object) as an "extra" attribute (or as an "extra" sub-element) of the object. The provider supports  
 3045 each "extra" (attribute or sub-element) data *independent of the schema* of the target that contains  
 3046 the object. The provider keeps all <capabilityData> separate from the regular schema-defined  
 3047 <data> within each <ps>.

### 3048 **3.6.6.3.2 Relationship Objects**

3049 The fact that capabilities cannot apply to references does not prevent a provider from offering this  
 3050 kind of rich function. There is an elegant way to represent a complex relationship that allows a

3051 requestor to operate directly on the relationship itself. A provider may model a complex relationship  
3052 between two objects as a third object that refers to each of the first two objects.

3053 This approach is analogous to a “linking record” in relational database design. In the “linking  
3054 record” approach, the designer “normalizes” reference relationships into a separate table. Each  
3055 row in a third table connects a row from one table to a row in another table. This approach allows  
3056 each relationship to carry additional information that is specific to that relationship. Data specific to  
3057 each reference are stored in the columns of the third table. Even when relationships do not need to  
3058 carry additional information, database designers often use this approach when two objects may be  
3059 connected by more than one instance of the same type of relationship, or when relationships are  
3060 frequently added or deleted and referential integrity must be maintained.

3061 Rather than have an object A refer to an object B directly, a third object C refers to both object A  
3062 and object B. Since object C represents the relationship itself, object C refers to object A as its  
3063 “fromObject” and object C refers to object B as its “toObject”.

3064 A provider that wants to treat each instance of a (specific type of) relationship as an object does so  
3065 by defining in the schema for a target a schema entity to contain the additional information (that is  
3066 specific to that type of relationship). The provider then declares two types of references that apply  
3067 to that schema entity: a “fromObject” type of reference and a “toObject” type of reference. The  
3068 provider may also declare that certain capabilities apply to that schema entity. This model allows a  
3069 requestor to operate conveniently on each instance of a complex relationship.

3070 For example, suppose that a provider models as a schema entity a type of relationship that has an  
3071 effective date and has an expiration date. As a convenience to requestors, the provider might  
3072 declare that this schema entity (that is, the “linking” entity) supports the Suspend Capability. The  
3073 ‘suspend’ and ‘resume’ operations could manipulate the expiration date and the effective date  
3074 *without the requestor having to understand the structure of that schema entity*. This convenience  
3075 could be very valuable where the attribute values or element content that are manipulated have  
3076 complex syntax, special semantics or implicit relationships with other elements or attributes.

3077 The following example shows how a provider’s listTargetsResponse might reflect this approach.  
3078 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>

```

3079 **Variations.** Naturally, many variations of this approach are possible. For example, an instance of  
 3080 RacfUserProfile could refer to an instance of RacfGroupMembership (rather than having an  
 3081 instance of RacfGroupMembership refer to both RacfUserProfile and an instance of  
 3082 RacfGroupProfile). However, such a variation would not permit an instance of RacfUserProfile to  
 3083 refer to more than one group (and could result in an orphaned relationship objects unless the  
 3084 provider carefully guards against this).

### 3085 3.6.6.3.3 *Bound Relationship Objects*

3086 One particularly robust variation of independent relationship objects is to *bind each relationship*  
 3087 *object beneath one of the objects it connects*. For example, one could bind each instance of  
 3088 RacfGroupMembership beneath the instance of RacfUserProfile that would otherwise be the  
 3089 "fromUser". That way, deleting an instance of RacfUserProfile also deletes all of its  
 3090 RacfGroupMemberships. This modeling approach makes clear that the relationship belongs with  
 3091 the "fromObject" and helps to prevent orphaned relationship objects.

3092 The next example illustrates bound relationship objects.

```

<listTargetsResponse status="success">
  <target targetID="RacfGroupMembership-BoundRelationshipObject">
    <schema>
      <schema targetNamespace="urn:example:schema:RACF"
        xmlns="http://www.w3.org/2001/XMLSchema" 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" />
</schema>
  <supportedSchemaEntity entityName="racfUserProfile" isContainer="true">
    <annotation>
      <documentation> Any number of racfGroupMembership objects may be
bound beneath a racfUserProfile object.</documentation>
    </annotation>
  </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>

```

#### 3093 3.6.6.4 Reference CapabilityData in a Request (normative)

- 3094 The general rules that govern an instance of {CapabilityDataType} in a request also apply to  
3095 an instance of {CapabilityDataType} that refers to the Reference Capability.  
3096 See the section titled "[CapabilityData in a Request \(normative\)](#)".



3097 **capabilityURI.** An instance of {CapabilityDataType}  
3098 that contains data that are specific to the Reference Capability MUST specify  
3099 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'".

3100 **mustUnderstand.** An instance of {CapabilityDataType} that refers to the Reference  
3101 Capability SHOULD specify "mustUnderstand='true'".

3102 **Capability defines structure.** An instance of {CapabilityDataType} that refers to the  
3103 Reference Capability MUST contain at least one <reference> element. An instance of  
3104 {CapabilityDataType} that refers to the Reference Capability SHOULD NOT contain any  
3105 element that is not a <reference> element.

3106 **No duplicates.** Within the set of references that is associated with an object, at most one  
3107 <reference> of a specific "typeOfReference" may refer to a specific object. That is, an  
3108 instance of {CapabilityDataType} MUST NOT contain two (and MUST NOT contain more than  
3109 two) instances of <reference> that specify the same value of "typeOfReference" and that  
3110 contain <toPsoID> elements that identify the same object.

3111 **Validate each reference.** Any <reference> that an instance of {CapabilityDataType}  
3112 contains must be an instance of {spmlref:ReferenceType}. In addition, a provider MUST  
3113 examine the following aspects of each <reference>:

- 3114 - The "from" object. (The object that contains--or that is intended to contain--the reference.)
- 3115 - The "to" object. (The object that the <toPsoID> of the reference identifies.)
- 3116 - The "from" schema entity. (The schema entity of which the "from" object is an instance.)
- 3117 - The "to" schema entity (The schema entity of which the "to" object is an instance.)
- 3118 - The typeOfReference
- 3119 - Any referenceData

3120 The standard aspects of SPML that specify supported schema entities and capabilities imply the  
3121 following:

- 3122 - The "to" object MUST exist (on a target that the provider exposes).
- 3123 - The target that contains the "from" object MUST support the "from" schema entity.
- 3124 - The target that contains the "to" object MUST support the "to" schema entity.
- 3125 - The target that contains the "from" object MUST support the Reference Capability.
- 3126 - The target that contains the "from" object MUST declare that
- 3127 the Reference Capability applies to the "from" schema entity.

3128 See the section titled "[listTargetsResponse \(normative\)](#)".

3129 **Check Reference Definition.** In addition, a provider must validate the "typeOfReference" that  
3130 each <reference> specifies (as well as the "from" schema entity and the "to" schema entity)  
3131 against the set of valid reference definitions..

3132 The <capability> that declares that the target (that contains the "from" object)  
3133 supports the Reference Capability for the "from" schema entity  
3134 MUST contain a <referenceDefinition> for which all of the following are true:

- 3135 - The <referenceDefinition> specifies the same "typeOfReference"  
3136 that the <reference> specifies
- 3137 - The <referenceDefinition> contains a <schemaEntity> element  
3138 that specifies the "from" schema entity
- 3139 - The <referenceDefinition> contains a <canReferTo> element  
3140 that specifies the "to" schema entity.

3141 See the section titled "[Reference Definitions](#)" above.

### 3142 3.6.6.5 Reference CapabilityData Processing (normative)

3143 The general rules that govern processing of an instance of {CapabilityDataType} in a request  
3144 also apply to an instance of {CapabilityDataType} that refers to the Reference Capability. See  
3145 the section titled "[CapabilityData Processing \(normative\)](#)".

3146 **capabilityURI.** An instance of {CapabilityDataType} that refers to the Reference Capability  
3147 MUST specify "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference' ". The  
3148 target (that contains the object to be manipulated) MUST support the Reference Capability for the  
3149 schema entity of which the object to be manipulated is an instance.

3150 **mustUnderstand.** An instance of {CapabilityDataType} that refers to the Reference  
3151 Capability SHOULD specify "mustUnderstand='true' ". A provider that supports the Reference  
3152 Capability MUST handle the content as this capability specifies (regardless of the value of  
3153 "mustUnderstand"). See the topic named "mustUnderstand" within the section titled  
3154 "[CapabilityData Processing \(normative\)](#)".

3155 **Open content.** An instance of {CapabilityDataType} that refers to the Reference Capability  
3156 MUST contain at least one <reference>. An instance of {CapabilityDataType} that refers to  
3157 the Reference Capability SHOULD NOT contain any element that is not a <reference>.

3158 **Validation.** A provider MUST examine the content of any instance of {CapabilityDataType}  
3159 that refers to the Reference Capability (regardless of the type of request that contains the instance  
3160 of {CapabilityDataType}) and ensure that it contains only valid instances of <reference>.  
3161 See the section titled "[Reference CapabilityData in a Request \(normative\)](#)".

3162 If the content (of the instance of {CapabilityDataType} that refers to the Reference Capability)  
3163 is not valid, then the provider's response MUST specify "status='failure' ".  
3164 See the section titled "[Request CapabilityData Errors \(normative\)](#)".

3165 **Process individual references.** In addition to the validation described above, the content of an  
3166 instance of {CapabilityDataType} that refers to the Reference Capability is not treated as  
3167 opaque, but instead as a set of individual references. The handling of each <reference>  
3168 depends on the type of element that contains the instance of {CapabilityDataType}).

- 3169 • If an <addRequest> contains an instance of {CapabilityDataType} that refers to the  
3170 Reference Capability, then the provider MUST associate the instance of  
3171 {CapabilityDataType} (and each <reference> that it contains)  
3172 with the newly created object.
- 3173 • If a <modification> contains an instance of {CapabilityDataType} that refers to the  
3174 Reference Capability, then the handling of each <reference> (that the instance of  
3175 {CapabilityDataType} contains) depends on the "modificationMode" of that  
3176 <modification> and also depends on whether a matching <reference> is already  
3177 associated with the object to be modified.
  - 3178 - If the <modification> specifies "modificationMode='add' ",  
3179 then the provider MUST *add each new reference* for which no matching <reference> is  
3180 already associated with the object.  
3181 That is, the provider MUST associate with the object to be modified each <reference>  
3182 (that the instance of {CapabilityDataType} within the <modification> contains)  
3183 for which no <reference> that is already associated with the object  
3184 specifies the same value for "typeOfReference" (that the <reference> from the  
3185 <modification> specifies) and contains a <toPsoID> that identifies the same object  
3186 (that the <toPsoID> of the <reference> from the <modification> identifies).

3187  
3188 The provider MUST *replace each matching reference* that is already associated with the  
3189 object with the <reference> from the <modification>.  
3190 That is, if a <reference> that is already associated with the object specifies the same  
3191 value for "typeOfReference" (that the <reference> from the <modification>  
3192 specifies) and if the <reference> that is already associated with the object contains a  
3193 <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from  
3194 the <modification> identifies), then the provider MUST *remove* the <reference> that  
3195 is already associated with the object and (the provider MUST) *add* the <reference> from  
3196 the <modification>.  
3197 This has the net effect of replacing any optional <referenceData> (as well as replacing  
3198 any open content) of the matching <reference>.

3199 - If the <modification> specifies "modificationMode='replace'",  
3200 then the provider MUST *add each new reference* for which no matching <reference> is  
3201 already associated with the object.  
3202 That is, the provider MUST associate with the object to be modified each <reference>  
3203 (that the instance of {CapabilityDataType} within the <modification> contains)  
3204 for which no <reference> that is already associated with the object  
3205 specifies the same value for "typeOfReference" (that the <reference> from the  
3206 <modification> specifies) and contains a <toPsoID> that identifies the same object  
3207 (that the <toPsoID> of the <reference> from the <modification> identifies).  
3208  
3209 The provider MUST *replace each matching reference* that is already associated with the  
3210 object with the <reference> from the <modification>.  
3211 That is, if a <reference> that is already associated with the object specifies the same  
3212 value for "typeOfReference" (that the <reference> from the <modification>  
3213 specifies) and if the <reference> that is already associated with the object contains a  
3214 <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from  
3215 the <modification> identifies), then the provider MUST *remove* the <reference> that  
3216 is already associated with the object and (the provider MUST) *add* the <reference> from  
3217 the <modification>.  
3218 This has the net effect of replacing any optional <referenceData> (as well as replacing  
3219 any open content) of the matching <reference>.

3220 - If the <modification> specifies "modificationMode='delete'",  
3221 then the provider MUST *remove each matching reference*.  
3222 A reference that omits <toPsoID> is *treated as a wildcard*.  
3223  
3224 If the <reference> from the <modification> contains a <toPsoID> element,  
3225 then the provider MUST remove (from the set of references that are associated with the  
3226 object) any <reference> that specifies the same value for "typeOfReference" (that  
3227 the <reference> from the <modification> specifies) and that contains a <toPsoID>  
3228 that identifies the same object (that the <toPsoID> of the <reference> from the  
3229 <modification> identifies).  
3230  
3231 If the <reference> from the <modification> contains no <toPsoID> element,  
3232 then the provider MUST remove (from the set of references that are associated with the  
3233 object) any <reference> that specifies the same value for "typeOfReference" (that  
3234 the <reference> from the <modification> specifies).  
3235  
3236 If no instance of <reference> that is associated with the object to be modified matches  
3237 the <reference> from the <modification>, then the provider MUST do nothing for that

3238 <reference>. In this case, the provider's response MUST NOT specify  
3239 "status='failure'" unless there is some other reason to do so.

### 3240 **3.6.6.6 Reference CapabilityData Errors (normative)**

3241 The general rules that govern errors related to an instance of {CapabilityDataType} in a  
3242 request also apply to an instance of {CapabilityDataType} that refers to the Reference  
3243 Capability. See the section titled "[CapabilityData Errors \(normative\)](#)".

3244 A provider's response to a request that contains an instance of {CapabilityDataType} that  
3245 refers to the Reference Capability (e.g., a <capabilityData> element that specifies  
3246 "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'")  
3247 MUST specify an error if any of the following is true:

- 3248 • The instance of {CapabilityDataType} that refers to the Reference Capability  
3249 does not contain at least one <reference> element.
- 3250 • The instance of {CapabilityDataType} that refers to the Reference Capability  
3251 contains a <reference> element that is not a valid instance of {ReferenceType}.
- 3252 • The instance of {CapabilityDataType} that refers to the Reference Capability  
3253 contains a <reference> element for which no instance of Reference Definition declares that  
3254 (an instance of) the "from" schema entity may refer to (an instance of) the "to" schema entity  
3255 with the typeOfReference that the <reference> specifies.  
3256 See the section titled "[Reference Definitions](#)" above.

3257 A provider's response to a request that contains an instance of {CapabilityDataType} that  
3258 refers to the Reference Capability MAY specify an error if any of the following is true:

- 3259 • The instance of {CapabilityDataType} that refers to the Reference Capability  
3260 contains data other than valid <reference> elements.

3261 A provider's response (to a request that contains an instance of {CapabilityDataType} that  
3262 refers to the Reference Capability) SHOULD contain an <errorMessage> for each <reference>  
3263 element that was not valid.

### 3264 **3.6.6.7 Reference CapabilityData in a Response (normative)**

3265 The general rules that govern an instance of {CapabilityDataType} in a response also apply to  
3266 an instance of {CapabilityDataType} that refers to the Reference Capability.  
3267 See the section titled "[CapabilityData in a Response \(normative\)](#)".

3268 The specific rules that apply to an instance of {CapabilityDataType} that refers to the  
3269 Reference Capability *in a response* also apply to an instance of {CapabilityDataType} (that  
3270 refers to the Reference Capability) *in a request*. (However, if the provider has applied the rules in  
3271 processing each request, the provider should not need to apply those rules again in formatting a  
3272 response.) See the section titled "[Reference CapabilityData in a Request \(normative\)](#)".

### 3273 3.6.7 Search Capability

3274 The Search Capability is defined in a schema associated with the following XML namespace:  
3275 `urn:oasis:names:tc:SPML:2:0:search`. This document includes the Search Capability XSD  
3276 as Appendix G.

3277 The Search Capability defines three operations: `search`, `iterate` and `closeiterator`. The search and  
3278 iterate operations together allow a requestor to obtain *in a scalable manner* the XML representation  
3279 of every object that matches specified selection criteria. The search operation returns in its  
3280 response a first set of matching objects. Each subsequent iterate operation returns more matching  
3281 objects. The `closeiterator` operation allows a requestor to tell a provider that it does not intend to  
3282 finish iterating a search result (and that the provider may therefore release the associated  
3283 resources).

3284 A provider that supports the search and iterate operations for a target SHOULD declare that the  
3285 target supports the Search Capability. A provider that does not support both search and iterate  
3286 MUST NOT declare that the target supports the Search Capability.

3287 **Resource considerations.** A provider must limit the size and duration of its search results (or that  
3288 provider will exhaust available resources). A provider must decide:

- 3289 • How large of a search result the provider will *select* on behalf of a requestor.
- 3290 • How large of a search result the provider will *queue* on behalf of a requestor  
3291 (so that the requestor may iterate the search results).
- 3292 • For *how long a time* the provider will queue a search result on behalf of a requestor.

3293 These decisions may be governed by the provider's implementation, by its configuration, or by  
3294 runtime computation.

3295 A provider that wishes to *never to queue search results* may return every matching object (up to the  
3296 provider's limit and up to any limit specified by the requestor) in the search response. Such a  
3297 provider would never return an iterator, and would not need to support the iterate operation. The  
3298 disadvantage is that, without an iterate operation, a provider's search capability either is limited to  
3299 small results or produces large search responses.

3300 A provider that wishes to support the iterate operation must store (or somehow queue) the objects  
3301 selected by a search operation until the requestor has a chance to iterate those results. (That is, a  
3302 provider must somehow queue the objects that matched the criteria of a search operation and that  
3303 were not returned in the search response.)

3304 If all goes well, the requestor will continue to iterate the search result until the provider has sent all  
3305 of the objects to the requestor. The requestor may also use the `closeiterator` operation to tell the  
3306 provider that the requestor is no longer interested in the search result. In either case, the provider  
3307 may free any resource that is still associated with the search result. However, it is possible that the  
3308 requestor may not iterate the search result in a timely manner—or that the requestor may *never*  
3309 iterate the search result completely. Such a requestor may also neglect to close the iterator.

3310 A provider cannot queue search results indefinitely. The provider must eventually release the  
3311 resources that are associated with a search result. (Put differently, any iterator that a provider  
3312 returns to a requestor must eventually expire.) Otherwise, the provider may run out of resources.

3313 Providers should carefully manage the resources associated with search results. For example:

- 3314 • A provider may define a *timeout interval* that specifies the maximum time between iterate  
3315 requests. If a requestor does not request an iterate operation within this interval, the provider

- 3316 will release the resources associated with the search result. This invalidates any iterator that  
3317 represents this search result.
- 3318 • A provider may also define an overall *result lifetime* that specifies the maximum length of time  
3319 to retain a search result. After this amount of time has passed, the provider will release the  
3320 search result.
- 3321 • A provider may also wish to enforce an *overall limit* on the resources available to queue search  
3322 results, and may wish to adjust its behavior (or even to refuse search requests) accordingly.
- 3323 • To prevent denial of service attacks, the provider should not allocate any resource on behalf of  
3324 a requestor until that requestor is properly authenticated.  
3325 See the section titled "[Security and Privacy Considerations](#)".

### 3326 **3.6.7.1 search**

3327 The search operation obtains every object that matches a specified query.

3328 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"/>

```

3329 The <query> is the same type of element that is specified as part of a <bulkModifyRequest> or  
3330 a <bulkDeleteRequest>. See the section titled "[SearchQueryType](#)".

3331 If the search operation is successful *but selects no matching object*, the <searchResponse> will  
3332 not contain a <pso>.

3333 If the search operation is successful *and selects at least one matching object*, the  
3334 <searchResponse> will contain any number of <pso> elements, each of which represents a  
3335 matching object. If the search operation selects more matching objects than the  
3336 <searchResponse> contains, the <searchResponse> will also contain an <iterator> that the  
3337 requestor can use to retrieve more matching objects. (See the iterate operation below.)

3338 If a search operation would select more objects than the provider can queue for subsequent  
3339 iteration by the requestor, the provider's <searchResponse> will specify  
3340 "error='resultSetTooLarge'".

3341 **Search is not batchable.** For reasons of scale, neither a search request nor an iterate request  
3342 should be nested in a [batch](#) request. When a search query matches more objects than the provider  
3343 can place directly in the response, the provider must temporarily store the remaining objects.  
3344 Storing the remaining objects allows the requestor to iterate the remaining objects, but also requires  
3345 the provider to commit resources.  
3346 See the topic named "Resource Considerations" earlier in this section.



3347 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
3348 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
3349 storing search results. Allowing a requestor to nest a search request within a batch request would  
3350 aggravate the resource problem, requiring a provider to store more information in larger chunks for  
3351 a longer amount of time.

### 3352 **3.6.7.1.1** *searchRequest (normative)*

3353 A requestor **MUST** send a `<searchRequest>` to a provider in order to (ask the provider to) obtain  
3354 every object that matches specified selection criteria.

3355 **Execution.** A `<searchRequest>` **MAY** specify "executionMode".  
3356 See the section titled "[Determining execution mode](#)".

3357 **query.** A `<query>` describes criteria that (the provider must use to) select objects on a target.  
3358 A `<searchRequest>` **MAY** contain at most one `<query>` element.

3359 • If the provider's `<listTargetsResponse>` contains only a single `<target>`,  
3360 then a `<searchRequest>` may omit the `<query>` element.

3361 • If the provider's `<listTargetsResponse>` contains more than one `<target>`,  
3362 then a `<searchRequest>` **MUST** contain exactly one `<query>` element  
3363 and that `<query>` must specify "targetID".

3364 See the section titled "[SearchQueryType in a Request \(normative\)](#)".

3365 **ReturnData.** A `<searchRequest>` **MAY** have a "returnData" attribute that tells the provider  
3366 which types of data to include in each selected object.

3367 • A requestor that wants the provider to return *nothing* of the added object  
3368 **MUST** specify "returnData='nothing'".

3369 • A requestor that wants the provider to return *only the identifier* of the added object  
3370 **MUST** specify "returnData='identifier'".

3371 • A requestor that wants the provider to return the identifier of the added object  
3372 *plus the XML representation of the object (as defined in the schema of the target)*  
3373 **MUST** specify "returnData='data'".

3374 • A requestor that wants the provider to return the identifier of the added object  
3375 *plus the XML representation of the object (as defined in the schema of the target)*  
3376 *plus any capability-specific data that is associated with the object*  
3377 **MAY** specify "returnData='everything'" or **MAY** omit the "returnData" attribute  
3378 (since "returnData='everything'" is the default).

3379 **maxSelect.** A `<searchRequest>` **MAY** have a "maxSelect" attribute. The value of the  
3380 "maxSelect" attribute specifies the maximum number of objects the provider should select.

3381 **IncludeDataForCapability.** A `<searchRequest>` **MAY** contain any number of  
3382 `<includeDataForCapability>` elements. Each `<includeDataForCapability>` element  
3383 specifies a capability for which the provider should return capability-specific data (unless the  
3384 "returnData" attribute specifies that the provider should return no capability-specific data at all).

3385 • A requestor that wants the provider to return (as part of each object) capability-specific data *for*  
3386 *only a certain set of capabilities* **MUST** enumerate that set of capabilities (by including an  
3387 `<includeDataForCapability>` element that specifies each such capability) in the  
3388 `<searchRequest>`.

3389 • A requestor that wants the provider to return (as part of each object) capability-specific data *for*  
3390 *all capabilities* MUST NOT include an <includeDataForCapability> element in the  
3391 <searchRequest>.

3392 • A requestor that wants the provider to return *no capability-specific data* MUST specify an  
3393 appropriate value for the "returnData" attribute.  
3394 See the topic named "ReturnData" immediately previous.

### 3395 3.6.7.1.2 *searchResponse (normative)*

3396 A provider that receives a <searchRequest> from a requestor that the provider trusts must  
3397 examine the content of the <searchRequest>. If the request is valid, the provider MUST return  
3398 (the XML that represents) every object that matches the specified <query> (if the provider can  
3399 possibly do so). However, the number of objects selected (for immediate return or for eventual  
3400 iteration) MUST NOT exceed any limit specified as "maxSelect" in the <searchRequest>.

3401 **Execution.** If an <searchRequest> does not specify "executionMode", the provider MUST  
3402 choose a type of execution for the requested operation.  
3403 See the section titled "[Determining execution mode](#)".

3404 A provider SHOULD execute a search operation synchronously if it is possible to do so. (The  
3405 reason for this is that the result of a search should reflect the current state of each matching object.  
3406 Other operations are more likely to intervene if a search operation is executed asynchronously.)

3407 **Response.** The provider MUST return to the requestor a <searchResponse>.

3408 **Status.** The <searchResponse> must contain a "status" attribute that indicates whether the  
3409 provider successfully selected every object that matched the specified query.  
3410 See the section titled "[Status \(normative\)](#)".

3411 • If the provider successfully returned (the XML that represents) every object that matched the  
3412 specified <query> up to any limit specified by the value of the "maxSelect" attribute, then the  
3413 <searchResponse> MUST specify "status='success'".

3414 • If the provider encountered an error in selecting any object that matched the specified <query>  
3415 or (if the provider encountered an error) in returning (the XML that represents) any of the  
3416 selected objects, then the <searchResponse> MUST specify "status='failure'".

3417 **PSO.** The <searchResponse> MAY contain any number of <psO> elements.

3418 • If the <searchResponse> specifies "status='success'" and *at least one object matched*  
3419 the specified <query>, then the <searchResponse> MUST contain at least one <psO>  
3420 element that contains (the XML representation of) a matching object.

3421 • If the <searchResponse> specifies "status='success'" and *no object matched* the  
3422 specified <query>, then the <searchResponse> MUST NOT contain a <psO> element.

3423 • If the <searchResponse> specifies "status='failure'", then the <searchResponse>  
3424 MUST NOT contain a <psO> element.

3425 **PSO and ReturnData.** Each <psO> contains the subset of (the XML representation of) a requested  
3426 object that the "returnData" attribute of the <searchRequest> specified. By default, each  
3427 <psO> contains the entire (XML representation of an) object.

- 3428 • A <ps> element MUST contain a <psID> element.  
3429 The <psID> element MUST contain the identifier of the requested object.  
3430 See the section titled “PSO Identifier (normative)”.
- 3431 • A <ps> element MAY contain a <data> element.
- 3432 - If the <searchRequest> specified “returnData=’ identifier’ ”,  
3433 then the <ps> MUST NOT contain a <data> element.
- 3434 - Otherwise, if the <searchRequest> specified “returnData=’ data’ ”  
3435 or (if the <searchRequest> specified) “returnData=’ everything’ ”  
3436 or (if the <searchRequest>) omitted the “returnData” attribute  
3437 then the <data> element MUST contain the XML representation of the object.  
3438 This XML must be valid according to the schema of the target for the schema entity of  
3439 which the newly created object is an instance.
- 3440 • A <ps> element MAY contain any number of <capabilityData> elements. Each  
3441 <capabilityData> element contains a set of *capability-specific data* that is associated with  
3442 the newly created object (for example, a *reference* to another object).
- 3443 - If the <searchRequest> specified “returnData=’ identifier’ ”  
3444 or (if the <searchRequest> specified) “returnData=’ data’ ”  
3445 then the <ps> MUST NOT contain a <capabilityData> element.
- 3446 - Otherwise, if the <searchRequest> specified “returnData=’ everything’ ”  
3447 or (if the <searchRequest>) omitted the “returnData” attribute,  
3448 then the <ps> MUST contain a <capabilityData> element for each set of capability-  
3449 specific data that is associated with the requested object  
3450 (and that is specific to a capability that the target supports for the schema entity of which  
3451 the requested object is an instance).
- 3452 **PSO capabilityData and IncludeDataForCapability.** A <searchResponse> MUST include (as  
3453 <capabilityData> sub-elements of each <ps>) any set of capability-specific data that is  
3454 associated with a matching object and for which *all* of the following are true:
- 3455 • The <searchRequest> specifies “returnData=’ everything’ ” or (the  
3456 <searchRequest>) omits the “returnData” attribute.
- 3457 • The schema for the target declares that the *target supports the capability* (for the schema entity  
3458 of which each matching object is an instance).
- 3459 • The <searchRequest> contains an <includeDataForCapability> element that contains  
3460 (as its string content) the URI of the capability to which the data are specific or the  
3461 <searchRequest> contains no <includeDataForCapability> element.
- 3462 A <searchResponse> SHOULD NOT include (as a <capabilityData> sub-element of each  
3463 <ps>) any set of capability-specific data for which any of the above is not true.
- 3464 **iterator.** A <searchResponse> MAY contain at most one <iterator> element.
- 3465 • If the <searchResponse> specifies “status=’ success’ ” and the search response *contains*  
3466 *all of the objects* that matched the specified <query>, then the <searchResponse> MUST  
3467 NOT contain an <iterator>.
- 3468 • If the <searchResponse> specifies “status=’ success’ ” and the search response *contains*  
3469 *some but not all of the objects* that matched the specified <query>, then the  
3470 <searchResponse> MUST contain exactly one <iterator>.

- 3471 • If the <searchResponse> specifies “status=' success' ” and *no object matched* the  
 3472 specified <query>, then the <searchResponse> MUST NOT contain an <iterator>.
- 3473 • If the <searchResponse> specifies “status=' failure' ”, then the <searchResponse>  
 3474 MUST NOT contain an <iterator>.

3475 **iterator ID.** An <iterator> MUST have an “ID” attribute.

3476 The value of the “ID” attribute uniquely identifies the <iterator> within the namespace of the  
 3477 provider. The “ID” attribute allows the provider to map each <iterator> token to the result set of  
 3478 the requestor’s <query> and (also allows the provider to map each <iterator> token) to any  
 3479 state that records the requestor’s position within that result set.

3480 The “ID” attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an  
 3481 <iterator>. An <iterator> is not a PSO.

3482 **Error.** If the <searchResponse> specifies “status=' failure' ”, then the <searchResponse>  
 3483 MUST have an “error” attribute that characterizes the failure.

3484 See the general section titled “[Error \(normative\)](#)”.

3485 The section titled “[SearchQueryType Errors \(normative\)](#)” describes errors specific to a request that  
 3486 contains a <query>. Also see the section titled “[SelectionType Errors \(normative\)](#)”.

3487 In addition, a <searchResponse> MUST specify an appropriate value of “error” if any of the  
 3488 following is true:

- 3489 • If the *number of objects that matched* the <query> that was specified in a <searchRequest>  
 3490 *exceeds any limit on the part of the provider* (but does not exceed any value of “maxSelect”  
 3491 that the requestor specified as part of the <query>). In this case, the provider’s  
 3492 <searchResponse> SHOULD specify “error=' resultSetTooLarge' ”.

### 3493 [3.6.7.1.3 search Examples \(non-normative\)](#)

3494 In the following example, a requestor asks a provider to search for every Person with an email  
 3495 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>
```

3496 The provider returns a <searchResponse>. The “status” attribute of the <searchResponse>  
 3497 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>
```

3498 In the following example, a requestor asks a provider to search for every account that is currently  
3499 owned by "joebob". The requestor uses the "returnData" attribute to specify that the provider  
3500 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>
```

3501 The provider returns a <searchResponse>. The "status" attribute of the <searchResponse>  
3502 indicates that the provider successfully executed the search operation.

```
<searchResponse requestID="138" status="success" >  
  <pso>  
    <psoID ID="1431" targetID="target1"/>  
  </pso>  
</searchResponse>
```

### 3503 **3.6.7.2 iterate**

3504 The iterate operation obtains the next set of objects from the result set that the provider selected for  
3505 a search operation. (See the description of the [search operation](#) above.)

3506 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"/>
```

3507 **An iterateRequest receives an iterateResponse.** A requestor supplies as input to an  
3508 <iterateRequest> the <iterator> that was part of the original <searchResponse> or the  
3509 <iterator> that was part of a subsequent <iterateResponse>, whichever is most recent. A  
3510 provider returns an <iterateResponse> in response to each <iterateRequest>. An  
3511 <iterateResponse> has the same structure as a <searchResponse>.

3512 The <iterateResponse> will contain at least one <psO> element that represents a matching  
3513 object. If more matching objects are available to return, then the <iterateResponse> will also  
3514 contain an <iterator>. The requestor can use this <iterator> in another  
3515 <iterateRequest> to retrieve more of the matching objects.

3516 **Iterate is not batchable.** For reasons of scale, neither a search request nor an iterate request  
3517 should be nested in a [batch](#) request. When a search query matches more objects than the provider  
3518 can place directly in the response, the provider must temporarily store the remaining objects.  
3519 Storing the remaining objects allows the requestor to iterate the remaining objects, but also requires  
3520 the provider to commit resources.  
3521 See the topic named "Resource Considerations" earlier in this section.

3522 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
3523 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
3524 search results. Allowing a requestor to nest a search request or an iterate request within a batch  
3525 request would aggravate the resource problem, requiring a provider to store more information in  
3526 larger chunks for a longer amount of time.

3527 **The iterate operation must be executed synchronously.** The provider is already queuing the  
3528 result set (every object beyond those returned in the first search response), so it is unreasonable  
3529 for a requestor to ask the provider to queue the results of a request for the next item in the result  
3530 set.

3531 Furthermore, asynchronous iteration would complicate the provider's maintenance of the result set.  
3532 Since a provider could never know that the requestor had processed the results of an  
3533 asynchronous iteration, the provider would not know when to increment its position in the result set.  
3534 In order to support asynchronous iteration both correctly and generally, a provider would have to  
3535 maintain a version of every result set *for each iteration* of that result set. This would impose an  
3536 unreasonable burden on the provider.

### 3537 [3.6.7.2.1](#) *iterateRequest (normative)*

3538 A requestor MUST send an <iterateRequest> to a provider in order to obtain any *additional*  
3539 objects that matched a previous <searchRequest> but that the provider has not yet returned to  
3540 the requestor. (That is, matching objects that were not contained in the response to that  
3541 <searchRequest> and that have not yet been contained in any response to an  
3542 <iterateRequest> associated with that <searchRequest>.)

3543 **Execution.** An <iterateRequest> MUST NOT specify "executionMode='asynchronous'".  
3544 An <iterateRequest> MUST specify "executionMode='synchronous' "  
3545 or (an <iterateRequest> MUST) omit "executionMode".  
3546 See the section titled "[Determining execution mode](#)".



3547 **iterator.** An <iterateRequest> MUST contain exactly one <iterator> element. A requestor  
3548 MUST supply as input to an <iterateRequest> the <iterator> from the original  
3549 <searchResponse> or (the requestor MUST supply as input to the <iterateRequest>) the  
3550 <iterator> from a subsequent <iterateResponse>. A requestor SHOULD supply as input  
3551 to an <iterateRequest> the most recent <iterator> that represents the search result set.

### 3552 3.6.7.2.2 *iterateResponse (normative)*

3553 A provider that receives a <iterateRequest> from a requestor that the provider trusts must  
3554 examine the content of the <iterateRequest>. If the request is valid, the provider MUST return  
3555 (the XML that represents) the next set of objects from the result set that the <iterator>  
3556 represents.

3557 **Execution.** The provider MUST execute the iterate operation synchronously (if the provider  
3558 executes the iterate operation at all). See the section titled “[Determining execution mode](#)”.

3559 **Response.** The provider MUST return to the requestor an <iterateResponse>.

3560 **Status.** The <iterateResponse> must contain a “status” attribute that indicates whether the  
3561 provider successfully returned the next set of objects from the result set that the <iterator>  
3562 represents. See the section titled “[Status \(normative\)](#)”.

3563 • If the provider successfully returned (the XML that represents) the next set of objects from the  
3564 result set that the <iterator> represents, then the <iterateResponse> MUST specify  
3565 “status=’ success’”.

3566 • If the provider encountered an error in returning (the XML that represents) the next set of  
3567 objects from the result set that the <iterator> represents, then the <iterateResponse>  
3568 MUST specify “status=’ failure’”.

3569 **PSO.** The <iterateResponse> MAY contain any number of <pso> elements.

3570 • If the <iterateResponse> specifies “status=’ success’” and *at least one object remained*  
3571 *to iterate* (in the result set that the <iterator> represents),  
3572 then the <iterateResponse> MUST contain at least one <pso> element  
3573 that contains the (XML representation of the) next matching object.

3574 • If the <iterateResponse> specifies “status=’ success’” and *no object remained to*  
3575 *iterate* (in the result set that the <iterator> represents),  
3576 then the <iterateResponse> MUST NOT contain a <pso> element.

3577 • If the <iterateResponse> specifies “status=’ failure’”,  
3578 then the <iterateResponse> MUST NOT contain a <pso> element.

3579 **PSO and ReturnData.** Each <pso> contains the subset of (the XML representation of) a requested  
3580 object that the “returnData” attribute of the original <searchRequest> specified. By default,  
3581 each <pso> contains the entire (XML representation of an) object.

3582 • A <pso> element MUST contain a <psoID> element.  
3583 The <psoID> element MUST contain the identifier of the requested object.  
3584 See the section titled “[PSO Identifier \(normative\)](#)”.

3585 • A <pso> element MAY contain a <data> element.

3586 - If the <searchRequest> specified “returnData=’ identifier’”,  
3587 then the <pso> MUST NOT contain a <data> element.



3588 - Otherwise, if the <searchRequest> specified "returnData=' data' "  
3589 or (if the <searchRequest> specified) "returnData=' everything' "  
3590 or (if the <searchRequest>) omitted the "returnData" attribute  
3591 then the <data> element MUST contain the XML representation of the object.  
3592 This XML must be valid according to the schema of the target for the schema entity of  
3593 which the newly created object is an instance.

3594 • A <pso> element MAY contain any number of <capabilityData> elements. Each  
3595 <capabilityData> element contains a set of *capability-specific data* that is associated with  
3596 the newly created object (for example, a *reference* to another object).

3597 - If the <searchRequest> specified "returnData=' identifier' "  
3598 or (if the <searchRequest> specified) "returnData=' data' "  
3599 then the <pso> MUST NOT contain a <capabilityData> element.

3600 - Otherwise, if the <searchRequest> specified "returnData=' everything' "  
3601 or (if the <searchRequest>) omitted the "returnData" attribute,  
3602 then the <pso> MUST contain a <capabilityData> element for each set of capability-  
3603 specific data that is associated with the requested object  
3604 (and that is specific to a capability that the target supports for the schema entity of which  
3605 the requested object is an instance).

3606 **PSO capabilityData and IncludeDataForCapability.** An <iterateResponse> MUST include (as  
3607 <capabilityData> sub-elements of each <pso>) any capability-specific data that is associated  
3608 with each matching object and for which *all* of the following are true:

3609 • The original <searchRequest> specified "returnData=' everything' "  
3610 or (the original <searchRequest>) omitted the "returnData" attribute.

3611 • The schema for the target declares that the *target supports the capability*  
3612 (for the schema entity of which each matching object is an instance).

3613 • The original <searchRequest> contained an <includeDataForCapability> element  
3614 that specified the capability to which the data are specific  
3615 or the original <searchRequest> contained no <includeDataForCapability> element.

3616 An <iterateResponse> SHOULD NOT include (as <capabilityData> sub-elements of each  
3617 <pso>) any capability-specific data for which any of the above is not true.

3618 **iterator.** A <iterateResponse> MAY contain at most one <iterator> element.

3619 • If the <iterateResponse> specifies "status=' success' " and the search response  
3620 *contains the last of the objects* that matched the <query> that was specified in the original  
3621 <searchRequest>, then the <iterateResponse> MUST NOT contain an <iterator>.

3622 • If the <iterateResponse> specifies "status=' success' " and the provider *still has more*  
3623 *matching objects* that have not yet been returned to the requestor, then the  
3624 <iterateResponse> MUST contain exactly one <iterator>.

3625 • If the <iterateResponse> specifies "status=' failure' ", then the <iterateResponse>  
3626 MUST NOT contain an <iterator>.

3627 **iterator ID.** An <iterator> MUST have an "ID" attribute.

3628 The value of the "ID" attribute uniquely identifies the <iterator> within the namespace of the  
3629 provider. The "ID" attribute allows the provider to map each <iterator> token to the result set of  
3630 the requestor's <query> and to any state that records the requestor's position within that result set.

3631 The “ID” attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an  
3632 <iterator>. An <iterator> is not a PSO.

3633 **Error.** If the <iterateResponse> specifies “status=’ failure’”, then the  
3634 <iterateResponse> MUST have an “error” attribute that characterizes the failure.  
3635 See the general section titled ““Error (normative)”” .

3636 In addition, the <iterateResponse> MUST specify an appropriate value of “error” if any of the  
3637 following is true:

- 3638 • If the provider does not recognize the <iterator> in an <iterateRequest> as representing  
3639 a result set.
- 3640 • If the provider does not recognize the <iterator> in an <iterateRequest> as representing  
3641 any result set that the provider currently maintains.

3642 The <iterateResponse> MAY specify an appropriate value of “error” if any of the following is  
3643 true:

- 3644 • If an <iterateRequest> contains an <iterator> that is *not the most recent version* of the  
3645 <iterator>. If the provider has returned to the requestor a more recent <iterator> that  
3646 represents the same search result set, then the provider MAY reject the older <iterator>.  
3647 (A provider that changes the ID—for example, to encode the state of iteration within a search  
3648 result set—may be sensitive to this.)

### 3649 **3.6.7.2.3** *iterate Examples (non-normative)*

3650 In order to illustrate the iterate operation, we first need a search operation that returns more than  
3651 one object. In the following example, a requestor asks a provider to search for every `Person` with  
3652 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>
```

3653 The provider returns a <searchResponse>. The “status” attribute of the <searchResponse>  
3654 indicates that the provider successfully executed the search operation. The <searchResponse>  
3655 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>
  </ps0>
</searchResponse>
```

```
<psoID ID="0002" targetID="target2"/>
</pso>
<iterator ID="1900"/>
</searchResponse>
```

3656 The requestor asks the provider to return the next matching objects (in the result set for the  
3657 search). The requestor supplies the <iterator> from the <searchResponse> as input to the  
3658 <iterateRequest>.

```
<iterateRequest requestID="148" >
  <iterator ID="1900"/>
</iterateRequest>
```

3659 The provider returns an <iterateResponse> in response to the <iterateRequest>. The  
3660 "status" attribute of the <iterateResponse> indicates that the provider successfully executed  
3661 the iterate operation. The <iterateResponse> contains two <pso> elements that represent the  
3662 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>
```

3663 The <iterateResponse> also contains another <iterator> element. The "ID" of this  
3664 <iterator> differs from the "ID" of the <iterator> in the original <searchResponse>. The  
3665 "ID" could remain constant (for each iteration of the result set that the <iterator> represents) if  
3666 the provider so chooses, but the "ID" value could change (e.g., if the provider uses "ID" to  
3667 encode the state of the result set).

3668 To get the final matching object, the requestor again supplies the <iterator> from the  
3669 <iterateResponse> as input to the <iterateRequest>.

```
<iterateRequest requestID="149">
  <iterator ID="1901"/>
</iterateRequest>
```

3670 The provider again returns an <iterateResponse> in response to the <iterateRequest>. The  
3671 "status" attribute of the <iterateResponse> indicates that the provider successfully executed  
3672 the iterate operation. The <iterateResponse> contains a <pso> element that represents the  
3673 final matching object. Since all of the matching objects have now been returned to the requestor,  
3674 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"/>
  </psol>
</iterateResponse>

```

### 3675 3.6.7.3 closeliterator

3676 The closeliterator operation tells the provider that the requestor has no further need for the search  
3677 result that a specific <iterator> represents. (See the description of the [search operation](#) above.)

3678 A requestor should send a <closeIteratorRequest> to the provider when the requestor no  
3679 longer intends to iterate a search result. (A provider will eventually free an inactive search result --  
3680 even if the provider never receives a <closeIteratorRequest> from the requestor-- but this  
3681 behavior is unspecified.) For more information, see the topic named "Resource Considerations"  
3682 topic earlier within this section.

3683 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"/>

```

3684 **A closeliteratorRequest receives a closeliteratorResponse.** A requestor supplies as input to a  
3685 <closeIteratorRequest> the <iterator> that was part of the original <searchResponse>  
3686 or the <iterator> that was part of a subsequent <iterateResponse>, whichever is most  
3687 recent. A provider returns a <closeIteratorResponse> in response to each  
3688 <closeIteratorRequest>. A <closeIteratorResponse> has the same structure as an  
3689 <spml:response>.

3690 **closeliterator is not batchable.** For reasons of scale, neither of a search request nor an iterate  
3691 request nor a closeliterator request should be nested in a [batch](#) request. When a search query

3692 matches more objects than the provider can place directly in the response, the provider must  
3693 temporarily store the remaining objects. Storing the remaining objects allows the requestor to  
3694 iterate the remaining objects, but also requires the provider to commit resources.  
3695 See the topic named "Resource Considerations" earlier in this section.

3696 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
3697 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
3698 search results. Allowing a requestor to nest a search request or an iterate request or a closeliterator  
3699 request within a batch request would aggravate the resource problem, requiring a provider to store  
3700 more information in larger chunks for a longer amount of time.

3701 **The closeliterator operation must be executed synchronously.** The provider is already queuing  
3702 the result set (every object beyond those returned in the first search response), so a request to  
3703 close the iterator (and thus to free the system resources associated with the result set) should be  
3704 executed as soon as possible. It is unreasonable for a requestor to ask the provider to queue the  
3705 results of a request to close an iterator (especially since the close iterator response contains little or  
3706 no information beyond success or failure).

### 3707 **3.6.7.3.1** *closeIteratorRequest (normative)*

3708 A requestor SHOULD send a `<closeIteratorRequest>` to a provider when the requestor no  
3709 longer intends to iterate a search result. (This allows the provider to free any system resources  
3710 associated with the search result.).

3711 **Execution.** A `<closeIteratorRequest>` MUST NOT specify  
3712 "executionMode='asynchronous'".  
3713 A `<closeIteratorRequest>` MUST specify "executionMode='synchronous' "  
3714 or (a `<closeIteratorRequest>` MUST) omit "executionMode".  
3715 See the section titled "[Determining execution mode](#)".

3716 **iterator.** A `<closeIteratorRequest>` MUST contain exactly one `<iterator>` element. A  
3717 requestor MUST supply as input to a `<closeIteratorRequest>` the `<iterator>` from the  
3718 original `<searchResponse>` or (a requestor MUST supply the `<iterator>` from a subsequent  
3719 `<iterateResponse>`). A requestor SHOULD supply as input to a  
3720 `<closeIteratorRequest>` the most recent `<iterator>` that represents the search result set.

3721 **iterator ID.** An `<iterator>` that is part of a `<closeIteratorRequest>` MUST have an "ID"  
3722 attribute. (The value of the "ID" attribute uniquely identifies the `<iterator>` within the  
3723 namespace of the provider. The "ID" attribute allows the provider to map each `<iterator>`  
3724 token to the result set of the requestor's `<query>` and also (allows the provider to map each  
3725 `<iterator>` token) to any state that records the requestor's iteration *within* that result set.)

### 3726 **3.6.7.3.2** *closeIteratorResponse (normative)*

3727 A provider that receives a `<closeIteratorRequest>` from a requestor that the provider trusts  
3728 must examine the content of the `<closeIteratorRequest>`. If the request is valid, the provider  
3729 MUST release any search result set that the `<iterator>` represents. Any subsequent request to  
3730 iterate that same search result set MUST fail.

3731 **Execution.** The provider MUST execute the closeliterator operation synchronously (if the provider  
3732 executes the closeliterator operation at all). See the section titled "[Determining execution mode](#)".

3733 **Response.** The provider MUST return to the requestor a `<closeIteratorResponse>`.

3734 **Status.** The `<closeIteratorResponse>` must contain a "status" attribute that indicates  
3735 whether the provider successfully released the search result set that the `<iterator>` represents.  
3736 See the section titled "[Status \(normative\)](#)".

3737 • If the provider successfully released the search result set that the `<iterator>` represents,  
3738 then the `<closeIteratorResponse>` MUST specify "status='success'".

3739 • If the provider encountered an error in releasing the search result set that the `<iterator>`  
3740 represents, then the `<closeIteratorResponse>` MUST specify "status='failure'".

3741 **Error.** If the `<closeIteratorResponse>` specifies "status='failure'", then the  
3742 `<closeIteratorResponse>` MUST have an "error" attribute that characterizes the failure.  
3743 See the general section titled "[Error \(normative\)](#)".

3744 In addition, the `<closeIteratorResponse>` MUST specify an appropriate value of "error" if  
3745 any of the following is true:

3746 • If the provider does not recognize the `<iterator>` in a `<closeIteratorRequest>` as  
3747 representing a search result set.

3748 • If the provider does not recognize the `<iterator>` in a `<closeIteratorRequest>` as  
3749 representing any search result set that the provider currently maintains.

3750 • If the provider recognized the `<iterator>` in a `<closeIteratorRequest>` as representing  
3751 a search result set that the provider currently maintains but *cannot release the resources*  
3752 *associated with that search result set*.

3753 The `<closeIteratorResponse>` MAY specify an appropriate value of "error" if any of the  
3754 following is true:

3755 • If a `<closeIteratorRequest>` contains an `<iterator>` that is *not the most recent version*  
3756 *of the <iterator>*. If the provider has returned to the requestor a more recent `<iterator>`  
3757 that represents the same search result set, then the provider MAY reject the older  
3758 `<iterator>`.

3759 (A provider that changes the ID—for example, to encode the state of iteration within a search  
3760 result set—may be sensitive to this.)

### 3761 [3.6.7.3.3 closeIterator Examples \(non-normative\)](#)

3762 In order to illustrate the `closeIterator` operation, we first need a search operation that returns more  
3763 than one object. In the following example, a requestor asks a provider to search for every `Person`  
3764 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>
```

3765 The provider returns a `<searchResponse>`. The "status" attribute of the `<searchResponse>`  
3766 indicates that the provider successfully executed the search operation. The `<searchResponse>`  
3767 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>
    <psoID ID="0001" targetID="target2"/>
  </pso>
  <pso>
    <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>

```

3768 The requestor decides that the two objects in the initial <searchResponse> will suffice, and does  
3769 not intend to retrieve any more matching objects (in the result set for the search). The requestor  
3770 supplies the <iterator> from the <searchResponse> as input to the  
3771 <closeIteratorRequest>.

```

<closeIteratorRequest requestID="151">
  <iterator ID="1900"/>
</closeIteratorRequest>

```

3772 The provider returns a <closeIteratorResponse> in response to the  
3773 <closeIteratorRequest>. The "status" attribute of the <closeIteratorResponse>  
3774 indicates that the provider successfully released the result set.

```

<closeIteratorResponse requestID="151" status="success"/>

```



### 3775 **3.6.8 Suspend Capability**

3776 The Suspend Capability is defined in a schema associated with the following XML namespace:  
3777 `urn:oasis:names:tc:SPML:2:0:suspend`. This document includes the Suspend Capability  
3778 XSD as Appendix H.

3779 The Suspend Capability defines three operations: suspend, resume and active.

3780 • The suspend operation *disables an object* (immediately or on a specified date).

3781 • The resume operation *re-enables an object* (immediately or on a specified date).

3782 • The active operation *tests whether an object is currently suspended*.

3783 The suspend operation disables an object *persistently* (rather than transiently). The suspend  
3784 operation is intended to revoke the privileges of an account, for example, while the authorized user  
3785 of the account is on vacation.

3786 The resume operation re-enables an object persistently. One might use the resume operation to  
3787 restore privileges for an account, for example, when the authorized user of the account returns from  
3788 vacation.

3789 A provider that supports the suspend, resume and active operations for a target SHOULD declare  
3790 that the target supports the Suspend Capability. A provider that does not support all of suspend,  
3791 resume and active MUST NOT declare that the target supports the Suspend Capability.

3792 **Idempotent.** The suspend operation and the resume operation are both *idempotent*. Any requestor  
3793 should be able to suspend (or to resume) the same object multiple times without error.

3794 **Search.** A requestor can *search for objects based on enabled state* using the `<isActive>` query  
3795 clause. The `{IsActiveType}` extends `{QueryClauseType}`, which indicates that an instance  
3796 of `{IsActiveType}` can be used to select objects. An `<isActive>` clause matches an object if  
3797 and only if the object is currently enabled. In order to select disabled objects, a requestor would  
3798 combine this clause with the logical operator `<not>`. See the section titled "[Selection](#)".

#### 3799 **3.6.8.1 suspend**

3800 The suspend operation enables a requestor to disable an object.

3801 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"/>
```

3802 **3.6.8.1.1 suspendRequest (normative)**

3803 A requestor MUST send a <suspendRequest> to a provider in order to (ask the provider to)  
3804 disable an existing object.

3805 **Execution.** A <suspendRequest> MAY specify "executionMode".  
3806 See the section titled "Determining execution mode".

3807 **psoid.** A <suspendRequest> MUST contain exactly one <psoid> element. A <psoid> element  
3808 MUST identify an object that exists on a target that is exposed by the provider.  
3809 See the section titled "PSO Identifier (normative)".

3810 **EffectiveDate.** A <suspendRequest> MAY specify an "effectiveDate". Any  
3811 "effectiveDate" value MUST be expressed in UTC form, with no time zone component.  
3812 A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.  
3813 A requestor MUST NOT generate time instants that specify leap seconds.

3814 **3.6.8.1.2 suspendResponse (normative)**

3815 A provider that receives a <suspendRequest> from a requestor that the provider trusts MUST  
3816 examine the content of the <suspendRequest>. If the request is valid and if the specified object  
3817 exists, then the provider MUST disable the object that the <psoid> specifies.

3818 If the <suspendRequest> specifies an "effectiveDate", the provider MUST enable the  
3819 specified object as of that date.

- 3820 • If the "effectiveDate" of the <suspendRequest> is in the past, then  
3821 the provider MUST do one of the following:
- 3822 - The provider MAY disable the specified object *immediately*.
  - 3823 - The provider MAY return an error. (The provider's response SHOULD indicate that the  
3824 request failed because the effective date is past.)
- 3825 • If the "effectiveDate" of the <suspendRequest> is in the future, then
- 3826 - The provider MUST NOT disable the specified object until that future date and time.
  - 3827 - The provider MUST disable the specified object at that future date and time  
3828 (unless a subsequent request countermands this request).

3829 **Execution.** If an <suspendRequest> does not specify "executionMode",  
3830 the provider MUST choose a type of execution for the requested operation.  
3831 See the section titled "Determining execution mode".

3832 **Response.** The provider must return to the requestor a <suspendResponse>. The  
3833 <suspendResponse> must have a "status" attribute that indicates whether the provider  
3834 successfully disabled the specified object. See the section titled "Status (normative)".

3835 **Error.** If the provider cannot create the requested object, the <suspendResponse> must contain  
3836 an `error` attribute that characterizes the failure. See the general section titled "Error (normative)".

3837 In addition, the <suspendResponse> MUST specify an appropriate value of "error" if any of the  
3838 following is true:

- 3839 • The <suspendRequest> contains a <psoid> for an object that does not exist.
- 3840 • The <suspendRequest> specifies an "effectiveDate" that is not valid.

3841 The provider MAY return an error if any of the following is true:

- 3842 • The `<suspendRequest>` specifies an "effectiveDate" that is in the past.
- 3843 The provider MUST NOT return an error when (the operation would otherwise succeed and) the  
 3844 object is already disabled. In this case, the `<suspendResponse>` MUST specify  
 3845 "status='success'".

### 3846 3.6.8.1.3 *suspend Examples (non-normative)*

3847 In the following example, a requestor asks a provider to suspend an existing `Person` object.

```
<suspendRequest requestID="139">
  <psolD ID="2244" targetID="target2"/>
</suspendRequest>
```

3848 The provider returns an `<suspendResponse>` element. The "status" attribute of the  
 3849 `<suspendResponse>` indicates that the provider successfully disabled the specified object.

```
<suspendResponse requestID="139" status="success" />
```

3850 In the following example, a requestor asks a provider to suspend an existing account.

```
<suspendRequest requestID="140" >
  <psolD ID="1431" targetID="target1"/>
</suspendRequest>
```

3851 The provider returns a `<suspendResponse>`. The "status" attribute of the  
 3852 `<suspendResponse>` indicates that the provider successfully disabled the specified account.

```
<suspendResponse requestID="140" status="success"/>
```

### 3853 3.6.8.2 **resume**

3854 The resume operation enables a requestor to re-enable an object that has been suspended. (See  
 3855 the description of the [suspend](#) operation above.)

3856 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="psolD" 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"/>
```

#### 3857 3.6.8.2.1 *resumeRequest (normative)*

3858 A requestor MUST send a `<resumeRequest>` to a provider in order to (ask the provider to) re-  
 3859 enable an existing object.

3860 **Execution.** A <resumeRequest> MAY specify "executionMode".  
3861 See the section titled "Determining execution mode".

3862 **psoid.** A <resumeRequest> MUST contain exactly one <psoid> element. A <psoid> element  
3863 MUST identify an object that exists on a target (that is supported by the provider).  
3864 See the section titled "PSO Identifier (normative)".

3865 **EffectiveDate.** A <resumeRequest> MAY specify an "effectiveDate". Any  
3866 "effectiveDate" value MUST be expressed in UTC form, with no time zone component.  
3867 A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.  
3868 A requestor MUST NOT generate time instants that specify leap seconds.

### 3869 3.6.8.2.2 resumeResponse (normative)

3870 A provider that receives a <resumeRequest> from a requestor that the provider trusts MUST  
3871 examine the content of the <resumeRequest>. If the request is valid and if the specified object  
3872 exists, then the provider MUST enable the object that is specified by the <psoid>.

3873 If the <resumeRequest> specifies an "effectiveDate", the provider MUST enable the  
3874 specified object as of that date.

3875 • If the "effectiveDate" of the <resumeRequest> is in the past, then  
3876 the provider MUST do one of the following:  
3877 - The provider MAY enable the specified object *immediately*.  
3878 - The provider MAY return an error. (The provider's response SHOULD indicate that the  
3879 request failed because the effective date is past.)

3880 • If the "effectiveDate" of the <resumeRequest> is in the future, then  
3881 - The provider MUST NOT enable the specified object until that future date and time.  
3882 - The provider MUST enable the specified object at that future date and time  
3883 (unless a subsequent request countermands this request).

3884 **Execution.** If an <resumeRequest> does not specify "executionMode",  
3885 the provider MUST choose a type of execution for the requested operation.  
3886 See the section titled "Determining execution mode".

3887 **Response.** The provider must return to the requestor a <resumeResponse>. The  
3888 <resumeResponse> must have a "status" attribute that indicates whether the provider  
3889 successfully enabled the specified object. See the section titled "Status (normative)".

3890 **Error.** If the provider cannot enable the requested object, the <resumeResponse> must contain  
3891 an error attribute that characterizes the failure. See the general section titled "Error (normative)".

3892 In addition, the <resumeResponse> MUST specify an appropriate value of "error" if any of the  
3893 following is true:

- 3894 • The <resumeRequest> contains a <psoid> for an object that does not exist.
- 3895 • The <resumeRequest> specifies an "effectiveDate" that is not valid.

3896 The provider MAY return an error if any of the following is true:

- 3897 • The <resumeRequest> specifies an "effectiveDate" that is in the past.

3898 The provider MUST NOT return an error when (the operation would otherwise succeed and) the  
3899 object is already enabled. In this case, the response should specify "status='success'".

3900 **3.6.8.2.3** *resume Examples (non-normative)*

3901 In the following example, a requestor asks a provider to resume an existing `Person` object.

```
<resumeRequest requestID="141">  
  <psoid ID="2244" targetID="target2"/>  
</resumeRequest>
```

3902 The provider returns a `<resumeResponse>` element. The `"status"` attribute of the  
3903 `<resumeResponse>` element indicates that the provider successfully disabled the specified object.

```
<resumeResponse requestID="141" status="success"/>
```

3904 In the following example, a requestor asks a provider to resume an existing account.

```
<resumeRequest requestID="142">  
  <psoid ID="1431" targetID="target1"/>  
</resumeRequest>
```

3905 The provider returns a `<resumeResponse>`. The `"status"` attribute of the  
3906 `<resumeResponse>` indicates that the provider successfully enabled the specified account.

```
<resumeResponse requestID="142" status="success"/>
```

3907 **3.6.8.3** *active*

3908 The active operation enables a requestor to determine whether a specified object has been  
3909 suspended. (See the description of the [suspend](#) operation above.)

3910 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"/>
```

3911 **3.6.8.3.1** *activeRequest (normative)*

3912 A requestor **MUST** send an `<activeRequest>` to a provider in order to (ask the provider to)  
3913 determine whether the specified object is enabled (active) or disabled.

3914 **Execution.** An <activeRequest> MAY specify "executionMode".  
3915 See the section titled "[Determining execution mode](#)".

3916 **psoid.** A <activeRequest> MUST contain exactly one <psoid> element. A <psoid> element  
3917 MUST identify an object that exists on a target that is exposed by the provider.  
3918 See the section titled "[PSO Identifier \(normative\)](#)".

### 3919 **3.6.8.3.2** *activeResponse (normative)*

3920 A provider that receives a <activeRequest> from a requestor that the provider trusts MUST  
3921 examine the content of the <activeRequest>. If the request is valid and if the specified object  
3922 exists, then the provider MUST disable the object that is specified by the <psoid>.

3923 **Execution.** If an <activeRequest> does not specify "executionMode", the provider MUST  
3924 choose a type of execution for the requested operation.  
3925 See the section titled "[Determining execution mode](#)".

3926 **Response.** The provider must return to the requestor an <activeResponse>. The  
3927 <activeResponse> must have a "status" attribute that indicates whether the provider  
3928 successfully determined whether the specified object is enabled (i.e. active).  
3929 See the section titled "[Status \(normative\)](#)".

3930 **active.** An <activeResponse> MAY have an "active" attribute that indicates whether the  
3931 specified object is suspended. An <activeResponse> that specifies "status=' success' "  
3932 MUST have an "active" attribute.

3933 • If the specified object is suspended, the <activeResponse> MUST specify  
3934 "active=' false'".

3935 • If the specified object is not suspended, the <activeResponse> MUST specify  
3936 "active=' true'".

3937 **Error.** If the provider cannot determine whether the requested object is suspended, the  
3938 <activeResponse> must contain an "error" attribute that characterizes the failure.  
3939 See the general section titled "[Error \(normative\)](#)".

3940 In addition, the <activeResponse> MUST specify an appropriate value of "error" if any of the  
3941 following is true:

3942 • The <activeRequest> contains a <psoid> that specifies an object that does not exist.

### 3943 **3.6.8.3.3** *active Examples (non-normative)*

3944 In the following example, a requestor asks a provider whether a `Person` object is active.

```
<activeRequest requestID="143">  
  <psoid ID="2244" targetID="target2"/>  
</activeRequest>
```

3945 The provider returns an <activeResponse> element. The "status" attribute of the  
3946 <activeResponse> element indicates that the provider successfully completed the requested  
3947 operation. The "active" attribute of the <activeResponse> indicates that the specified object is  
3948 active.

```
<activeResponse requestID="143" status="success" active="true"/>
```

3949 In the following example, a requestor asks a provider whether an account is active.

```
<activeRequest requestID="144">  
  <psolD ID="1431" targetID="target1"/>  
</activeRequest>
```

- 3950 The provider returns an <activeResponse>. The "status" attribute of the  
3951 <activeResponse> indicates that the provider successfully completed the requested operation.  
3952 The "active" attribute of the <activeResponse> indicates that the specified object is active.

```
<activeResponse requestID="144" status="success" active="true"/>
```



### 3953 3.6.9 Updates Capability

3954 The Updates Capability is defined in a schema associated with the following XML namespace:  
3955 `urn:oasis:names:tc:SPML:2:0:updates`. This document includes the Updates Capability  
3956 XSD as Appendix I.

3957 The Updates Capability defines three operations: `updates`, `iterate` and `closeiterator`. The updates  
3958 and iterate operations together allow a requestor to obtain *in a scalable manner* every recorded  
3959 *update* (i.e., modification to an object) that matches specified selection criteria. The updates  
3960 operation returns in its response a first set of matching updates. Each subsequent iterate operation  
3961 returns more matching updates. The `closeiterator` operation allows a requestor to tell a provider that  
3962 it does not intend to finish iterating a result set and that the provider may therefore release the  
3963 associated resources).

3964 A provider that supports the updates and iterate operations for a target SHOULD declare that the  
3965 target supports the Updates Capability. A provider that does not support both updates and iterate  
3966 MUST NOT declare that the target supports the Updates Capability.

3967 **Resource considerations.** A provider must limit the size and duration of its updates result sets (or  
3968 that provider will exhaust available resources). A provider must decide:

- 3969 • How large of an updates result set the provider will *select* on behalf of a requestor.
- 3970 • How large of an updates result set the provider will *queue* on behalf of a requestor  
3971 (so that the requestor may iterate the updates result set).
- 3972 • For *how long a time* the provider will queue an updates result set on behalf of a requestor.

3973 These decisions may be governed by the provider's implementation, by its configuration, or by  
3974 runtime computation.

3975 A provider that wishes to *never to queue updates result sets* may return every matching object (up  
3976 to the provider's limit and up to any limit that the request specifies) in the updates response. Such  
3977 a provider would never return an iterator, and would not need to support the iterate operation. The  
3978 disadvantage is that, without an iterate operation, a provider's updates capability either is limited to  
3979 small results or produces large updates responses.

3980 A provider that wishes to support the iterate operation must store (or somehow queue) the updates  
3981 selected by an updates operation until the requestor has a chance to iterate those results. (That is,  
3982 a provider must somehow queue the updates that matched the criteria of an updates operation and  
3983 that were not returned in the updates response.)

3984 If all goes well, the requestor will continue to iterate the updates result set until the provider has  
3985 sent all of the updates to the requestor. The requestor may also use the `closeiterator` operation to  
3986 tell the provider that the requestor is no longer interested in the search result. Once all of the  
3987 updates have been sent to the requestor, the provider may free any resource that is still associated  
3988 with the updates result set. However, it is possible that the requestor may not iterate the updates  
3989 result set in a timely manner--or that the requestor may *never* iterate the updates result set  
3990 completely. Such a requestor may also neglect to close the iterator.

3991 A provider cannot queue updates result sets indefinitely. The provider must eventually release the  
3992 resources associated with an updates result set. (Put differently, any iterator that a provider returns  
3993 to a requestor must eventually expire.) Otherwise, the provider may run out of resources.

3994 Providers should carefully manage the resources associated with updates result sets. For example:

- 3995 • A provider may define a *timeout interval* that specifies the maximum time between iterate  
3996 requests. If a requestor does not request an iterate operation within this interval, the provider  
3997 will release the resources associated with the result set. This invalidates any iterator that  
3998 represents this result set.
- 3999 • A provider may also define an overall *result lifetime* that specifies the maximum length of time  
4000 to retain a result set. After this amount of time has passed, the provider will release the result  
4001 set.
- 4002 • A provider may also wish to enforce an *overall limit* on the resources available to queue result  
4003 sets, and may wish to adjust its behavior (or even to refuse updates requests) accordingly.
- 4004 • To prevent denial of service attacks, the provider should not allocate any resource on behalf of  
4005 a requestor until that requestor is properly authenticated.  
4006 See the section titled "[Security and Privacy Considerations](#)".

### 4007 3.6.9.1 updates

4008 The updates operation obtains *records of changes to objects*. A requestor may select change  
4009 records based on changed-related criteria and (may also select change records) based on the set  
4010 of objects.

4011 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"/>

```

4012 The <query> is the same type of element that is specified as part of a <bulkModifyRequest> or  
4013 a <bulkDeleteRequest> or a <searchRequest>. This <query> selects the objects for which  
4014 the provider will return recorded updates. See the section titled "[SearchQueryType](#)".

4015 The "updatedSince" attribute allows the requestor to select only updates that occurred since a  
4016 specific date and time.

4017 If the updates operation is successful *but selects no matching update*, the <updatesResponse>  
4018 will not contain an <update>.

4019 If the updates operation is successful *and selects at least one matching update*, the  
4020 <updatesResponse> will contain any number of <update> elements, each of which represents a  
4021 matching update. If the updates operation selects more matching updates than the  
4022 <updatesResponse> contains, the <updatesResponse> will also contain an <iterator> that  
4023 the requestor can use to retrieve more matching updates. (See the description of the [iterate](#)  
4024 operation below.)

4025 If an updates operation would select more updates than the provider can queue for subsequent  
4026 iteration by the requestor, the provider's <updatesResponse> will specify  
4027 "error='resultSetTooLarge'".

4028 **Updates is not batchable.** For reasons of scale, neither an updates request nor an iterate request  
4029 should be nested in a [batch](#) request. When an updates query matches more updates than the  
4030 provider can place directly in the response, the provider must temporarily store the remaining  
4031 updates. Storing the remaining updates allows the requestor to iterate the remaining updates, but  
4032 also requires the provider to commit resources.  
4033 See the topic named "Resource Considerations" earlier in this section.

4034 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
4035 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
4036 updates result sets. Allowing a requestor to nest an updates request within a batch request would  
4037 aggravate the resource problem, requiring a provider to store more information in larger chunks for  
4038 a longer amount of time.

### 4039 [3.6.9.1.1 updatesRequest \(normative\)](#)

4040 A requestor **MUST** send an `<updatesRequest>` to a provider in order to (ask the provider to)  
4041 obtain every update that matches specified selection criteria.

4042 **Execution.** An `<updatesRequest>` **MAY** specify "executionMode".  
4043 See the section titled "[Determining execution mode](#)".

4044 **query.** A `<query>` describes criteria that (the provider must use to) select objects on a target.  
4045 The provider will return only updates that affect objects that match these criteria.  
4046 An `<updatesRequest>` **MAY** contain at most one `<query>` element.

- 4047 • If the provider's `<listTargetsResponse>` contains only a single `<target>`,  
4048 then an `<updatesRequest>` may omit the `<query>` element.
- 4049 • If the provider's `<listTargetsResponse>` contains more than one `<target>`,  
4050 then an `<updatesRequest>` **MUST** contain exactly one `<query>` element  
4051 and that `<query>` must specify "targetID".

4052 See the section titled "[SearchQueryType in a Request \(normative\)](#)".

4053 **updatedByCapability.** An `<updatesRequest>` **MAY** contain any number of  
4054 `<updatedByCapability>` elements. Each `<updatedByCapability>` element contains the  
4055 URN of an XML namespace that uniquely identifies a capability. Each `<updatedByCapability>`  
4056 element must identify a capability that the target supports.

- 4057 • A requestor that wants the provider to return no update that reflects a change to capability-  
4058 specific data associated with an object **MUST NOT** place an `<updatedByCapability>`  
4059 element in its `<updatesRequest>`.
- 4060 • A requestor that wants the provider to return updates that reflect changes to capability-specific  
4061 data associated with one or more objects **MUST** specify each capability (for which the provider  
4062 should return updates) as an `<updatedByCapability>` element in its `<updatesRequest>`.

4063 **updatedSince.** A `<updatesRequest>` **MAY** have an "updatedSince" attribute. (The provider  
4064 will return only updates with a timestamp greater than this value.)

4065 Any "updatedSince" value **MUST** be expressed in UTC form, with no time zone component.  
4066 A requestor or a provider **SHOULD NOT** rely on time resolution finer than milliseconds.  
4067 A requestor **MUST NOT** generate time instants that specify leap seconds.

4068 **maxSelect.** An `<updatesRequest>` **MAY** have a "maxSelect" attribute. The value of the  
4069 "maxSelect" attribute specifies the maximum number of updates the provider should select.

4070 **token.** An <updatesRequest> MAY have a "token" attribute. Any "token" value MUST  
4071 match a value that the provider returned to the requestor as the value of the "token" attribute in a  
4072 previous <updatesResponse> for the same target. Any "token" value SHOULD match the  
4073 (value of the "token" attribute in the) provider's *most recent* <updatesResponse> for the same  
4074 target.

### 4075 3.6.9.1.2 updatesResponse (normative)

4076 A provider that receives an <updatesRequest> from a requestor that the provider trusts must  
4077 examine the content of the <updatesRequest>. If the request is valid, the provider MUST return  
4078 updates that represent every change (that occurred since any time specified as "updatedSince")  
4079 to every object that matches the specified <query> (if the provider can possibly do so). However,  
4080 the number of updates selected (for immediate return or for eventual iteration) MUST NOT exceed  
4081 any limit specified as "maxSelect" in the <updatesRequest>.

4082 **Execution.** If an <updatesRequest> does not specify "executionMode",  
4083 the provider MUST choose a type of execution for the requested operation.  
4084 See the section titled "Determining execution mode".

4085 A provider SHOULD execute an updates operation synchronously if it is possible to do so. (The  
4086 reason for this is that the result of an updates should reflect the set of changes currently recorded  
4087 for each matching object. Other operations are more likely to intervene if an updates operation is  
4088 executed asynchronously.)

4089 **Response.** The provider MUST return to the requestor a <updatesResponse>.

4090 **Status.** The <updatesResponse> must contain a "status" attribute that indicates whether the  
4091 provider successfully selected every object that matched the specified query.  
4092 See the section titled "Status (normative)" for values of this attribute.

- 4093 • If the provider successfully returned every update that occurred (since any time specified by  
4094 "updatedSince") to every object that matched the specified <query>  
4095 up to any limit specified by the value of the "maxSelect" attribute,  
4096 then the <updatesResponse> MUST specify "status='success'".
- 4097 • If the provider encountered an error in selecting any object that matched the specified <query>  
4098 or (if the provider encountered an error) in returning any of the selected updates, then the  
4099 <updatesResponse> MUST specify "status='failure'".

4100 **Update.** The <updatesResponse> MAY contain any number of <update> elements.

- 4101 • If the <updatesResponse> specifies "status='success'" and *at least one update matched*  
4102 the specified criteria, then the <updatesResponse> MUST contain at least one <update>  
4103 element that describes a change to a matching object.
- 4104 • If the <updatesResponse> specifies "status='success'" and *no object matched the*  
4105 specified criteria, then the <updatesResponse> MUST NOT contain an <update> element.
- 4106 • If the <updatesResponse> specifies "status='failure'", then the <updatesResponse>  
4107 MUST NOT contain an <update> element.

4108 **Update Psoid.** Each <update> MUST contain exactly one <psoid> element. Each <psoid>  
4109 element uniquely identifies the object that was changed.

4110 **Update timestamp.** Each <update> must have a "timestamp" attribute that specifies when the  
4111 object was changed.

- 4112 Any "timestamp" value MUST be expressed in UTC form, with no time zone component.  
 4113 A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.
- 4114 **Update updateKind.** Each <update> must have an "updateKind" attribute that describes how  
 4115 the object was changed.
- 4116 • If the <update> specifies "updateKind='add'", then the object was added.
  - 4117 • If the <update> specifies "updateKind='modify'",  
 4118 then the (schema-defined XML data that represents the) object was modified.
  - 4119 • If the <update> specifies "updateKind='delete'", then the object was deleted.
  - 4120 • If the <update> specifies "updateKind='capability'",  
 4121 then a set of capability-specific data that is (or was) associated with the object was modified.
- 4122 **Update wasUpdatedByCapability.** Each <update> MAY have a "wasUpdatedByCapability"  
 4123 attribute that identifies the capability for which data (specific to that capability and associated with  
 4124 the object) was changed.
- 4125 • An <update> that specifies "updateKind='capability'"  
 4126 MUST have a "wasUpdatedByCapability" attribute.
  - 4127 • An <update> that specifies "updateKind='add'" or (that specifies)  
 4128 "updateKind='modify'" or (that specifies) "updateKind='delete'"  
 4129 MUST NOT have a "wasUpdatedByCapability" attribute.
  - 4130 • The value of each "wasUpdatedByCapability" MUST be the URN of an XML namespace  
 4131 that uniquely identifies a capability. Each "wasUpdatedByCapability" attribute MUST  
 4132 identify a capability that the target supports.
- 4133 **iterator.** A <updatesResponse> MAY contain at most one <iterator> element.
- 4134 • If the <updatesResponse> specifies "status='success'" and the updates response  
 4135 *contains all of the objects* that matched the specified <query>, then the  
 4136 <updatesResponse> MUST NOT contain an <iterator>.
  - 4137 • If the <updatesResponse> specifies "status='success'" and the updates response  
 4138 *contains some but not all of the objects* that matched the specified <query>, then the  
 4139 <updatesResponse> MUST contain exactly one <iterator>.
  - 4140 • If the <updatesResponse> specifies "status='success'" and *no object matched* the  
 4141 specified <query>, then the <updatesResponse> MUST NOT contain an <iterator>.
  - 4142 • If the <updatesResponse> specifies "status='failure'", then the <updatesResponse>  
 4143 MUST NOT contain an <iterator>.
- 4144 **iterator ID.** An <iterator> MUST have an "ID" attribute.
- 4145 The value of the "ID" attribute uniquely identifies the <iterator> within the namespace of the  
 4146 provider. The "ID" attribute allows the provider to map each <iterator> token to the result set of  
 4147 the requestor's <query> and to any state that records the requestor's position within that result set.
- 4148 The "ID" attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an  
 4149 <iterator>. An <iterator> is not a PSO.

4150 **token.** An <updatesResponse> MAY have a "token" attribute. (The requestor may pass this  
4151 "token" value in the next <updatesRequest> for the same target. See the topic named "token"  
4152 within the section titled "[UpdatesRequest](#)" above.)

4153 **Error.** If the <updatesResponse> specifies "status='failure'", then the  
4154 <updatesResponse> MUST have an "error" attribute that characterizes the failure.  
4155 See the general section titled "[Error \(normative\)](#)".

4156 The section titled "[SearchQueryType Errors \(normative\)](#)" describes errors specific to a request that  
4157 contains a <query>. Also see the section titled "[SelectionType Errors \(normative\)](#)".  
4158 In addition, the <updatesResponse> MUST specify an appropriate value of "error" if any of the  
4159 following is true:

- 4160 • If the *number of updates that matched* the criteria that were specified in an  
4161 <updatesRequest> *exceeds any limit on the part of the provider.* (but does not exceed any  
4162 value of "maxSelect" that the requestor specified as part of the <query>).  
4163 In this case, the provider's <updatesResponse> SHOULD specify  
4164 "error='resultSetTooLarge'".

### 4165 [3.6.9.1.3 updates Examples \(non-normative\)](#)

4166 In the following example, a requestor asks a provider to updates for every Person with an email  
4167 address matching "joebob@example.com". The requestor includes no <updatedByCapability>  
4168 element, which indicates that only updates to the schema-defined data for each matching object  
4169 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>
```

4170 The provider returns a <updatesResponse>. The "status" attribute of the  
4171 <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>
```

4172 The requestor next asks the provider to include capability-specific updates (i.e., recorded changes  
4173 to capability-specific data items that are associated with each matching object). The requestor  
4174 indicates interest in updates specific to the reference capability and (indicates interest in updates  
4175 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>
```

4176 The provider returns a <updatesResponse>. The "status" attribute of the  
4177 <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>

```

4178 This time the provider's response contains two updates: the "modify" update from the original  
4179 response plus a second "capability" update that is specific to the Reference Capability.

### 4180 3.6.9.2 iterate

4181 The iterate operation obtains the next set of objects from the result set that the provider selected for  
4182 a updates operation. (See the description of the [updates operation](#) above.)

4183 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"/>

```

4184 **An iterateRequest receives an iterateResponse.** A requestor supplies as input to an  
4185 <iterateRequest> the <iterator> that was part of the original <updatesResponse> or the  
4186 <iterator> that was part of a subsequent <iterateResponse>, whichever is most recent. A  
4187 provider returns an <iterateResponse> in response to each <iterateRequest>. An  
4188 <iterateResponse> has the same structure as a <updatesResponse>.

4189 The <iterateResponse> will contain at least one <update> element that records a change to  
4190 an object. If more matching updates are available to return, then the <iterateResponse> will  
4191 also contain an <iterator>. The requestor can use this <iterator> in another  
4192 <iterateRequest> to retrieve more of the matching objects.

4193 **Iterate is not batchable.** For reasons of scale, neither an updates request nor an iterate request  
4194 should be nested in a [batch](#) request. When an updates query matches more updates than the  
4195 provider can place directly in the response, the provider must temporarily store the remaining  
4196 updates. Storing the remaining updates allows the requestor to iterate the remaining updates, but  
4197 also requires the provider to commit resources.  
4198 See the topic named "Resource Considerations" earlier in this [Updates Capability](#) section.

4199 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
4200 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
4201 updates result sets. Allowing a requestor to nest a updates request or an iterate request within a  
4202 batch request would aggravate the resource problem, requiring a provider to store more information  
4203 in larger chunks for a longer amount of time.

4204 **The iterate operation must be executed synchronously.** The provider is already queuing the  
4205 result set (every update beyond those returned in the first updates response), so it is unreasonable  
4206 for a requestor to ask the provider to queue the results of a request for the next item in the result  
4207 set.

4208 Furthermore, asynchronous iteration would complicate the provider's maintenance of the result set.  
4209 Since a provider could never know that the requestor had processed the results of an  
4210 asynchronous iteration, the provider would not know when to increment its position in the result set.  
4211 In order to support asynchronous iteration both correctly and generally, a provider would have to  
4212 maintain a version of every result set for each iteration of that result set. This would impose an  
4213 unreasonable burden on the provider.

#### 4214 **3.6.9.2.1** *iterateRequest (normative)*

4215 A requestor MUST send an <iterateRequest> to a provider in order to obtain any *additional*  
4216 objects that matched a previous <updatesRequest> but that the provider has not yet returned to  
4217 the requestor. (That is, matching objects that were not contained in the response to that  
4218 <updatesRequest> and that have not yet been contained in any response to an  
4219 <iterateRequest> associated with that <updatesRequest>.)

4220 **Execution.** An <iterateRequest> MUST NOT specify "executionMode='asynchronous'".  
4221 An <iterateRequest> MUST specify "executionMode='synchronous' " or (an  
4222 <iterateRequest> MUST) omit "executionMode".  
4223 See the section titled "[Determining execution mode](#)".

4224 **iterator.** An <iterateRequest> MUST contain exactly one <iterator> element. A requestor  
4225 MUST supply as input to an <iterateRequest> the <iterator> from the original  
4226 <searchResponse> or (the requestor MUST supply as input to the <iterateRequest>) the  
4227 <iterator> from a subsequent <iterateResponse>. A requestor SHOULD supply as input  
4228 to an <iterateRequest> the most recent <iterator> that represents the updates result set.

#### 4229 **3.6.9.2.2** *iterateResponse (normative)*

4230 A provider that receives a <iterateRequest> from a requestor that the provider trusts must  
4231 examine the content of the <iterateRequest>. If the request is valid, the provider MUST return  
4232 (the XML that represents) the next object in the result set that the <iterator> represents.

4233 **Execution.** The provider MUST execute the iterate operation synchronously (if the provider  
4234 executes the iterate operation at all). See the section titled "[Determining execution mode](#)".

4235 **Response.** The provider MUST return to the requestor an <iterateResponse>.

4236 **Status.** The <iterateResponse> must contain a "status" attribute that indicates whether the  
4237 provider successfully returned the next update from the result set that the <iterator> represents.  
4238 See the section titled "[Status \(normative\)](#)".

4239 • If the provider successfully returned (the XML that represents) the next update from the result  
4240 set that the <iterator> represents, then the <iterateResponse> MUST specify  
4241 "status='success'".

4242 • If the provider encountered an error in returning (the XML that represents) the next update from  
4243 the result set that the <iterator> represents, then the <iterateResponse> MUST specify  
4244 "status='failure'".

4245 **Update.** The <iterateResponse> MAY contain any number of <update> elements.

4246 • If the <iterateResponse> specifies "status='success'" and *at least one update*  
4247 *remained to iterate* (in the updates result set that the <iterator> represents), then the  
4248 <iterateResponse> MUST contain at least one <update> element that records a change to  
4249 an object.

- 4250 • If the `<iterateResponse>` specifies “status=' success' ” and *no update remained to*  
 4251 *iterate* (in the updates result set that the `<iterator>` represents), then the  
 4252 `<iterateResponse>` MUST NOT contain an `<update>` element.
- 4253 • If the `<iterateResponse>` specifies “status=' failure' ”, then the `<iterateResponse>`  
 4254 MUST NOT contain an `<update>` element.
- 4255 **iterator.** A `<iterateResponse>` to an `<iterateRequest>` MAY contain at most one  
 4256 `<iterator>` element.
- 4257 • If the `<iterateResponse>` specifies “status=' success' ” and the `<iterateResponse>`  
 4258 *contains the last of the updates* that matched the criteria that the original `<updatesRequest>`  
 4259 specified, then the `<updatesResponse>` MUST NOT contain an `<iterator>`.
- 4260 • If the `<iterateResponse>` specifies “status=' success' ” and the provider *still has more*  
 4261 *matching updates* that have not yet been returned to the requestor, then the  
 4262 `<iterateResponse>` MUST contain exactly one `<iterator>`.
- 4263 • If the `<iterateResponse>` specifies “status=' failure' ”, then the `<iterateResponse>`  
 4264 MUST NOT contain an `<iterator>`.
- 4265 **iterator ID.** An `<iterator>` MUST have an “ID” attribute.
- 4266 The value of the “ID” attribute uniquely identifies the `<iterator>` within the namespace of the  
 4267 provider. The “ID” attribute allows the provider to map each `<iterator>` token to the result set of  
 4268 the requestor’s `<query>` and to any state that records the requestor’s position within that result set.
- 4269 The “ID” attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an  
 4270 `<iterator>`. An `<iterator>` is not a PSO.
- 4271 **Error.** If the `<iterateResponse>` specifies “status=' failure' ”, then the  
 4272 `<iterateResponse>` MUST have an “error” attribute that characterizes the failure.  
 4273 See the general section titled “[Error \(normative\)](#)”.
- 4274 In addition, the `<iterateResponse>` MUST specify an appropriate value of “error” if any of the  
 4275 following is true:
- 4276 • The provider does not recognize the `<iterator>` in an `<iterateRequest>` as representing  
 4277 an updates result set.
- 4278 • The provider does not recognize the `<iterator>` in an `<iterateRequest>` as representing  
 4279 any updates result set that the provider currently maintains.
- 4280 The `<iterateResponse>` MAY specify an appropriate value of “error” if any of the following is  
 4281 true:
- 4282 • An `<iterateRequest>` contains an `<iterator>` that is *not the most recent version* of the  
 4283 `<iterator>`. If the provider has returned to the requestor a more recent `<iterator>` that  
 4284 represents the same updates result set, then the provider MAY reject the older `<iterator>`.  
 4285 (A provider that changes the ID—for example, to encode the state of iteration within an updates  
 4286 result set—may be sensitive to this.)

4287 **3.6.9.2.3** *iterate Examples (non-normative)*

4288 In order to illustrate the iterate operation, we first need an updates operation that returns more than  
4289 one update. In the following example, a requestor asks a provider to return updates for every  
4290 `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>
```

4291 The provider returns a `<updatesResponse>`. The "status" attribute of the  
4292 `<updatesResponse>` indicates that the provider successfully executed the updates operation.  
4293 The `<updatesResponse>` contains two `<update>` elements that represent the first matching  
4294 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>
```

4295 The requestor asks the provider to return the next set of matching updates (from the original result  
4296 set). The requestor supplies the `<iterator>` from the `<updatesResponse>` as input to the  
4297 `<iterateRequest>`.

```
<iterateRequest requestID="153">  
  <iterator ID="1970"/>  
</iterateRequest>
```

4298 The provider returns an `<iterateResponse>` in response to the `<iterateRequest>`. The  
4299 "status" attribute of the `<iterateResponse>` indicates that the provider successfully executed  
4300 the iterate operation. The `<iterateResponse>` contains two `<update>` elements that represent  
4301 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>
```

4302 The `<iterateResponse>` also contains another `<iterator>` element. The "ID" of this  
4303 `<iterator>` differs from the "ID" of the `<iterator>` in the original `<updatesResponse>`. The  
4304 "ID" could remain constant (for each iteration of the result set that the `<iterator>` represents) if  
4305 the provider so chooses, but the "ID" value could change (e.g., if the provider uses "ID" to  
4306 encode the state of the result set).

4307 To get the next set of matching updates, the requestor again supplies the `<iterator>` from the  
4308 `<iterateResponse>` as input to an `<iterateRequest>`.

```
<iterateRequest requestID="154">  
  <iterator ID="1971"/>  
</iterateRequest>
```

4309 The provider again returns an `<iterateResponse>` in response to the `<iterateRequest>`. The  
4310 "status" attribute of the `<iterateResponse>` indicates that the provider successfully executed  
4311 the iterate operation. The `<iterateResponse>` contains an `<update>` element that represents  
4312 the final matching object. Since all of the matching objects have now been returned to the  
4313 requestor, this `<iterateResponse>` contains no `<iterator>`.

```
<iterateResponse requestID="154" status="success">  
  <update timestamp="20050704115900" updateKind="modify">  
    <psolID ID="2244" targetID="target2"/>  
  </update>  
</iterateResponse>
```

4314

### 4315 **3.6.9.3 closeliterator**

4316 The closeliterator operation tells the provider that the requestor has no further need for the updates  
4317 result set that a specific `<iterator>` represents. (See the description of the [updates operation](#)  
4318 above.)

4319 A requestor should send a `<closeIteratorRequest>` to the provider when the requestor no  
4320 longer intends to iterate an updates result set. (A provider will eventually free an inactive updates  
4321 result set--even if the provider never receives a `<closeIteratorRequest>` from the requestor--  
4322 but this behavior is unspecified.) For more information, see the topic named "Resource  
4323 Considerations" topic earlier within this section.

4324 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"/>

```

4325 **A closeliteratorRequest receives a closeliteratorResponse.** A requestor supplies as input to a  
4326 <closeIteratorRequest> the <iterator> that was part of the original <updatesResponse>  
4327 or the <iterator> that was part of a subsequent <iterateResponse>, whichever is most  
4328 recent. A provider returns a <closeIteratorResponse> in response to each  
4329 <closeIteratorRequest>. A <closeIteratorResponse> has the same structure as an  
4330 <spml:response>.

4331 **closeliterator is not batchable.** For reasons of scale, neither an updates request nor an iterate  
4332 request nor a closeliterator request should be nested in a [batch](#) request. When an updates query  
4333 matches more updates than the provider can place directly in the response, the provider must  
4334 temporarily store the remaining updates. Storing the remaining updates allows the requestor to  
4335 iterate the remaining updates, but also requires the provider to commit resources.  
4336 See the topic named "Resource Considerations" earlier in this section.

4337 Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the  
4338 results of asynchronous batch operations imposes on providers a resource burden similar to that of  
4339 search results. Allowing a requestor to nest an updates request or an iterate request or a  
4340 closeliterator request within a batch request would aggravate the resource problem, requiring a  
4341 provider to store more information in larger chunks for a longer amount of time.

4342 **The closeliterator operation must be executed synchronously.** The provider is already queuing  
4343 the result set (every update beyond those returned in the first updates response), so a request to  
4344 close the iterator (and thus to free the system resources associated with the result set) should be  
4345 executed as soon as possible. It is unreasonable for a requestor to ask the provider to queue the  
4346 results of a request to close an iterator (especially since the close iterator response contains little or  
4347 no information beyond success or failure).



### 4348 3.6.9.3.1 *closeIteratorRequest (normative)*

4349 A requestor SHOULD send a `<closeIteratorRequest>` to a provider when the requestor no  
4350 longer intends to iterate an updates result set. (This allows the provider to free any system  
4351 resources associated with the updates result set.)

4352 **Execution.** A `<closeIteratorRequest>` MUST NOT specify

4353 "executionMode='asynchronous'".

4354 A `<closeIteratorRequest>` MUST specify "executionMode='synchronous' "

4355 or (a `<closeIteratorRequest>` MUST) omit "executionMode".

4356 See the section titled "[Determining execution mode](#)".

4357 **iterator.** A `<closeIteratorRequest>` MUST contain exactly one `<iterator>` element. A  
4358 requestor MUST supply as input to a `<closeIteratorRequest>` the `<iterator>` from the  
4359 original `<updatesResponse>` or (a requestor MUST supply the `<iterator>`) from a subsequent  
4360 `<iterateResponse>`. A requestor SHOULD supply as input to a  
4361 `<closeIteratorRequest>` the most recent `<iterator>` that represents the updates result set.

4362 **iterator ID.** An `<iterator>` that is part of a `<closeIteratorRequest>` MUST have an "ID"  
4363 attribute. (The value of the "ID" attribute uniquely identifies the `<iterator>` within the  
4364 namespace of the provider. The "ID" attribute allows the provider to map each `<iterator>`  
4365 token to the result set of the requestor's `<query>` and also (allows the provider to map each  
4366 `<iterator>` token) to any state that records the requestor's iteration *within* that result set.)

### 4367 3.6.9.3.2 *closeIteratorResponse (normative)*

4368 A provider that receives a `<closeIteratorRequest>` from a requestor that the provider trusts  
4369 must examine the content of the `<closeIteratorRequest>`. If the request is valid, the provider  
4370 MUST release any updates result set that the `<iterator>` represents. Any subsequent request to  
4371 iterate that same updates result set MUST fail.

4372 **Execution.** The provider MUST execute the `closeIterator` operation synchronously (if the provider  
4373 executes the `closeIterator` operation at all). See the section titled "[Determining execution mode](#)".

4374 **Response.** The provider MUST return to the requestor a `<closeIteratorResponse>`.

4375 **Status.** The `<closeIteratorResponse>` must contain a "status" attribute that indicates  
4376 whether the provider successfully released the updates result set that the `<iterator>` represents.  
4377 See the section titled "[Status \(normative\)](#)".

4378 • If the provider successfully released the updates result set that the `<iterator>` represents,  
4379 then the `<closeIteratorResponse>` MUST specify "status='success'".

4380 • If the provider encountered an error in releasing the updates result set that the `<iterator>`  
4381 represents, then the `<closeIteratorResponse>` MUST specify "status='failure'".

4382 **Error.** If the `<closeIteratorResponse>` specifies "status='failure' ", then the  
4383 `<closeIteratorResponse>` MUST have an "error" attribute that characterizes the failure.  
4384 See the general section titled "[Error \(normative\)](#)".

4385 In addition, the `<closeIteratorResponse>` MUST specify an appropriate value of "error" if  
4386 any of the following is true:

4387 • If the provider does not recognize the `<iterator>` in a `<closeIteratorRequest>` as  
4388 representing an updates result set.

- 4389 • If the provider does not recognize the <iterator> in a <closeIteratorRequest> as  
4390 representing any updates result set that the provider currently maintains.
- 4391 • If the provider recognized the <iterator> in a <closeIteratorRequest> as representing  
4392 a updates result set that the provider currently maintains but *cannot release the resources*  
4393 *associated with that updates result set*.

4394 The <closeIteratorResponse> MAY specify an appropriate value of "error" if any of the  
4395 following is true:

- 4396 • If a <closeIteratorRequest> contains an <iterator> that is *not the most recent version*  
4397 *of the <iterator>*. If the provider has returned to the requestor a more recent <iterator>  
4398 that represents the same updates result set, then the provider MAY reject the older  
4399 <iterator>.  
4400 (A provider that changes the ID—for example, to encode the state of iteration within a updates  
4401 result set—may be sensitive to this.)

### 4402 3.6.9.3.3 *closeIterator Examples (non-normative)*

4403 In order to illustrate the iterate operation, we first need an updates operation that returns more than  
4404 one update. In the following example, a requestor asks a provider to return updates for every  
4405 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>
```

4406 The provider returns a <updatesResponse>. The "status" attribute of the  
4407 <updatesResponse> indicates that the provider successfully executed the updates operation.  
4408 The <updatesResponse> contains two <update> elements that represent the first matching  
4409 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>
```

4410 The requestor decides that the two objects in the initial <searchResponse> will suffice, and does  
4411 not intend to retrieve any more matching objects (in the result set for the search). The requestor  
4412 supplies the <iterator> from the <updatesResponse> as input to the  
4413 <closeIteratorRequest>.

```
<closeIteratorRequest requestID="153">  
  <iterator ID="1900"/>  
</closeIteratorRequest>
```

- 4414 The provider returns a `<closeIteratorResponse>` in response to the  
4415 `<closeIteratorRequest>`. The "status" attribute of the `<closeIteratorResponse>`  
4416 indicates that the provider successfully released the result set.

```
<closeIteratorResponse requestID="153" status="success"/>
```

4417

### 4418 **3.7 Custom Capabilities**

4419 The features of SPMLv2 that allow the PSTC to define optional operations as part of standard  
4420 capabilities are *open mechanisms* that will work for anyone. An individual provider (or any third  
4421 party) can define a custom capability that integrates with SPMLv2. Whoever controls the  
4422 namespace of the capability controls the extent to which it can be shared. Each provider  
4423 determines which capabilities are supported for which types of objects on which types of targets.

4424 The SPMLv2 capability mechanism is extensible. Any party may define additional capabilities. A  
4425 provider declares its support for a custom capability in exactly the same way that it declares support  
4426 for a standard capability: as a target <capability> element.

4427 The standard capabilities that SPMLv2 defines will not address all needs. Contributors may define  
4428 additional custom capabilities.

4429 Since the schema for each capability is defined in a separate namespace, a custom capability will  
4430 not ordinarily conflict with a standard capability that is defined as part of SPMLv2, nor will a custom  
4431 capability ordinarily conflict with another custom capability. In order for a custom capability B to  
4432 conflict with another capability A, capability B would have to import the namespace of capability A  
4433 and re-declare a schema element from capability A. Such a conflict is clearly intentional and a  
4434 provider can easily avoid such a conflict by not declaring support for capability B.

4435 Also see the section titled "[Conformance](#)".

4436

---

## 4 Conformance (normative)

4437

### 4.1 Core operations and schema are mandatory

4438  
4439

A conformant provider **MUST** support the elements, attributes, and types defined in the SPMLv2 Core XSD. This includes all the core operations and protocol behavior.

4440  
4441  
4442

Schema syntax for the SPMLv2 core operations is defined in a schema that is associated with the following XML namespace: `urn:oasis:names:tc:SPML:2:0`. This document includes the Core XSD as Appendix A.

4443

### 4.2 Standard capabilities are optional

4444  
4445

A conformant provider **SHOULD** support the XML schema and operations defined by each standard capability of SPMLv2.

4446

### 4.3 Custom capabilities must not conflict

4447  
4448

A conformant provider **MUST** use the custom capability mechanism of SPMLv2 to expose any operation beyond those specified by the core and standard capabilities of SPMLv2.

4449

A conformant provider **MAY** support any custom capability that conforms to SPMLv2.

4450  
4451

**Must conform to standard schema.** Any operation that a custom capability defines **MUST** be defined as a request-response pair such that all of the following are true:

4452  
4453  
4454

- The request type (directly or indirectly) extends `{RequestType}`
- The response type is `{ResponseType}` or (the response type directly or indirectly) extends `{ResponseType}`.

4455  
4456  
4457  
4458

**Must not conflict with another capability.** Since each custom capability is defined in its own namespace, an element or attribute in the XML schema that is associated with a *custom capability* **SHOULD NOT conflict with** (i.e., **SHOULD NOT** redefine and **SHOULD NOT** otherwise change the definition of) any element or attribute in any other namespace:

4459  
4460  
4461

- A custom capability **MUST NOT** conflict with the Core XSD of SPMLv2.
- A custom capability **MUST NOT** conflict with any standard capability of SPMLv2.
- A custom capability **SHOULD NOT** conflict with another custom capability.

4462  
4463  
4464

**Must not bypass standard capability.** A conformant provider **MUST NOT** expose an operation that competes with (i.e., whose functions overlap those of) an operation defined by a standard capability of SPMLv2) **UNLESS** all of the following are true:

4465  
4466  
4467

- The provider **MUST** *define the competing operation in a custom capability*.
- Every target (and every schema entity on a target) that supports the provider's custom capability **MUST** also *support the standard capability*.

4468 **4.4 Capability Support is all-or-nothing**

4469 A provider that claims to support a particular capability for (a set of schema entities on) a target  
4470 MUST support (for every instance of those schema entities on the target) every operation that the  
4471 capability defines.

4472 **4.5 Capability-specific data**

4473 A capability MAY imply capability-specific data. That is, a capability MAY specify that data specific  
4474 to that capability may be associated with one or more objects. (For example, the Reference  
4475 Capability implies that each object may contain a set of references to other objects.)

4476 Any capability that implies capability-specific data MAY rely on the default processing that SPMLv2  
4477 specifies for capability-specific data (see the section titled “CapabilityData Processing (normative”).  
4478 However, any capability that implies capability-specific data SHOULD specify the structure of that  
4479 data. (For example, the Reference Capability specifies that its capability-specific data must contain  
4480 at least one <reference> and should contain only <reference> elements.)

4481 Furthermore, any capability that implies capability-specific data and *for which the default processing*  
4482 *of capability-specific data is inappropriate* (i.e., any capability for which an instance of  
4483 {CapabilityDataType} that refers to the capability would specify “mustUnderstand=’true’”)

- 4484 • MUST specify the structure of that capability-specific data.
- 4485 • MUST specify how core operations should handle that capabilityData.  
4486 (For example, the Reference Capability specifies how each reference must be validated and  
4487 processed. See the section titled “[Reference CapabilityData Processing \(normative\)](#).”)

---

4488

## 5 Security Considerations

4489

### 5.1 Use of SSL 3.0 or TLS 1.0

4490 When using Simple Object Access Protocol (SOAP) **[SOAP]** as the protocol for the [requestor](#)  
4491 (client) to make SPMLv2 requests to a [provider](#) (server), Secure Sockets Layer (SSL 3.0) or  
4492 Transport Layer Security (TLS 1.0) **[RFC 2246]** SHOULD be used.

4493 The TLS implementation SHOULD implement the TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA or the  
4494 TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA **[AES]** cipher suite.

4495

### 5.2 Authentication

4496 When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as  
4497 the SOAP **[SOAP]** transport protocol, the [provider](#) (server) SHOULD be authenticated to the  
4498 [requestor](#) (client) using X.509 v3 **[X509]** service certificates. The [requestor](#) (client) SHOULD be  
4499 authenticated to the [provider](#) (server) using X.509 v3 service certificates.

4500 For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require  
4501 intermediaries, Web Services Security **[WSS]** SHOULD be used for authentication.

4502

### 5.3 Message Integrity

4503 When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as  
4504 the SOAP **[SOAP]** transport protocol, message integrity is reasonably assured for point-to-point  
4505 message exchanges.

4506 For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require  
4507 intermediaries, Web Services Security **[WSS]** SHOULD be used to ensure message integrity.

4508

### 5.4 Message Confidentiality

4509 When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as  
4510 the SOAP **[SOAP]** transport protocol, message confidentiality is reasonably assured for point-to-  
4511 point message exchanges, and for the entire message.

4512 For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require  
4513 intermediaries, Web Services Security **[WSS]** SHOULD be used to ensure confidentiality for the  
4514 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>
    <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>

```

```

        </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="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 B. 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>
```

4518

## Appendix C. 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>
```

4520

## Appendix D. 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"/>
          <attribute name="recursive" type="boolean" use="optional"/>
        </sequence>
      </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>
```

4522

## Appendix E. 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 F. 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>

```

4526

## Appendix G. 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 H. 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 I. 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 J. Document References

4531

- 4532     **[AES]**                     National Institute of Standards and Technology (NIST), FIPS-197:  
4533     Advanced Encryption Standard,  
4534     <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>,  
4535     National Institute of Standards and Technology (NIST)
- 4536     **[ARCHIVE-1]**             OASIS Provisioning Services Technical Committee, email archive,  
4537     [http://www.oasis-  
4538     open.org/apps/org/workgroup/provision/email/archives/index.  
4539     html](http://www.oasis-open.org/apps/org/workgroup/provision/email/archives/index.html), OASIS PS-TC
- 4540     **[DS]**                     IETF/W3C, *W3C XML Signatures*, <http://www.w3.org/Signature/>,  
4541     W3C/IETF
- 4542     **[DSML]**                   OASIS Directory Services Markup Standard, *DSML V2.0  
4543     Specification*, [http://www.oasis-  
4544     open.org/specs/index.php#dsmlv2](http://www.oasis-open.org/specs/index.php#dsmlv2), OASIS DSML Standard
- 4545     **[GLOSSARY]**             OASIS Provisioning Services TC, *Glossary of Terms*,  
4546     [http://www.oasis-  
4547     open.org/apps/org/workgroup/provision/download.php](http://www.oasis-open.org/apps/org/workgroup/provision/download.php), OASIS  
4548     PS-TC
- 4549     **[RFC 2119]**             S. Bradner., *Key words for use in RFCs to Indicate Requirement  
4550     Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF
- 4551     **[RFC 2246]**             T. Dierks and C. Allen, *The TLS Protocol*,  
4552     <http://www.ietf.org/rfc/rfc2246.txt>, IETF
- 4553     **[SAML]**                    OASIS Security Services TC, [http://www.oasis-  
4554     open.org/committees/tc\\_home.php?wg\\_abbrev=security](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security),  
4555     OASIS SS-TC
- 4556     **[SOAP]**                    W3C XML Protocol Working Group,  
4557     <http://www.w3.org/2000/xp/Group/>
- 4558     **[SPML-Bind]**             OASIS Provisioning Services TC, *SPML V1.0 Protocol Bindings*,  
4559     [http://www.oasis-  
4560     open.org/apps/org/workgroup/provision/download.php/1816/d  
4561     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
- 4562     **[SPML-REQ]**             OASIS Provisioning Services Technical Committee, *Requirements*,  
4563     [http://www.oasis-  
4564     open.org/apps/org/workgroup/provision/download.php/2277/d  
4565     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
- 4566     **[SPML-UC]**             OASIS Provisioning Services Technical Committee, *SPML V1.0  
4567     Use Cases*, [http://www.oasis-  
4568     open.org/apps/org/workgroup/provision/download.php/988/drf  
4569     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
- 4570     **[SPMLv2-Profile-DSML]**   OASIS Provisioning Services Technical Committee, *SPMLv2  
4571     DSMLv2 Profile*, OASIS PS-TC
- 4572     **[SPMLv2-Profile-XSD]**   OASIS Provisioning Services Technical Committee, *SPML V2 XSD  
4573     Profile*, OASIS PS-TC

|                      |                         |   |
|----------------------|-------------------------|---|
| 4574<br>4575         | <b>[SPMLv2-REQ]</b>     | OASIS Provisioning Services Technical Committee, Requirements, OASIS PS-TC  |
| 4576<br>4577         | <b>[SPMLv2-ASYNC]</b>   | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Async Capability of SPMLv2, OASIS PS-TC   |
| 4578<br>4579         | <b>[SPMLv2-BATCH]</b>   | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Batch Capability of SPMLv2, OASIS PS-TC   |
| 4580<br>4581         | <b>[SPMLv2-BULK]</b>    | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Bulk Capability of SPMLv2, OASIS PS-TC  |
| 4582<br>4583         | <b>[SPMLv2-CORE]</b>    | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Core Operations of SPMLv2, OASIS PS-TC  |
| 4584<br>4585         | <b>[SPMLv2-PASS]</b>    | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Password Capability of SPMLv2, OASIS PS-TC  |
| 4586<br>4587         | <b>[SPMLv2-REF]</b>     | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Reference Capability of SPMLv2, OASIS PS-TC   |
| 4588<br>4589         | <b>[SPMLv2-SEARCH]</b>  | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Search Capability of SPMLv2, OASIS PS-TC  |
| 4590<br>4591         | <b>[SPMLv2-SUSPEND]</b> | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Suspend Capability of SPMLv2, OASIS PS-TC   |
| 4592<br>4593         | <b>[SPMLv2-UPDATES]</b> | OASIS Provisioning Services Technical Committee, XML Schema Definitions for Updates Capability of SPMLv2, OASIS PS-TC   |
| 4594<br>4595         | <b>[SPMLv2-UC]</b>      | OASIS Provisioning Services Technical Committee., SPML V2.0 Use Cases, OASIS PS-TC  |
| 4596<br>4597<br>4598 | <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 |
| 4599<br>4600         | <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>                          |
| 4601<br>4602         | <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  |
| 4603                 |                         |   |

---

4604 **Appendix K. Acknowledgments**

4605 The following individuals were voting members of the Provisioning Services committee at the time  
4606 that this version of the specification was issued:

4607           Jeff Bohren, BMC  
4608           Robert Boucher, CA  
4609           Gary Cole, Sun Microsystems  
4610           Rami Elron, BMC  
4611           Marco Fanti, Thor Technologies  
4612           James Hu, HP  
4613           Martin Raeppe, SAP  
4614           Gavenraj Sodhi, CA  
4615           Kent Spaulding, Sun Microsystems  
4616

4617

---

## Appendix L. Notices

4618 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
4619 that might be claimed to pertain to the implementation or use of the technology described in this  
4620 document or the extent to which any license under such rights might or might not be available;  
4621 neither does it represent that it has made any effort to identify any such rights. Information on  
4622 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
4623 website. Copies of claims of rights made available for publication and any assurances of licenses to  
4624 be made available, or the result of an attempt made to obtain a general license or permission for  
4625 the use of such proprietary rights by implementers or users of this specification, can be obtained  
4626 from the OASIS Executive Director.

4627 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
4628 applications, or other proprietary rights which may cover technology that may be required to  
4629 implement this specification. Please address the information to the OASIS Executive Director.

4630 **Copyright © OASIS Open 2006. All Rights Reserved.**

4631 This document and translations of it may be copied and furnished to others, and derivative works  
4632 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
4633 published and distributed, in whole or in part, without restriction of any kind, provided that the above  
4634 copyright notice and this paragraph are included on all such copies and derivative works. However,  
4635 this document itself may not be modified in any way, such as by removing the copyright notice or  
4636 references to OASIS, except as needed for the purpose of developing OASIS specifications, in  
4637 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights  
4638 document must be followed, or as required to translate it into languages other than English.

4639 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
4640 successors or assigns.

4641 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
4642 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
4643 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY  
4644 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
4645 PARTICULAR PURPOSE

4646