OASIS Service Provisioning Markup Language (SPML) Version 2

OASIS Standard
2006 April 1

Document identifier: pstc-spml2-os.pdf

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

Send comments to: pstc-comment@lists.oasis-open.org

Editor:

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

Contributors:

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

Abstract:

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

Status:

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

If you are on the provision list for committee members, send comments there. If you are not on that list, subscribe to the provision-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to provision-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

Table of contents

1 Introduction

1.1 Purpose

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

1.2 Organization

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

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

  Each section that describes an operation includes:
  - The relevant XML Schema
  - A *normative* subsection that describes the *request* for the operation
  - A *normative* subsection that describes the *response* to the operation
  - A *non-normative* sub-section that discusses *examples* of the operation
- The Conformance section describes the aspects of this protocol that a requestor or provider must support in order to be considered conformant.
- A Security and Privacy Considerations section describes risks that an implementer of this protocol should weigh in deciding how to deploy this protocol in a specific environment.

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

1.3 Audience

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

One group of readers will want to know: ***"What is SPML?"***
A reader of this type should pay special attention to the Concepts section.

A second group of readers will want to know: ***"How would I use SPML?"***
A reader of this type should read the Protocol section
(with special attention to the *examples*).

A third group of readers will want to know: ***"How must I implement SPML?"***
A reader of this type must read the Protocol section
(with special attention to normative *request* and *response* sub-sections).

A reader who is already familiar with SPML 1.0 will want to know: ***"What is new in SPMLv2?"***
A reader of this type should read the Concepts section thoroughly.

1.4 Notation

1.4.1 Normative sections

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

1.4.2 Normative terms

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

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

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

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

1.4.3 Typographical conventions

This specification uses the following typographical conventions in text:

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

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

```
Listings of SPML schemas appear like this.
```

Example code listings appear like this.

1.4.4 Namespaces

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

- The prefix `dsml:` stands for the Directory Services Markup Language namespace **[DSML]**.
- The prefix `xsd:` stands for the W3C XML Schema namespace [**XSD**].
- The prefix `spml:` stands for the SPMLv2 Core XSD namespace **[SPMLv2-CORE]**.
- The prefix `spmlasync:` stands for the SPMLv2 Async Capability XSD namespace. **[SPMLv2-ASYNC]**.
- The prefix `spmlbatch:` stands for the SPMLv2 Batch Capability XSD namespace **[SPMLv2-BATCH]**.
- The prefix `spmlbulk:` stands for the SPMLv2 Bulk Capability XSD namespace **[SPMLv2-BULK]**.
- The prefix `spmlpass:` stands for the SPMLv2 Password Capability XSD namespace **[SPMLv2-PASS]**.
- The prefix `spmlref:` stands for the SPMLv2 Reference Capability XSD namespace **[SPMLv2-REF]**.
- The prefix `spmlsearch:` stands for the SPMLv2 Search Capability XSD namespace **[SPMLv2-SEARCH]**.
- The prefix `spmlsuspend:` stands for the SPMLv2 Suspend Capability XSD namespace **[SPMLv2-SUSPEND]**.
- The prefix `spmlupdates:` stands for the SPMLv2 Updates Capability XSD namespace **[SPMLv2-UPDATES]**.

2 Concepts

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

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.

2.1 Domain Model

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.

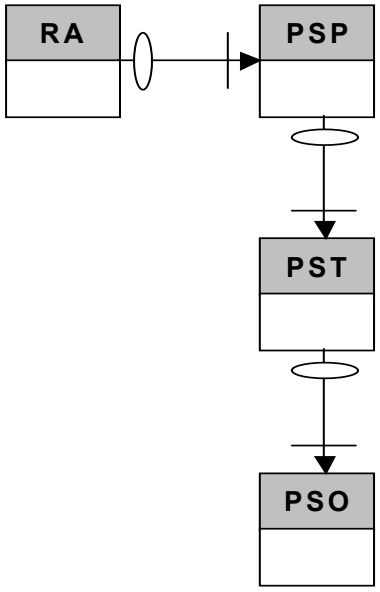


Figure 1. Domain model elements

2.1.1 Requestor

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:

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

Trust relationship. In an end-to-end integrated provisioning scenario, any component that issues an SPML request is said to be operating as a requestor. This description assumes that the

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

2.1.2 Provider

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

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

2.1.3 Target

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

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

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

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

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

- A requestor can discover from the provider
- No requestor can add, modify, delete or otherwise act upon
- May contain any number of provisioning service objects (PSO) upon which a requestor may act
- May contain a schema that defines the XML structure of the provisioning service objects (PSO) that the target may contain
- May define which schema entities the target supports
- May expose capabilities:
  - That apply to every supported schema entity
  - That apply only to specific schema entities

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

- A provider (PSP) must uniquely identify each target that it exposes.
- A provider must uniquely identify each object (PSO) that a target contains.
- Exactly one target must contain each object (PSO) that the provider manages.

2.1.3.1 Target Schema

The schema for each target defines the XML structure of the objects (PSO) that the target may contain.

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

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

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

2.1.3.2 Supported Schema Entities

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

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

2.1.3.3 Capabilities

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

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

- The OASIS *PSTC defines each standard capability* in an SPML namespace. See the section titled “Namespaces”.
- *Anyone may define a custom capability* in another namespace.

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

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

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

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

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

2.1.4 Provisioning Service Object (PSO)

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

NOTE: Within this document, the term "object" (unless otherwise qualified) refers to a PSO.

Every object is contained by exactly one target. Each object has a unique identifier (PSO-ID).

2.2 Core Protocol

SPMLv2 retains the SPML1.0 concept of a "core protocol". The SPMLv2 Core XSD defines:

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

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

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

Conformant SPMLv2 implementations must support the core protocol, including:

- The new listTargets operation
- The basic operations for every schema entity that a target supports
- The modal mechanisms for asynchronous operations

(For more information, see the section titled "Conformance").

2.3 Profile

SPMLv2 defines two "profiles" in which a requestor and provider may exchange SPML protocol:

- XML Schema as defined in the "SPMLv2 XSD Profile" **[SPMLv2-Profile-XSD]**.
- DSMLv2 as defined in the "SPMLv2 DSMLv2 Profile" **[SPMLv2-Profile-DSML]**.

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

SPML 1.0 defined file bindings and SOAP bindings that assumed the SPML1.0 Schema for DSML **[SPML-Bind]**. The SPMLv2 DSMLv2 Profile provides *a degree of backward compatibility* with SPML 1.0. The DSMLv2 profile supports a schema model similar to that of SPML 1.0.

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

3 Protocol

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

Core Operations. In order to encourage adoption of this standard, this specification minimizes the set of operations that a provider must implement. The Core Operations section discusses these *required operations*.

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

Custom Capabilities. The capability mechanism in SPMLv2 is *open* and allows an individual provider (or any third party) to define additional *custom capabilities*. See the sub-section titled "Custom Capabilities".

3.1 Request/Response Model

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

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

- the exchange of requests and responses between requestor and provider
- synchronous and asynchronous execution of operations
- individual and batch requests

3.1.1 Conversational flow

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

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

3.1.2 Status and Error codes

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

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

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

- A status request asks for the status of another operation that the provider is *already executing asynchronously*. See the section titled "Synchronous and asynchronous operations" below. A status response has status and error attributes that tell the requestor what happened to the status request itself. However, the response to a successful status operation also contains a *nested response* that tells what has happened to the operation that the provider is executing asynchronously.

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

3.1.2.1 Status (normative)

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

- A response that specifies "status='success'" indicates that the provider has completed the requested operation. In this case, the response contains any result of the operation and the response MUST NOT specify "error" (see below).

- A response that specifies "status='failure'" indicates that the provider could not complete the requested operation. In this case, the response MUST specify an appropriate value of "error" (see below).

- A response that specifies "status='pending'" indicates that the provider will execute the requested operation asynchronously (see "Synchronous and asynchronous operations" below). In this case, the response acknowledges the request and contains the "requestID" value that identifies the asynchronous operation.

3.1.2.2 Error (normative)

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

- A response that specifies "error='malformedRequest'" indicates that the provider could not interpret the request. This includes, but is not limited to, parse errors.

- A response that specifies "error='unsupportedOperation'" indicates that the provider does not support the operation that the request specified.

- A response that specifies "error='unsupportedIdentifierType'" indicates that the provider does not support the type of identifier specified in the request.

- A response that specifies "error='noSuchIdentifier'" indicates that the provider (supports the type of identifier specified in the request, but the provider) cannot find the object to which an identifier refers.

- A response that specifies "error='unsupportedExecutionMode'" indicates that the provider does not support the requested mode of execution.

- A response that specifies "error='invalidContainment'" indicates that the provider cannot add the specified object to the specified container.

  - The request may have specified as container an object that *does not exist*.

  - The request may have specified as container an object that *is not a valid container*. The target schema implicitly or explicitly declares each supported schema entity. An explicit declaration of a supported schema entity specifies 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).

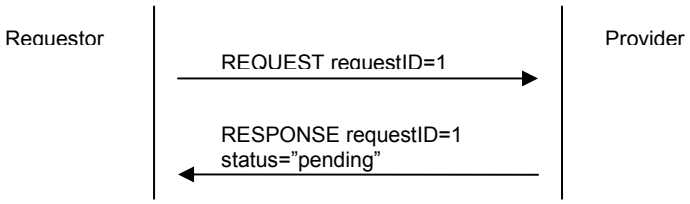
3.1.3 Synchronous and asynchronous operations

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

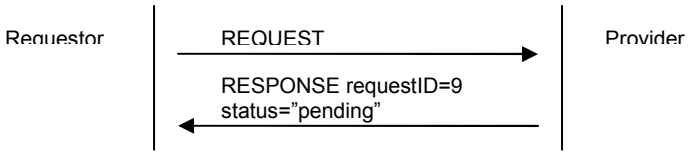
- **Synchronous: operation before response**. If a provider executes a requested operation *synchronously*, the provider completes the requested operation before the provider returns a response to the requestor. The response will include the status and any error or result.

- **Asynchronous: response before operation**. If a provider executes a requested operation *asynchronously*, the provider returns to the requestor a response (that indicates that the operation will be executed asynchronously) before the provider executes the requested operation. The response will specify "`status='pending'`" and will specify a "`requestID`" value that the requestor must use in order to cancel the asynchronous operation or (in order to) obtain the status or results of the asynchronous operation.

  - If a request *specifies* "`requestID`", then the provider's response to that request will specify the *same* "`requestID`" value.



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



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

3.1.3.1 ExecutionMode attribute

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

3.1.3.2 Async Capability

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

SPMLv2's Async Capability also defines two operations that a requestor may use to manage other operations that a provider is executing asynchronously:

- A status operation allows a requestor to check the status (and optionally results) of an operation (or of all operations)
- A cancel operation asks the provider to stop executing an operation.

For more information, see the section titled "Async Capability".

3.1.3.3 Determining execution mode

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

- Requestor specifies synchronous execution
- Requestor specifies asynchronous execution
- Provider chooses synchronous execution
- Provider chooses asynchronous execution

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

3.1.3.3.1 Requestor specifies synchronous execution (normative)

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

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

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

- If the provider's response specifies "`status='failed'`", then the provider's response must have an "`error`" attribute.

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

3.1.3.3.2 Requestor specifies asynchronous execution (normative)

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

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

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

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

- If the request specifies a "`requestID`" value, then the provider's response MUST specify the same "`requestID`" value.
- If the request omits "`requestID`", then the provider's response MUST specify a "`requestID`" value that uniquely identifies the requested operation within the namespace of the provider.

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

See also the sections titled "Async Capability" and "Results of asynchronous operations (normative)".

3.1.3.3.3 Provider chooses synchronous execution (normative)

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

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

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

3.1.3.3.4 Provider chooses asynchronous execution (normative)

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

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

- If the request specifies a "requestID" value, then the provider's response MUST specify the same "requestID" value.
- If the request omits "requestID", then the provider's response MUST specify a "requestID" value that uniquely identifies the requested operation within the namespace of the provider.

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

See also the sections titled "Async Capability" and "Results of asynchronous operations (normative)".

3.1.3.4 Results of asynchronous operations (normative)

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

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

3.1.4 Individual and batch requests

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

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

Batch. SPMLv2 defines batch as an optional operation that allows a requestor to combine any number of requests into a single request. See the section titled "Batch Capability".

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>
```

SPMLv2 uses several different types of identifiers.

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

3.2.1 Request Identifier (normative)

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

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

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

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

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

3.2.2 Target Identifier (normative)

Each of a provider's targets has a string identifier. Within a provider's listTargets response, the `"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}.

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

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

3.3 Selection

3.3.1 QueryClauseType

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

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

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

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

3.3.2 Logical Operators

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

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

3.3.3 SelectionType

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

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

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

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

3.3.3.1 SelectionType in a Request (normative)

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

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

- Within a <modification> element, the value of the "path" attribute MUST specify a *target* schema entity (i.e., an element or attribute) of the object that the provider is to modify.

- Within a <query> element, the value of the "path" attribute MUST specify a *filter* that selects objects based on:
  - The presence (or absence) of a specific element or attribute
  - The presence (or absence) of a specific value in the content of an element or (the presence of absence of a specific value) in the value of an attribute

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

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

- Each <namespacePrefixMap> MUST have a "prefix" attribute whose value specifies a namespace prefix (that may occur in the filter expression that is the value of the "path" attribute).

- Each <namespacePrefixMap> MUST have a "namespace" attribute whose value is the URI for an XML namespace.

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

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

Refer to the documentation of each profile for normative specifics.

3.3.3.2 SelectionType Processing (normative)

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

- Within a <modification> element, a provider must resolve the value of the "path" attribute to a schema entity (i.e., to an element or attribute) of the object that the provider is to modify.

- Within a <query> element, a provider must evaluate the value of the "path" attribute as a filter expression that selects objects based on:
  - The presence (or absence) of a specific element or attribute
  - The presence (or absence) of a specific value in the content of an element or (the presence of absence of a specific value) in the value of an attribute

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

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

object.)
See the section titled “SearchQueryType Errors (normative)” below.

- A requestor that wants the provider to search *only direct descendants* of the target or (that wants to search only direct descendants) of the object specified by `<basePsoID>` MUST specify `“scope=’oneLevel’”`.

- A requestor that wants the provider to search *any direct or indirect descendant* of the target or (that wants to search any direct or indirect descendant) of the object specified by `<basePsoID>` MUST specify “`scope=’subTree’`”.

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

- Any capability may define elements of (a type that extends) `{QueryClauseType}`. These elements allow a requestor to select objects based on capability-defined data.
See the section titled "QueryClauseType" above.

- A `<select>` element is an instance of `{SelectionType}`, which extends `{QueryClauseType}` to filter objects based on schema-defined content.
See the section titled “SelectionType in a Request (normative)“.

- Logical Operators such as `<and>`, `<or>` and `<not>` combine individual selection criteria. A logical operator MUST contain at least one instance of a type that extends `{QueryClauseType}` or a (logical operator MUST contain at least one) logical operator.
See the section titled "Logical Operators" above.

3.3.4.2 SearchQueryType Errors (normative)

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

- The `<query>` in a **<**`searchRequest`**>** specifies `“scope=’pso’”` but does not contain `<basePsoID>`. (The target itself is not a PSO.)

- The `"targetID"` of the instance of `{SearchQueryType}` does not specify a valid target.

- An instance of `{SearchQueryType}` specifies `"targetID"` and (the instance of `{SearchQueryType}` also) contains `<basePsoID>`, but the value of `"targetID"` in the instance of `{SearchQueryType}` does not match the value of `"targetID"` in the `<basePsoID>`.

- An instance of `{SearchQueryType}` contains a `<basePsoID>` that does not identify an object that exists on a target.
(That is, the `<basePsoID>` does not match the `<psoID>` of any object that exists on a target.)

- The provider cannot evaluate an instance of `{QueryClauseType}` that the instance of `{SearchQueryType}` contains.

- The open content of the instance of `{SearchQueryType}` is too complex for the provider to evaluate.

- The open content of the instance of `{SearchQueryType}` contains a syntactic error (such as an invalid structure of logical operators or query clauses).

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

Also see the section titled "SelectionType Errors (normative)".

3.4 CapabilityData

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

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

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

3.4.1 CapabilityDataType

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

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

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

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

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

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

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

Capability-specific processing. Any capability that implies capability-specific data may specify how operations should handle the data specific to that capability. Capability-specific handling takes precedence over the default handling.
See the section titled "CapabilityData Processing (normative)".

3.4.1.1 CapabilityData in a Request (normative)

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

For normative specifics of supported capabilities,
see the section titled "listTargetsResponse (normative)".

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

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

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

- A requestor that wants the request to *fail if the provider cannot provide capability-specific handling* for the set of data specific to a certain capability MUST specify "mustUnderstand='true'" on the instance of {CapabilityDataType} that contains the data specific to that capability.

- A requestor that will *accept* d*efault handling* for any data specific to a certain capability MUST specify "mustUnderstand='false'" on the instance of {CapabilityDataType} that contains the data specific to that capability or (the requestor MUST) omit the "mustUnderstand" attribute (from the instance of {CapabilityDataType} that contains the data specific to that capability).

The section titled "CapabilityData Processing (normative)" describes the default handling for capability-specific data. Any capability for which the default handling is inappropriate MUST specify how operations should handle data specific to that capability. The section titled "Reference CapabilityData Processing (normative)" specifies handling of data specific to the Reference Capability.

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

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

3.4.1.2 CapabilityData Processing (normative)

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

For normative specifics of supported capabilities, see the section titled "listTargetsResponse (normative)".

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

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

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

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

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

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

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

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

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

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

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

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

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

and that specifies the same "capabilityURI".

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

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

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

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

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

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

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

3.4.1.3 CapabilityData Errors (normative)

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

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

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

- The provider does not recognize the value of the "capabilityURI" (that the instance of {CapabilityDataType} specifies) as identifying a capability that the provider supports *for the target* that contains the object to be manipulated *and* that the provider supports *for the schema entity* of which the object to be manipulated is an instance.

  Alternatively, the provider MAY perform the default handling as described above in the section titled "CapabilityData Processing (normative)".

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

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

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

3.4.1.4 CapabilityData in a Response (normative)

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

schema entity of which the object to be manipulated is an instance.
See the section titled "listTargetsResponse (normative)".

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

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

mustUnderstand. An instance of `{CapabilityDataType}` within a response MAY specify `"mustUnderstand"`. A provider SHOULD preserve any `"mustUnderstand"` attribute of an instance of `{CapabilityDataType}`. See the discussions of the "mustUnderstand" attribute within the sections titled "CapabilityData in a Request (normative)" and "CapabilityData Processing (normative)" above.

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

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

3.5 Transactional Semantics

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

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

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

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

3.6 Operations

The first subsection discusses the required Core Operations.

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

- Async Capability
- Batch Capability
- Bulk Capability
- Password Capability
- Reference Capability
- Search Capability
- Suspend Capability
- Updates Capability

3.6.1 Core Operations

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

A conformant provider must implement all the operations defined in the Core XSD. For more information, see the section entititled "Conformance".

The SPMLv2 core operations include:

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

3.6.1.1 listTargets

The listTargets operation enables a requestor to determine the set of targets that a provider makes available for provisioning and (the listTargets operation also enables a requestor) to determine the 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"/>
```

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

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

3.6.1.1.1 listTargetsRequest (normative)

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

Execution. A `<listTargetsRequest>` MUST NOT specify `"executionMode='asynchronous'"`. A `<listTargetsRequest>` MUST specify `"executionMode='synchronous'"` or (a `<listTargetsRequest>` MUST) omit `"executionMode"`.

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

For more information, see the section titled "Determining execution mode".

Profile. a `<listTargetsRequest>` MAY specify `"profile"`.
Any profile value MUST be a URI (e.g., of an XML namespace) that identifies an SPML profile.

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

3.6.1.1.2 listTargetsResponse (normative)

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

- If a `<listTargetsRequest>` does not specify a `"profile"`, then the `<listTargetsResponse>` MUST contain every instance of `<target>` that the provider exposes for provisioning operations *regardless of the profile* or profiles for (which the provider supports) that target.

- If a `<listTargetsRequest>` specifies a `"profile"` that the provider supports, then the `<listTargetsResponse>` MUST contain only instances of `<target>` for which the provider supports the specified profile.

- If a `<listTargetsRequest>` specifies a `"profile"` that the provider *does not support*, then the `<listTargetsResponse>` MUST specify `"status='failure'"`.
  See the topic named "Error" below within this section.

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

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

For more information, see the section titled "Determining execution mode".

Status. A `<listTargetsResponse>` MUST have a `"status"` attribute that indicates whether the provider successfully processed the request. See the section titled "Status (normative)".

Error. If the provider cannot return a list of its targets, then the `<listTargetsResponse>` MUST contain an `error` attribute that characterizes the failure.
See the general section titled "Error (normative)".

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

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

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

- If the <listTargetsResponse> contains only one <target> then the <target> MAY omit "targetID".

- If the <listTargetsResponse> contains more than one <target> then each <target> MUST specify "targetID".

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

SupportedSchemaEntity targetID. A <supportedSchemaEntity> SHOULD specify a "targetID".

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

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

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

Capabilities. A target may also declare a set of capabilities that it supports. Each capability defines optional operations or semantics. For general information, see the subsection titled "Capabilities" within the "Concepts" section.

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

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

- Each <capability> element MUST specify (as the value of its "namespaceURI" attribute) an XML namespace that *identifies* the capability.
- Each <capability> element MAY specify (as the value of its "location" attribute) the URL of an XML schema that defines any structure that is associated with the capability (e.g., an SPML request/response pair that defines an operation—see below).

Capability operations. An XML schema document that a capability "location" attribute specifies MAY define operations. An XML schema document for a capability MUST define any operation as a paired request and response such that both of the following are true:

- The (XSD type of the) request (directly or indirectly) extends {RequestType}
- The (XSD type of the) response (directly or indirectly) extends {ResponseType}

Capability appliesTo. A target may support a capability for *all* of the target's supported schema entities or only for *a specific subset* of the target's supported schema entities. Each capability element may specify any number of supported schema entities to which it applies. A capability that does not specify a supported schema entity to which it applies must apply to every supported schema entity.

A <capability> element MAY contain any number of <appliesTo> elements.

A <capability> element that contains no <appliesTo> element MUST apply to *every* schema entity that the target supports. If the XML schema for the capability defines an operation, the provider MUST support the capability-defined operation for (any object that is instance of) any schema entity that the target supports. If the capability implies semantic meaning, then the provider MUST apply that semantic meaning to (every object that is an instance of) any schema entity that the target supports.

Capability appliesTo entityName. Each <appliesTo> element MUST have an "entityName" attribute that refers to a supported schema entity of the containing target. (See the topic named "Supported Schema Entities entityName" earlier in this section.)

- In the XSD Profile, each <appliesTo> element MUST specify a QName (as the value of its "entityName" attribute).
- In the DSMLv2 Profile **[SPMLv2-Profile-DSML]**, each <appliesTo> element MUST specify the name of an objectclass (as the value of its "entityName" attribute).

An <appliesTo> element MAY have a "targetID" attribute.

- A provider MAY omit "targetID" in any <appliesTo>. (That is, a provider MAY omit the optional "targetID" attribute of {SchemaEntityRefType} in an <appliesTo> element.)
- Any "targetID" MUST refer to the containing target. (That is, any "targetID" attribute of an <appliesTo> element MUST contain the same value as the "targetID" attribute of the <target> element that contains the <appliesTo> element.)

Capability content. SPMLv2 specifies only the optional <appliesTo> element as content for most capability elements. However, a declaration of support for the reference capability is special.

Reference Capability content. A <capability> element that refers to the Reference Capability (i.e., any <capability> element that specifies "namespaceURI='urn:oasis:names:tc:SPML:2.0:reference'") MUST contain (as open content) at least one <referenceDefinition> element. (For normative specifics, please see the topic named "Reference Definition" immediately below.

For background and for general information, please see the section titled "Reference Capability". For Reference Capability XSD, please see Appendix F.)

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

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

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

- The <schemaEntity> MUST have an "entityName" attribute that refers to a supported schema entity of the containing target. (See topic named the "Supported Schema Entities" earlier in this section.)

- The <schemaEntity> MAY have a "targetID" attribute. Any "targetID" that the <schemaEntity> specifies MUST refer to the containing target. (That is, any "targetID" value that a <schemaEntity> specifies MUST match the value of the "targetID" attribute of the <target> element that contains the <referenceDefinition>.)

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

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

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

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

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

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

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

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

contains no <referenceDataType> element implicitly declares that an instance of that type of reference will never contain reference data.

- A <referenceDataType> element MUST have an "entityName" attribute that refers to a supported schema entity. The value of the "entityName" attribute MUST be the name of a top-level entity that is valid in the schema.
- A <referenceDataType> element SHOULD have a "targetID" attribute.
  - If the <listTargetsResponse> contains only one <target>, then any <referenceDataType> element MAY omit "targetID".
  - If the <listTargetsResponse> contains more than one <target>, then any <referenceDataType> element MUST specify "targetID".
  - If the <referenceDataType> element specifies "targetID", then the "entityName" attribute (of the <canReferTo> element) MUST refer to a supported schema entity of the specified target (i.e., the <target> whose "targetID" value matches the "targetID" value that the <referenceDataType> element specifies).
  - If the <referenceDataType> element does not specify "targetID", then the "entityName" attribute (of the <canReferTo> element) MUST refer to a supported schema entity of the containing target (i.e., the <target> that contains the <referenceDefinition>).

3.6.1.1.3 listTargets Examples (non-normative)

In the following example, a requestor asks a provider to list the targets that the provider exposes for provisioning operations.

```
<listTargetsRequest/>
```

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

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

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

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

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

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

Reference Definitions. Within `target1`’s declaration of the Reference Capability for `Account`, the provider also declares two types of references: `owner` and `memberOf`. The provider declares that an instance of `Account` on `target1` may refer to an instance of `Person` on `target2` as its `owner`. An instance of `Account` on `target1` may also use a `memberOf` type of reference to refer 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"/>
```

3.6.1.2.1 addRequest (normative)

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

Execution. A `<addRequest>` MAY specify "`executionMode`".
See the section titled "Determining execution mode".

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

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

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

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

- A requestor that wants to bind a new object *in the top-level namespace of a target* MUST NOT supply `<containerID>`.
- A requestor that wants to bind a new object *beneath a specific object on a target* MUST supply `<containerID>`. Any `<containerID>` must identify an existing object. (That is, the content of `<containerID>` in an `<addRequest>` must match the `<psoID>` of an object that already exists on the target.)

Data. An `<addRequest>` MUST contain a `<data>` element that supplies initial content for the new object. A `<data>` element MUST contain only elements and attributes defined by the target schema as valid for the schema entity of which the object to be added is an instance.

CapabilityData. An `<addRequest>` element MAY contain any number of `<capabilityData>` elements. (Each `<capabilityData>` element contains data specific to a single capability. Each `<capabilityData>` element may contain any number of items of capability-specific data. Capability-specific data need not be defined by the target schema as valid for schema entity of which the object to be added is an instance.
See the section titled "CapabilityData in a Request (normative)".

ReturnData. An `<addRequest>` MAY have a `"returnData"` attribute that tells the provider which types of data to include in the provider's response.

- A requestor that wants the provider to return *nothing* of the added object MUST specify `"returnData='nothing'"`.
- A requestor that wants the provider to return *only the identifier* of the added object MUST specify `"returnData='identifier'"`.
- A requestor that wants the provider to return the identifier of the added object *plus the XML representation of the object (as defined in the schema of the target)* MUST specify `"returnData='data'"`.
- A requestor that wants the provider to return the identifier of the added object plus the XML representation of the object (as defined in the schema of the target) *plus any capability-specific data that is associated with the object* MAY specify `"returnData='everything'"` or MAY omit the `"returnData"` attribute (since `"returnData='everything'"` is the default).

3.6.1.2.2 addResponse (normative)

A provider that receives an `<addRequest>` from a requestor that the provider trusts MUST examine the content of the `<addRequest>`. If the request is valid, the provider MUST create the requested object under the specified parent (i.e., target or container object) if it is possible to do so.

PSO Identifier. The provider MUST create the object with any `<psoID>` that the `<addRequest>` supplies. If the provider cannot create the object with the specified `<psoID>` (e.g., because the `<psoID>` is not valid or because an object that already exists has that `<psoID>`), then the provider must fail the request. See the topic named "Error" below within this section.

Data. The provider MUST create the object with any XML element or attribute contained by the `<data>` element in the `<addRequest>`.

CapabilityData. The provider SHOULD associate with the created object the content of each `<capabilityData>` that the `<addRequest>` contains. The `"mustUnderstand"` attribute of each `<capabilityData>` indicates whether the provider MUST process the content of the `<capabilityData>` *as the corresponding capability specifies*. See the sections titled "CapabilityData in a Request (normative)" and "CapabilityData Processing (normative)".

Also see the section titled "CapabilityData Errors (normative)".

Execution. If an <addRequest> does not specify a type of execution, a provider MUST choose a type of execution for the requested operation.
See the section titled "Determining execution mode".

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

Status. The <addResponse> MUST have a "status" attribute that indicates whether the provider successfully created the requested object. See the section titled "Status (normative)".

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

- A <pso> element MUST contain a <psoID> element.
  The <psoID> element MUST contain the identifier of the newly created object.
  See the section titled "PSO Identifier (normative)".
  - If the <addRequest> supplies a <psoID>, then <psoID> of the newly created object MUST match the <psoID> supplied by the <addRequest>.
    (See the topic named "PSO Identifier" above within this section.)
  - If the <addRequest> does not supply <psoID>, the provider must generate a <psoID> that uniquely identifies the newly created object.
- A <pso> element MAY contain a <data> element.
  - If the <addRequest> specified "returnData='identifier'"
    then the <pso> MUST NOT contain a <data> element.
  - Otherwise, if the <addRequest> specified "returnData='data'"
    or (if the <addRequest> specified) "returnData='everything'"
    or (if the <addRequest>) omitted the "returnData" attribute,
    then the <pso> MUST contain exactly one <data> element that contains the XML representation of the object.
    This XML must be valid according to the schema of the target for the schema entity of which the newly created object is an instance.
- A <pso> element MAY contain any number of <capabilityData> elements. Each <capabilityData> element contains a set of *capability-specific data* that is associated with the newly created object (for example, a *reference* to another object).
  See the section titled "CapabilityData in a Response (normative)".
  - If the <addRequest> "returnData='identifier'"
    or (if the <addRequest> specified) "returnData='data'"
    then the <addResponse> MUST NOT contain a <capabilityData> element.
  - Otherwise, if the <addRequest> specified "returnData='everything'"
    or (if the <addRequest>) omitted the "returnData" attribute
    then the <addResponse> MUST contain a <capabilityData> element for each set of capability-specific data that is associated with the newly created object.

Error. If the provider cannot create the requested object, the <addResponse> MUST contain an "error" attribute that characterizes the failure. See the general section titled "Error (normative)".

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

- An <addRequest> specifies "targetID" but the value of "targetID" does not identify a target that the provider supports.
  In this case, the <addResponse> SHOULD specify "error='noSuchIdentifier'".

- An <addRequest> specifies "targetID" and (the <addRequest> also) contains <containerID> but the value of the "targetID" attribute in the <addRequest> does not match the value of the "targetID" attribute in the <containerID>.
  In this case, the <addResponse> SHOULD specify "error='malformedRequest'".

- An <addRequest> contains <containerID> but the content of <containerID> does not identify an object that exists. (That is, <containerID> does not match the <psoID> of an object that exists.)
  In this case, the <addResponse> SHOULD specify "error='noSuchIdentifier'".

- An <addRequest> contains <containerID> but the <supportedSchemaEntity> (of which <containerID> identifies an instance) does not specify "isContainer='true'"
  In this case, the <addResponse> SHOULD specify "error='invalidContainment'".

- An <addRequest> contains <containerID> but the target does not allow the specified parent object to contain the object to be created.
  In this case, the <addResponse> SHOULD specify "error='invalidContainment'".

- An <addRequest> supplies <psoID> but the <psoID> element is not valid.
  In this case, the <addResponse> SHOULD specify "error='invalidIdentifier'".

- An <addRequest> supplies <psoID> but an object with that <psoID> already exists.
  In this case, the <addResponse> SHOULD specify "error='alreadyExists'".

- The <data> element is missing an element or attribute that is required (according to the schema of the target) for the object to be added.

- A <capabilityData> element specifies "mustUnderstand='true'" and the provider cannot associate the content of the <capabilityData> with the object to be created.

The provider MAY return an error if:

- The <data> element contains data that the provider does not recognize as *valid according to the target schema* for the type of object to be created.

- The provider does not recognize the content of a <capabilityData> element as specific to any capability that the target supports (for the schema entity of which the object to be created is an instance).

Also see the section titled "CapabilityData Errors (normative)".

3.6.1.2.3 add Examples (non-normative)

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

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

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

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

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

3.6.1.3 lookup

The lookup operation enables a requestor to *obtain the XML that represents an object* on a target. The lookup operation also obtains any *capability-specific data* that is associated with the object.

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>
```

```
            <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"/>
```

3.6.1.3.1 lookupRequest (normative)

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

Execution. A <lookupRequest> MAY specify "executionMode". See the section titled "Determining execution mode".

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

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

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

- A requestor that wants the provider to return *nothing* of a requested object MUST specify "returnData='nothing'".
- A requestor that wants the provider to return *only the identifier* of a requested object MUST specify "returnData='identifier'".
- A requestor that wants the provider to return the identifier of a requested object *plus the XML representation of the object (as defined in the schema of the target)* MUST specify "returnData='data'".
- A requestor that wants the provider to return the identifier of a requested object plus the XML representation of the object (as defined in the schema of the target) *plus any capability-specific data that is associated with the object* MAY specify "returnData='everything'" or MAY omit the "returnData" attribute (since "returnData='everything'" is the default).

3.6.1.3.2 lookupResponse (normative)

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

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

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

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

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

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

- A <pso> element MUST contain a <psoID> element.
  The <psoID> element MUST contain the identifier of the requested object.
  See the section titled "PSO Identifier (normative)".
- A <pso> element MAY contain a <data> element.
  - If the <lookupRequest> specified "returnData='identifier'",
    then the <pso> MUST NOT contain a <data> element.
  - Otherwise, if the <lookupRequest> specified "returnData='data'"
    or (if the <lookupRequest> specified) "returnData='everything'"
    or (if the <lookupRequest>) omitted the "returnData" attribute
    then the <data> element MUST contain the XML representation of the object.
    This XML must be valid according to the schema of the target for the schema entity of which the newly created object is an instance.
- A <pso> element MAY contain any number of <capabilityData> elements.
  Each <capabilityData> element MUST contain all the data (that are associated with the object and) that are specific to the capability that the <capabilityData> specifies as "capabilityURI". For example, a <capabilityData> that refers to the Reference Capability (i.e., a <capabilityData> that specifies "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'")
  must contain at least one *reference* to another object.
  See the section titled "CapabilityData in a Response (normative)".
  - If the <lookupRequest> specified "returnData='identifier'"
    or (if the <lookupRequest> specified) "returnData='data'"
    then the <pso> MUST NOT contain a <capabilityData> element.
  - Otherwise, if the <lookupRequest> specified "returnData='everything'"
    or (if the <lookupRequest>) omitted the "returnData" attribute,
    then the <pso> MUST contain a <capabilityData> element
    for each set of capability-specific data that is associated with the requested object

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

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

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

- A <lookupRequest> contains no <psoID>.
- A <lookupRequest> contains a <psoID> that does not identify an object that exists on a target.

The provider MAY return an error if:

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

3.6.1.3.3 lookup Examples (non-normative)

In the following example, a requestor asks a provider to return the Person object from the add examples above. The requestor specifies the <psoID> for the Person object.

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

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

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

Next, the requestor asks a provider to return the Account object from the add examples above. The requestor specifies a <psoID> for the Account object.

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

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

The <pso> element contains a <psoID> element that uniquely identifies the object. The <pso> also contains a <data> element that contains the XML representation of the object (according to the schema of the target).

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

```
<lookupResponse requestID=”126” 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="2244" targetID=“target2”/>
            </reference>
        </capabilityData>
    </pso>
</lookupResponse>
```

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

```
<lookupRequest requestID=”129” returnData=”identifier”>
    <psoID ID="1431" targetID=“target1”/>
</lookupRequest>
```

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

```
<lookupResponse requestID=”129” status=“success">
    <pso>
        <psoID ID="1431" targetID=“target1”/>
    </pso>
</lookupResponse>
```

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

```
<lookupRequest requestID=”130” returnData=”data”>
    <psoID ID="1431" targetID=“target1”/>
</lookupRequest>
```

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

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

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

```
<lookupResponse requestID=”130” status=“success">
    <pso>
        <psoID ID="1431" targetID=“target1”/>
        <data>
            <Account accountName=”joebob”/>
        </data>
    </pso>
</lookupResponse>
```

3.6.1.4 modify

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

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

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

For normative specifics, please see the section titled "PSO Identifier (normative)".

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

The default processing for capability-specific data is to treat the content of each <capabilityData> as opaque. See the section titled "CapabilityData".

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>

   <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"/>
```

3.6.1.4.1 modifyRequest (normative)

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

Execution. A `<modifyRequest>` MAY specify "`executionMode`". See the section titled "Determining execution mode".

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

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

- A requestor that wants the provider to return *nothing* of the modified object MUST specify "returnData='nothing'".
- A requestor that wants the provider to return *only the identifier* of the modified object MUST specify "returnData='identifier'".
- A requestor that wants the provider to return the identifier of the modified object *plus the XML representation of the object (as defined in the schema of the target)* MUST specify "returnData='data'".
- A requestor that wants the provider to return the identifier of the modified object plus the XML representation of the object (as defined in the schema of the target) *plus any capability-specific data that is associated with the object* MAY specify "returnData='everything'" or MAY omit the "returnData" attribute (since "returnData='everything'" is the default).

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

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

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

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

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

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

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

3.6.1.4.2 modifyResponse (normative)

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

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

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

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

Status. The <modifyResponse> must have a "status" attribute that indicates whether the provider successfully applied the requested modifications to each identified object.
See the section titled "Status (normative)".

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

- A <pso> element MUST contain a <psoID> element.
  The <psoID> element MUST contain the identifier of the requested object.
  See the section titled "PSO Identifier (normative)".
- A <pso> element MAY contain a <data> element.
  - If the <modifyRequest> specified "returnData='identifier'",
    then the <pso> MUST NOT contain a <data> element.
  - Otherwise, if the <modifyRequest> specified "returnData='data'"
    or (if the <modifyRequest> specified) "returnData='everything'"
    or (if the <modifyRequest>) omitted the "returnData" attribute
    then the <data> element MUST contain the XML representation of the object.
    This XML must be valid according to the schema of the target for the schema entity of which the newly created object is an instance.
- A <pso> element MAY contain any number of <capabilityData> elements. Each <capabilityData> element contains a set of capability-*specific data* that is associated with the newly created object (for example, a *reference* to another object).
  See the section titled "CapabilityData in a Response (normative)".
  - If the <modifyRequest> specified "returnData='identifier'"
    or (if the <modifyRequest> specified) "returnData='data'"
    then the <modifyResponse> MUST NOT contain a <capabilityData> element.
  - Otherwise, if the <modifyRequest> specified "returnData='everything'"
    or (if the <modifyRequest>) omitted the "returnData" attribute,
    then the <modifyResponse> MUST contain a <capabilityData> element for each set of capability-specific data that is associated with the requested object
    (and that is specific to a capability that the target supports for the schema entity of which the requested object is an instance).

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

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

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

The provider MAY return an error if:

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

In addition, see the section titled "SelectionType Errors (normative)" as well as the section titled "CapabilityData Errors (normative)".

3.6.1.4.3 modify Examples (non-normative)

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

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

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

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

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

```
<modifyResponse requestID=”124” 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>
        <capabilityData mustUnderstand="true"
capabilityURI=”urn:oasis:names:tc:SPML:2.0:reference”>
            <reference typeOfReference="owns">
                <toPsoID ID="1431" targetID=“target1”/>
            </reference>
        </capabilityData>
    </pso>
</modifyResponse>
```

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

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

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

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

```
<modifyResponse requestID=”122” status=“success">
    <pso>
        <psoID 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">
                <toPsoID ID="group1" targetID=“target1”/>
            </reference>
            <reference typeOfReference="owner">
                <toPsoID ID="2245" targetID=“target2”/>
            </reference>
        </capabilityData>
    </pso>
</modifyResponse>
```

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

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

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

```
<modifyResponse requestID=”122” status=“success">
    <pso>
        <psoID 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">
                <toPsoID ID="group1" targetID=“target1”/>
            </reference>
            <reference typeOfReference="owner">
                <toPsoID ID="2245" targetID=“target2”/>
            </reference>
        </capabilityData>
    </pso>
</modifyResponse>
```

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

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

The provider returns a `<modifyResponse>` element. The `“status”` attribute of the `<modifyResponse>` element indicates that the modify request was successfully processed. The `<pso>` element of the `<modifyResponse>` shows that the foo-specific `<capabilityData>` has been removed.

```
<modifyResponse requestID=”122” 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>
```

Modifying a reference. The previous topic illustrates the default processing of capability data. The Reference Capability specifies enhanced behavior for the modify operation. See the section titled "Reference CapabilityData Processing (normative)".

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

Since SPMLv2 does not specify any mechanism to define the cardinality of a type of reference, a requestor should not assume that a provider enforces any specific cardinality for any type of reference. For a general discussion of the issues surrounding references, see the section titled “Reference Capability”.

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

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

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

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

3.6.1.5 delete

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

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"/>
```

3.6.1.5.1 deleteRequest (normative)

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

Execution. A `<deleteRequest>` MAY specify `“executionMode”`. See the section titled "Determining execution mode".

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

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

- A requestor that wants the provider to *delete any object that the specified object contains* (along with the specified object) MUST specify "recursive='true'".

- A requestor that wants the provider to delete the specified object *only if the specified object contains no other object* MUST NOT specify "recursive='true'". Such a requestor MAY specify "recursive='false'" or (such a requestor MAY) omit the "recursive" attribute (since "recursive='false'" is the default).

3.6.1.5.2 deleteResponse (normative)

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

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

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

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

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

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

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

- The <psoID> sub-element of the <deleteRequest> is empty (that is, the identifier element has no content). In this case, the <deleteResponse> SHOULD specify "error='noSuchIdentifier'".

- The <psoID> sub-element of the <deleteRequest> contains invalid data. In this case the provider SHOULD return "error='unsupportedIdentifierType'".

- The <psoID> sub-element of the <deleteRequest> does not specify an object that exists. In this case the <deleteResponse> MUST specify "error='noSuchIdentifier'".

- The <psoID> sub-element of the <deleteRequest> specifies an object that contains another object and the <deleteRequest> does not specify "recursive='true'". In such a case the provider should return "error='containerNotEmpty'".

3.6.1.5.3 delete Examples (non-normative)

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

3.6.2 Async Capability

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

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

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

1) MUST allow a requestor to specify `"executionMode='asynchronous'"`.
   The provider MUST NOT fail such a request with
   `"error='unsupportedExecutionMode'"`.
   The provider MUST execute the requested operation asynchronously
   (if the provider executes the requested operation at all).
   See the section titled "Requestor specifies asynchronous execution (normative)".

2) MAY choose to execute a requested operation asynchronously
   when the request does not specify the `"executionMode"` attribute.
   See the section titled "Provider chooses asynchronous execution (normative)".

The Async Capability also defines two operations that a requestor may use to manage another operation that a provider is executing asynchronously:

- A status operation allows a requestor to check the status (and possibly results) of an operation.
- A cancel operation asks the provider to stop executing an operation.

Status. When a provider is executing SPML operations asynchronously, the requestor needs a way to check the status of requests. The status operation allows a requestor to determine whether an asynchronous operation has succeeded or has failed or is still pending. The status operation also allows a requestor to obtain the output of an asynchronous operation.

Cancel. A requestor may also need to cancel an asynchronous operation. The cancel operation allows a requestor to ask a provider to stop executing an asynchronous operation.

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

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

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

These decisions may be governed by the provider's implementation, by its configuration, or by runtime computation.

A provider that wishes to *never to store the results of operations* SHOULD NOT declare that it supports the Async Capability. (Such a provider may *internally* execute requested operations asynchronously, but must respond to each request exactly as if the request had been processed synchronously.)

A provider that wishes to support the asynchronous execution of requested operations MUST store the results of an asynchronous operation *for a reasonable period of time* in order to allow the requestor to obtain those results. SPMLv2 does not specify a minimum length of time.

As a practical matter, a provider cannot queue the results of asynchronous operations forever. The provider must eventually release the resources associated with asynchronous operation results. (Put differently, a provider must eventually discard the results of an operation that the provider executes asynchronously.) Otherwise, the provider may run out of resources.

Providers should carefully manage the resources associated with operation results. For example:

- A provider may define a *timeout interval* that specifies the maximum time between status requests. If a requestor does not request the status of asynchronous operation within this interval, the provider will release the results of the asynchronous operation. (Any subsequent request for status on this asynchronous operation will receive a response that specifies `"error='noSuchRequest'"`.)

- A provider may also define an overall *result lifetime* that specifies the maximum length of time to retain the results of an asynchronous operation. After this amount of time has passed, the provider will release the results of the operation.

- A provider may also wish to enforce an *overall limit* on the resources available to store the results of asynchronous operations, and may wish to adjust its behavior (or even to refuse requests for asynchronous execution) accordingly.

- To prevent denial of service attacks, the provider should not allocate any resource on behalf of a requestor until that requestor is properly authenticated.

3.6.2.1 cancel

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

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"/>
```

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

Cancel is not batchable. Because the cancel operation must be synchronous, a requestor must not nest a cancel request in a batch request.

3.6.2.1.1 cancelRequest (normative)

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

Execution. A `<cancelRequest>` MUST NOT specify `"executionMode='asynchronous'"`. A `<cancelRequest>` MUST specify `"executionMode='synchronous'"` or (a `<cancelRequest>` MUST) omit the `"executionMode"` attribute. See the section titled "Determining execution mode".

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

3.6.2.1.2 cancelResponse (normative)

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

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

Execution. The provider MUST execute the cancel operation synchronously (if the provider executes the cancel operation at all). See the section titled "Determining execution mode".

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

Status. A `<cancelResponse>` must have a `"status"` attribute that indicates whether the provider successfully processed the request to cancel the specified operation. See the section titled "Status (normative)".

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

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

Error. If the provider cannot cancel the specified operation, the <cancelResponse> MUST contain an "error" attribute that characterizes the failure.
See the general section titled "Error (normative)".

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

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

3.6.2.1.3 cancel Examples (non-normative)

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

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

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

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

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

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

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

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

3.6.2.2 status

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

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

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

Status is not batchable. Because the status operation must be synchronous, a requestor must not nest a status request in a batch request.

3.6.2.2.1 statusRequest (normative)

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

Execution. A `<statusRequest>` MUST NOT specify `"executionMode='asynchronous'"`. A `<statusRequest>` MUST specify `"executionMode='synchronous'"` or (a `<statusRequest>` MUST) omit "`executionMode`".
See the section titled "Determining execution mode".

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

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

3.6.2.2.2 statusResponse (normative)

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

Execution. The provider MUST execute the status operation synchronously (if the provider executes the status operation at all). See the section titled "Determining execution mode".

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

- If a <statusRequest> specifies "returnResults='true'", then the provider MUST also return in the <statusResponse> any results (or output) of each operation.
- If a <statusRequest> specifies "returnResults='false'", then the provider MUST return in the <statusResponse> only the "status" of the each operation.
- If the <statusRequest> does not specify a value for "returnResults", the provider MUST assume that the requestor wants only the "status" (and the provider MUST NOT return in the <statusResponse> any result) of the operation that is executing asynchronously.

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

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

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

- If the provider successfully obtained the status of the specified operation (and successfully obtained any output of the specified operation if the <statusRequest> specifies "returnOutput='true'"), the <statusResponse> MUST specify "status='success'".
- If the provider failed to obtain the status of the specified operation (or failed to obtain any output of the specified operation if the <statusRequest> specifies "returnOutput='true'"), the <statusResponse> MUST specify "status='failure'".

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

- A <statusResponse> that specifies "status='failure'" MUST NOT contain an embedded response. Since the status operation failed, the response should not contain data.
- A <statusResponse> that specifies "status='success'" MAY contain any number of responses.
  - If the <statusRequest> specifies "asyncRequestID", then a successful <statusResponse> MUST contain *exactly one nested response* that represents the operation that "asyncRequestID" specifies.

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

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

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

- A nested response that represents an operation that failed MUST specify "status='failure'".
- A nested response that represents an operation that succeeded MUST specify "status='success'".
- A nested response that represents an operation that the provider is still executing MUST specify "status='pending'".

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

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

Error. If the provider cannot obtain the status of the specified operation, the <statusResponse> MUST contain an "error" attribute that characterizes the failure.
See the general section titled "Error (normative)".

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

- The "asyncRequestID" attribute of the <statusRequest> has no value. In this case, the <statusResponse> SHOULD specify "error='invalidIdentifier'".
- The "asyncRequestID" attribute of the <statusRequest> has a value, but does not identify an operation for which the provider retains status and results. In this case the provider SHOULD return "error='noSuchRequest'".

3.6.2.2.3 status Examples (non-normative)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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>

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

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

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

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

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

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

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

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

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

- When a `<batchRequest>` specifies "`processing='sequential'`", the provider must execute each requested operation *one at a time and in the exact order* that it occurs within the `<batchRequest>`.

- When a `<batchRequest>` specifies "`processing='parallel'`", the provider may execute the requested operations within the `<batchRequest>` *in any order*.

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

- When a `<batchRequest>` specifies "`onError='exit'`", the provider stops executing individual operations within the batch as soon as the provider encounters an error.
  Any operation that produces an error is marked as failed.
  Any operation that the provider does not execute is also marked as failed.

- When a <batchRequest> specifies "onError='resume'", the provider handles any error that occurs in processing an individual operation within that <batchRequest>.
  No error that occurs in processing an individual operation prevents execution of any other individual operation in the batch.
  Any operation that produces an error is marked as failed.

(Note that a requestor can guarantee pre-requisite processing in batch operations by specifying both "processing='sequential'" and "onError='exit'".)

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

3.6.3.1.1 batchRequest (normative)

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

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

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

- {spml:ListTargetsRequestType}
- {spmlbatch:BatchRequestType}
- {spmlsearch:SearchRequestType}
- {spmlsearch:IterateRequestType}
- {spmlsearch:CloseIteratorRequestType}
- {spmlasync:CancelRequestType}
- {spmlasync:StatusRequestType}
- {spmlupdates:UpdatesRequestType}
- {spmlupdates:IterateRequestType}
- {spmlupdates:CloseIteratorRequestType}

Processing. A <batchRequest> MAY specify "processing". The value of any "processing" attribute MUST be either 'sequential' or 'parallel'.

- A requestor who wants the provider to process the nested requests *concurrently with one another* MUST specify "processing='parallel'".

- A requestor who wants the provider to process the nested requests one-by-one and in the order that they appear MAY specify "processing='sequential'".

- A requestor who does not specify "processing" is *implicitly* asking the provider to process the nested requests *sequentially*.

onError. A <batchRequest> MAY specify "onError". The value of any "onError" attribute MUST be either 'exit' or 'resume'.

- A requestor who wants the provider to *continue processing* nested requests whenever processing one of the nested requests produces in an error MUST specify "onError='resume'".

- A requestor who wants the provider to *cease processing* nested requests as soon as processing any of the nested requests produces an error MAY specify "`onError='exit'`".

- A requestor who does not specify an "`onError`" attribute *implicitly* asks the provider to cease processing nested requests as soon as processing any of the nested requests produces an error.

3.6.3.1.2 batchResponse (normative)

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

processing. If a `<batchRequest>` specifies "`processing='parallel'`", the provider SHOULD begin executing each of the nested requests as soon as possible. (Ideally, the provider would begin executing all of the nested requests immediately and concurrently.) If the provider cannot begin executing all of the nested requests at the same time, then the provider SHOULD begin executing *as many as possible* of the nested requests *as soon as possible*.

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

onError. The effect (on the provider's behavior) of the "`onError`" attribute of a `<batchRequest>` depends on the "`processing`" attribute of the `<batchRequest>`.

- If a `<batchRequest>` specifies (or defaults to) "`onError='exit'`" and (the `<batchRequest>` specifies or defaults to) "`processing='sequential'`" then the provider MUST NOT execute any (operation that is described by a) nested request that is subsequent to the first nested request that produces an error.

  If the provider encounters an error in executing (the operation that is described by) a nested request, the provider MUST report the error in the nested response that corresponds to the nested request and then (the provider MUST) specify "`status='failure'`" in every nested response that corresponds to a subsequent nested request within the same `<batchRequest>`. The provider MUST also specify "`status='failure'`" in the overall `<batchResponse>`.

- If a `<batchRequest>` specifies (or defaults to) "`onError='exit'`" and (the `<batchRequest>` specifies) "`processing='parallel'`" then the provider's behavior once an error occurs (in processing an operation that is described by a nested request) is *not fully specified*.

  If the provider encounters an error in executing (the operation that is described by) a nested request, the provider MUST report the error in the nested response that corresponds to the nested request. The provider MUST also specify "`status='failure'`" in the overall `<batchResponse>`. The provider MUST also specify "`status='failure'`" in the nested response that corresponds to any operation the provider has not yet begun to execute. However, the provider's behavior with respect to any operation that has already begun to execute but that is not yet complete is not fully specified.

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

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

- If a <batchRequest> specifies "onError='resume'" and (the <batchRequest> specifies) "processing='parallel'", then the provider MUST execute every (operation that is described by a) nested request within the <batchRequest>. If the provider encounters an error in executing any (operation that is described by a) nested request, the provider MUST report the error in the nested response that corresponds to the nested request and then (the provider MUST) specify "status='failure'" in the overall <batchResponse>.

- If a <batchRequest> specifies "onError='resume'" and (the <batchRequest> specifies or defaults to) "processing='sequential'", then the provider MUST execute every (operation that is described by a) nested request within the <batchRequest>. If the provider encounters an error in executing any (operation that is described by a) nested request, the provider MUST report the error in the nested response that corresponds to the nested request and then (the provider MUST) specify "status='failure'" in the overall <batchResponse>.

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

Status. The <batchResponse> must contain a "status" attribute that indicates whether the provider successfully processed every nested request.
See the section titled "Status (normative)".

- If the provider successfully executed every (operation described by a) nested request, then the <batchResponse> MUST specify "status='success'".

- If the provider encountered an error in processing (the operation described by) any nested request, the <batchResponse> MUST specify "status='failure'".

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

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

- Each nested response that corresponds to a nested request *that the provider did not process* MUST specify "status='failed'". (A provider might not process a nested request, for example, if the provider encountered an error processing an earlier nested request and the requestor specified both "processing='sequential'" and "onError='exit'".)

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

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

3.6.3.1.3 batch Examples (non-normative)

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

The provider returns an `<batchResponse>` element. The “`status`” of the `<batchResponse>` indicates that all of the nested requests were processed successfully. The `<batchResponse>` contains an `<addResponse>` for each `<addRequest>` that the `<batchRequest>` contained. Each `<addResponse>` contains the same data that it would have contained if the corresponding `<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>
            <psoID ID=”1431” targetID=“target1”/>
            <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>
        </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:

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

3.6.4.1.1 bulkModifyRequest (normative)

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

Execution. A <bulkModifyRequest> MAY specify "executionMode".
See the section titled "Determining execution mode".

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

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

3.6.4.1.2 bulkModifyResponse (normative)

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

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

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

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

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

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

3.6.4.1.3 bulkModify Examples (non-normative)

In the following example, a requestor asks a provider to change every `Person` with an email address matching `'jbbriggs@example.com'` to have instead an email address of `'joebob@example.com'`.

```
<bulkModifyRequest>
    <query scope="subtree" targetID="target2">
        <select path="/Person/email='jbbriggs@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>
```

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

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

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

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

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

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

3.6.4.2 bulkDelete

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"/>
```

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

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

3.6.4.2.1 bulkDeleteRequest (normative)

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

Execution. A `<bulkDeleteRequest>` MAY specify `"executionMode"`.
See the section titled "Determining execution mode".

query. A `<bulkDeleteRequest>` MUST contain exactly one `<query>` element.
A `<query>` describes criteria that (the provider must use to) select objects on a target.
See the section titled "SearchQueryType in a Request (normative)".

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

3.6.4.2.2 bulkDeleteResponse (normative)

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

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

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

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

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

Status. The `<bulkDeleteResponse>` must contain a `"status"` attribute that indicates whether the provider successfully deleted every object that matched the specified query. See the section titled "Status (normative)".

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

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

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

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

3.6.4.2.3 bulkDelete Examples (non-normative)

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

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

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

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

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

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

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

3.6.5 Password Capability

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

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

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

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

3.6.5.1 setPassword

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

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

3.6.5.1.1 setPasswordRequest (normative)

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

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

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

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

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

3.6.5.1.2 setPasswordResponse (normative)

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

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

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

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

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

- The <setPasswordRequest> contains a <psoID> for an object that does not exist.
- The target system or application will not accept (as the new password) the value that a <setPasswordRequest> supplies as the content of the <password> element.
- The target system or application *requires the current password* in order to change the password and a <setPasswordRequest> supplies no content for <currentPassword>.
- The target system or application *requires the current password* in order to change the password and the target system or application will not accept (as the current password) the value that a <setPasswordRequest> supplies as the content of <currentPassword>.
- The target system or application *returns an error (or throws an exception)* when the provider tries to set the password.

3.6.5.1.3 setPassword Examples (non-normative)

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>
```

The provider returns a `<setPasswordResponse>` element. The `“status”` of the `<setPasswordResponse>` indicates that the provider successfully changed the password.

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

3.6.5.2 expirePassword

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

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

3.6.5.2.1 expirePasswordRequest (normative)

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

Execution. A `<expirePasswordRequest>` MAY specify `“executionMode”`. See the section titled "Determining execution mode".

psoID. A `<expirePasswordRequest>` MUST contain exactly one `<psoID>` element. The `<psoID>` MUST identify an object that exists on a target (that is supported by the provider). See the section titled "PSO Identifier (normative)".

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

3.6.5.2.2 expirePasswordResponse (normative)

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

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

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

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

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

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

3.6.5.2.3 expirePassword Examples (non-normative)

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>
```

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

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

3.6.5.3 resetPassword

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

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

3.6.5.3.1 resetPasswordRequest (normative)

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

Execution. A <resetPasswordRequest> MAY specify "executionMode". See the section titled "Determining execution mode".

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

3.6.5.3.2 resetPasswordResponse (normative)

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

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

Response. The provider must return to the requestor a <resetPasswordResponse>. The <resetPasswordResponse> must have a "status" attribute that indicates whether the provider successfully changed the password that is associated with the specified object and successfully returned to the requestor the new password value. See the section titled "Status (normative)".

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

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

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

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

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

3.6.5.3.3 resetPassword Examples (non-normative)

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>
```

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

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

3.6.5.4 validatePassword

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

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

3.6.5.4.1 validatePasswordRequest (normative)

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

Execution. A <validatePasswordRequest> MAY specify "executionMode". See the section titled "Determining execution mode".

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

password. A <validatePasswordRequest> MUST contain exactly one <password> element. The <password> element MUST contain a string value.

3.6.5.4.2 validatePasswordResponse (normative)

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

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

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

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

Error. If the provider cannot determine whether the specified value would be valid as the password that is associated with the specified object, then the <validatePasswordResponse> MUST specify an "error" value that characterizes the failure. See the general section titled "Error (normative)".

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

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

3.6.5.4.3 validatePassword Examples (non-normative)

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

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

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

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

- In order to *create an object with references*, a requestor specifies capability-specific data to the add operation.
- In order to *add, remove or replace references* to an object, a requestor specifies capability-specific data to the modify operation.
- In order to *obtain references* for an object, a requestor examines capability-specific data returned as output by the add, lookup and search operations.

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

- Managing references to other objects can be an important part of managing objects.
- Object references to other objects present a *scalability* problem.
- Object references to other objects present an *integrity* problem.

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

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

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

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

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

3.6.6.1 Reference Definitions

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

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

Each reference definition names a specific type of reference and also specifies the following:

- *which schema entity* (on the `<target>` that contains the `<capability>` that contains the `<referenceDefinition>`) *can refer…*
- *…to which schema entity* or schema entities (on which targets).

For normative specifics, see the topic named "Reference Capability content" within the section titled "listTargetsResponse (normative)".

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

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

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

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

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

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

ReferenceDataType. A reference definition may be *complex*, which means that an instance of that type of reference may have reference data associated with it.
See the section titled "Complex References" below.

The definition of a type of reference that is complex must contain a <referenceDataType> for each possible structure of reference data. Each <referenceDataType> element refers to a specific entity in a target schema. A <referenceData> element (within any instance of that type of reference) may contain one element of any of these types (to which a <referenceDataType> refers).

A reference definition that contains no <referenceDataType> sub-element indicates that the type of reference it defines *does not support reference data*.

For a normative description, see the topic named “ReferenceDefinition referenceDataType” within the section titled "listTargetsResponse (normative)".

3.6.6.2 References

Must contain toPsoID. Any <reference> MUST specify its “toObject”. That is, any instance of {ReferenceType} MUST contain a valid <toPsoID>. The only exception is a <reference> that is used as a wildcard within a <modification> that specifies “modificationMode='delete'”. In this case (and only in this case), the <reference> MUST specify a valid “typeOfReference” but (the <reference>) MAY omit <toPsoID>. See the section titled “Reference CapabilityData Processing (normative)”.

No duplicates. Within the set of references that is associated with an object, at most one <reference> of a specific "typeOfReference" may refer to a particular object. That is, an instance of {CapabilityDataType} MUST NOT contain two (and MUST NOT contain more than two) instances of <reference> that specify the same value of "typeOfReference" and that contain <toPsoID> elements that identify the same object. See the section titled “Reference CapabilityData in a Request (normative)”.

Reference Data. SPMLv2 allows each reference (i.e., each instance of {ReferenceType}) to contain additional reference data. Most references between objects require no additional data, but allowing references to contain additional data supports cases in which a reference from one object to another may carry additional information “on the arrow” of the relationship. For example, a RACF user’s membership in a particular RACF group carries with it the additional information of whether that user has the ADMINISTRATOR or SPECIAL privilege within that group. Several other forms of group membership carry with them additional information about the member’s expiration. See the section titled “Complex References” below.

Search. A requestor can *search for objects based on reference values* using the <hasReference> query clause. The {HasReferenceType} extends {QueryClauseType}, which indicates that an instance of {HasReferenceType} can be used to select objects. A <hasReference> clause matches an object if and only if the object has a reference that *matches every specified component* (i.e., element or attribute) of the <hasReference> element. See the section titled “search Examples”.

3.6.6.3 Complex References

The vast majority of reference types are simple: that is, one object’s reference to another object carries no additional information. However certain types of references may support additional information that is specific to a particular reference. For example, when a user is assigned to one or more Entrust GetAccess Roles, each role assignment has a start date and an end date. We describe a *reference that contains additional data* (where that data is specific to the reference) as a “complex” reference.

Example: RACF Group Membership is another example of a complex type of reference. Each RACF group membership carries with it additional data about whether the user has the SPECIAL, AUDITOR, or OPERATIONS privileges in that group.

- Group-SPECIAL gives a group administrator control over all profiles within the group
- Group-AUDITOR allows a user to monitor the use of the group's resources
- Group-OPERATIONS allows a user to perform maintenance operations on the group's resources

For purposes of this example, let us represent these three group-specific privileges as attributes of an XML type called "RacfGroupMembershipType". Suppose that the XML Schema for such a type 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"/>
```

The following subsections describe several different ways to model RACF Group Membership. The fictional `<xsd:schema>` is the same in all of the examples. In each subsection, however, the provider's `<target>` definition varies with the approach.

3.6.6.3.1 Using Reference Data

The simplest way to model a complex reference such as RACF Group membership is to represent the additional information as arbitrary *reference data*. The `<referenceData>` element within a `<reference>` may contain any data.

The following example shows how a provider's listTargetsResponse might reflect this approach. The sample schema for the "RACF" target is very simple (for the sake of brevity). The provider defines a type of reference called "memberOfGroup". Within a `<reference>` of this type, the `<referenceData>` element must contain exactly one `<racfGroupMembership>` element (and 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>
```

Manipulating Reference Data. The only way to manipulate the reference data associated with a complex reference is by using the modify operation that is part of the Core XSD. A requestor may add, replace or delete any capability-specific data that is associated with an object.

Capabilities Do Not Apply. SPML specifies no way to apply a capability-specific operation to a reference. Thus, for example, one can neither suspend nor resume a reference. This is because a *reference is not a provisioning service object*. A reference is instead *capability-specific data that is associated with an object*.

You can think of an object's references (or any set of capability-specific data that is associated with an object) as an “extra” attribute (or as an “extra” sub-element) of the object. The provider supports each “extra” (attribute or sub-element) data *independent of the schema* of the target that contains the object. The provider keeps all `<capabilityData>` separate from the regular schema-defined `<data>` within each `<pso>`.

3.6.6.3.2 Relationship Objects

The fact that capabilities cannot apply to references does not prevent a provider from offering this kind of rich function. There is an elegant way to represent a complex relationship that allows a

requestor to operate directly on the relationship itself. A provider may model a complex relationship between two objects as a third object that refers to each of the first two objects.

This approach is analogous to a “linking record” in relational database design. In the “linking record” approach, the designer “normalizes” reference relationships into a separate table. Each row in a third table connects a row from one table to a row in another table. This approach allows each relationship to carry additional information that is specific to that relationship. Data specific to each reference are stored in the columns of the third table. Even when relationships do not need to carry additional information, database designers often use this approach when two objects may be connected by more than one instance of the same type of relationship, or when relationships are frequently added or deleted and referential integrity must be maintained.

Rather than have an object A refer to an object B directly, a third object C refers to both object A and object B. Since object C represents the relationship itself, object C refers to object A as its “fromObject” and object C refers to object B as its “toObject”.

A provider that wants to treat each instance of a (specific type of) relationship as an object does so by defining in the schema for a target a schema entity to contain the additional information (that is specific to that type of relationship). The provider then declares two types of references that apply to that schema entity: a “fromObject” type of reference and a “toObject” type of reference. The provider may also declare that certain capabilities apply to that schema entity. This model allows a requestor to operate conveniently on each instance of a complex relationship.

For example, suppose that a provider models as a schema entity a type of relationship that has an effective date and has an expiration date. As a convenience to requestors, the provider might declare that this schema entity (that is, the “linking” entity) supports the Suspend Capability. The ‘suspend’ and ‘resume’ operations could manipulate the expiration date and the effective date *without the requestor having to understand the structure of that schema entity*. This convenience could be very valuable where the attribute values or element content that are manipulated have complex syntax, special semantics or implicit relationships with other elements or attributes.

The following example shows how a provider’s listTargetsResponse might reflect this approach. 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”/>
```

```
            <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>
```

3.6.6.4 Reference CapabilityData in a Request (normative)

The general rules that govern an instance of `{CapabilityDataType}` in a request also apply to an instance of `{CapabilityDataType}` that refers to the Reference Capability. See the section titled "CapabilityData in a Request (normative)".

capabilityURI. An instance of `{CapabilityDataType}` that contains data that are specific to the Reference Capability MUST specify `"capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'"`.

mustUnderstand. An instance of `{CapabilityDataType}` that refers to the Reference Capability SHOULD specify `"mustUnderstand='true'"`.

Capability defines structure. An instance of `{CapabilityDataType}` that refers to the Reference Capability MUST contain at least one `<reference>` element. An instance of `{CapabilityDataType}` that refers to the Reference Capability SHOULD NOT contain any element that is not a `<reference>` element.

No duplicates. Within the set of references that is associated with an object, at most one `<reference>` of a specific `"typeOfReference"` may refer to a specific object. That is, an instance of `{CapabilityDataType}` MUST NOT contain two (and MUST NOT contain more than two) instances of `<reference>` that specify the same value of `"typeOfReference"` and that contain `<toPsoID>` elements that identify the same object.

Validate each reference. Any `<reference>` that an instance of `{CapabilityDataType}` contains must be an instance of `{spmlref:ReferenceType}`. In addition, a provider MUST examine the following aspects of each `<reference>`:

- The *"from" object*. (The object that contains--or that is intended to contain--the reference.)
- The *"to" object.* (The object that the `<toPsoID>` of the reference identifies.)
- The *"from" schema entity*. (The schema entity of which the "from" object is an instance.)
- The *"to" schema entity* (The schema entity of which the "to" object is an instance.)
- The *typeOfReference*
- Any *referenceData*

The standard aspects of SPML that specify supported schema entities and capabilities imply the following:

- The "to" object MUST exist (on a target that the provider exposes).
- The target that contains the "from" object MUST support the "from" schema entity.
- The target that contains the "to" object MUST support the "to" schema entity.
- The target that contains the "from" object MUST support the Reference Capability.
- The target that contains the "from" object MUST declare that the Reference Capability applies to the "from" schema entity.

See the section titled "listTargetsResponse (normative)".

Check Reference Definition. In addition, a provider must validate the `"typeOfReference"` that each `<reference>` specifies (as well as the "from" schema entity and the "to" schema entity) against the set of valid reference definitions..
The `<capability>` that declares that the target (that contains the "from" object) supports the Reference Capability for the "from" schema entity MUST contain a `<referenceDefinition>` for which all of the following are true:

- The `<referenceDefinition>` specifies the same `"typeOfReference"` that the `<reference>` specifies
- The `<referenceDefinition>` contains a `<schemaEntity>` element that specifies the "from" schema entity
- The `<referenceDefinition>` contains a `<canReferTo>` element that specifies the "to" schema entity.

See the section titled "Reference Definitions" above.

3.6.6.5 Reference CapabilityData Processing (normative)

The general rules that govern processing of an instance of {CapabilityDataType} in a request also apply to an instance of {CapabilityDataType} that refers to the Reference Capability. See the section titled "CapabilityData Processing (normative)".

capabilityURI. An instance of {CapabilityDataType} that refers to the Reference Capability MUST specify "capabilityURI='urn:oasis:names:tc:SPML:2.0:reference'". The target (that contains the object to be manipulated) MUST support the Reference Capability for the schema entity of which the object to be manipulated is an instance.

mustUnderstand. An instance of {CapabilityDataType} that refers to the Reference Capability SHOULD specify "mustUnderstand='true'". A provider that supports the Reference Capability MUST handle the content as this capability specifies (regardless of the value of "mustUnderstand"). See the topic named "mustUnderstand" within the section titled "CapabilityData Processing (normative)".

Open content. An instance of {CapabilityDataType} that refers to the Reference Capability MUST contain at least one <reference>. An instance of {CapabilityDataType} that refers to the Reference Capability SHOULD NOT contain any element that is not a <reference>.

Validation. A provider MUST examine the content of any instance of {CapabilityDataType} that refers to the Reference Capability (regardless of the type of request that contains the instance of {CapabilityDataType}) and ensure that it contains only valid instances of <reference>. See the section titled "Reference CapabilityData in a Request (normative)".

If the content (of the instance of {CapabilityDataType} that refers to the Reference Capability) is not valid, then the provider's response MUST specify "status='failure'".
See the section titled "Request CapabilityData Errors (normative)".

Process individual references. In addition to the validation described above, the content of an instance of {CapabilityDataType} that refers to the Reference Capability is not treated as opaque, but instead as a set of individual references. The handling of each <reference> depends on the type of element that contains the instance of {CapabilityDataType}).

- If an <addRequest> contains an instance of {CapabilityDataType} that refers to the Reference Capability, then the provider MUST associate the instance of {CapabilityDataType} (and each <reference> that it contains) with the newly created object.

- If a <modification> contains an instance of {CapabilityDataType} that refers to the Reference Capability, then the handling of each <reference> (that the instance of {CapabilityDataType} contains) depends on the "modificationMode" of that <modification> and also depends on whether a matching <reference> is already associated with the object to be modified.

  - If the <modification> specifies "modificationMode='add'", then the provider MUST *add each new reference* for which no matching <reference> is already associated with the object.
    That is, the provider MUST associate with the object to be modified each <reference> (that the instance of {CapabilityDataType} within the <modification> contains) for which no <reference> that is already associated with the object specifies the same value for "typeOfReference" (that the <reference> from the <modification> specifies) and contains a <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from the <modification> identifies).

The provider MUST *replace each matching reference* that is already associated with the object with the <reference> from the <modification>.
That is, if a <reference> that is already associated with the object specifies the same value for "typeOfReference" (that the <reference> from the <modification> specifies) and if the <reference> that is already associated with the object contains a <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from the <modification> identifies), then the provider MUST *remove* the <reference> that is already associated with the object and (the provider MUST) *add* the <reference> from the <modification>.
This has the net effect of replacing any optional <referenceData> (as well as replacing any open content) of the matching <reference>.

- If the <modification> specifies "modificationMode='replace'",
then the provider MUST *add each new reference* for which no matching <reference> is already associated with the object.
That is, the provider MUST associate with the object to be modified each <reference> (that the instance of {CapabilityDataType} within the <modification> contains) for which no <reference> that is already associated with the object specifies the same value for "typeOfReference" (that the <reference> from the <modification> specifies) and contains a <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from the <modification> identifies).

  The provider MUST *replace each matching reference* that is already associated with the object with the <reference> from the <modification>.
  That is, if a <reference> that is already associated with the object specifies the same value for "typeOfReference" (that the <reference> from the <modification> specifies) and if the <reference> that is already associated with the object contains a <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from the <modification> identifies), then the provider MUST *remove* the <reference> that is already associated with the object and (the provider MUST) *add* the <reference> from the <modification>.
  This has the net effect of replacing any optional <referenceData> (as well as replacing any open content) of the matching <reference>.

- If the <modification> specifies "modificationMode='delete'",
then the provider MUST *remove each matching reference*.
A reference that omits <toPsoID> is *treated as a wildcard*.

  If the <reference> from the <modification> contains a <toPsoID> element,
then the provider MUST remove (from the set of references that are associated with the object) any <reference> that specifies the same value for "typeOfReference" (that the <reference> from the <modification> specifies) and that contains a <toPsoID> that identifies the same object (that the <toPsoID> of the <reference> from the <modification> identifies).

  If the <reference> from the <modification> contains no <toPsoID> element,
then the provider MUST remove (from the set of references that are associated with the object) any <reference> that specifies the same value for "typeOfReference" (that the <reference> from the <modification> specifies).

  If no instance of <reference> that is associated with the object to be modified matches 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)".

3.6.7 Search Capability

The Search Capability is defined in a schema associated with the following XML namespace: `urn:oasis:names:tc:SPML:2:0:search`. This document includes the Search Capability XSD as Appendix G.

The Search Capability defines three operations: search, iterate and closeIterator. The search and iterate operations together allow a requestor to obtain *in a scalable manner* the XML representation of every object that matches specified selection criteria. The search operation returns in its response a first set of matching objects. Each subsequent iterate operation returns more matching objects. The closeIterator operation allows a requestor to tell a provider that it does not intend to finish iterating a search result (and that the provider may therefore release the associated resources).

A provider that supports the search and iterate operations for a target SHOULD declare that the target supports the Search Capability. A provider that does not support both search and iterate MUST NOT declare that the target supports the Search Capability.

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

- How large of a search result the provider will *select* on behalf of a requestor.
- How large of a search result the provider will *queue* on behalf of a requestor (so that the requestor may iterate the search results).
- For *how long a time* the provider will queue a search result on behalf of a requestor.

These decisions may be governed by the provider's implementation, by its configuration, or by runtime computation.

A provider that wishes to *never to queue search results* may return every matching object (up to the provider's limit and up to any limit specified by the requestor) in the search response. Such a provider would never return an iterator, and would not need to support the iterate operation. The disadvantage is that, without an iterate operation, a provider's search capability either is limited to small results or produces large search responses.

A provider that wishes to support the iterate operation must store (or somehow queue) the objects selected by a search operation until the requestor has a chance to iterate those results. (That is, a provider must somehow queue the objects that matched the criteria of a search operation and that were not returned in the search response.)

If all goes well, the requestor will continue to iterate the search result until the provider has sent all of the objects to the requestor. The requestor may also use the closeIterator operation to tell the provider that the requestor is no longer interested in the search result. In either case, the provider may free any resource that is still associated with the search result. However, it is possible that the requestor may not iterate the search result in a timely manner--or that the requestor may *never* iterate the search result completely. Such a requestor may also neglect to close the iterator.

A provider cannot queue search results indefinitely. The provider must eventually release the resources that are associated with a search result. (Put differently, any iterator that a provider returns to a requestor must eventually expire.) Otherwise, the provider may run out of resources.

Providers should carefully manage the resources associated with search results. For example:

- A provider may define a *timeout interval* that specifies the maximum time between iterate requests. If a requestor does not request an iterate operation within this interval, the provider

will release the resources associated with the search result. This invalidates any iterator that represents this search result.

- A provider may also define an overall *result lifetime* that specifies the maximum length of time to retain a search result. After this amount of time has passed, the provider will release the search result.

- A provider may also wish to enforce an *overall limit* on the resources available to queue search results, and may wish to adjust its behavior (or even to refuse search requests) accordingly.

- To prevent denial of service attacks, the provider should not allocate any resource on behalf of a requestor until that requestor is properly authenticated.
  See the section titled "Security and Privacy Considerations".

3.6.7.1 search

The search operation obtains every object that matches a specified query.

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"/>
```

The <query> is the same type of element that is specified as part of a <bulkModifyRequest> or a <bulkDeleteRequest>. See the section titled "SearchQueryType".

If the search operation is successful *but selects no matching object*, the <searchResponse> will not contain a <pso>.

If the search operation is successful *and selects at least one matching object*, the <searchResponse> will contain any number of <pso> elements, each of which represents a matching object. If the search operation selects more matching objects than the <searchResponse> contains, the <searchResponse> will also contain an <iterator> that the requestor can use to retrieve more matching objects. (See the iterate operation below.)

If a search operation would select more objects than the provider can queue for subsequent iteration by the requestor, the provider's <searchResponse> will specify "error='resultSetTooLarge'".

Search is not batchable. For reasons of scale, neither a search request nor an iterate request should be nested in a batch request. When a search query matches more objects than the provider can place directly in the response, the provider must temporarily store the remaining objects. Storing the remaining objects allows the requestor to iterate the remaining objects, but also requires the provider to commit resources.
See the topic named “Resource Considerations” earlier in this section.

Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the results of asynchronous batch operations imposes on providers a resource burden similar to that of storing search results. Allowing a requestor to nest a search request within a batch request would aggravate the resource problem, requiring a provider to store more information in larger chunks for a longer amount of time.

3.6.7.1.1 searchRequest (normative)

A requestor MUST send a <searchRequest> to a provider in order to (ask the provider to) obtain every object that matches specified selection criteria.

Execution. A <searchRequest> MAY specify "executionMode".
See the section titled "Determining execution mode".

query. A <query> describes criteria that (the provider must use to) select objects on a target.
A <searchRequest> MAY contain at most one <query> element.

- If the provider's <listTargetsResponse> contains only a single <target>, then a <searchRequest> may omit the <query> element.
- If the provider's <listTargetsResponse> contains more than one <target>, then a <searchRequest> MUST contain exactly one <query> element and that <query> must specify "targetID".

See the section titled "SearchQueryType in a Request (normative)".

ReturnData. A <searchRequest> MAY have a "returnData" attribute that tells the provider which types of data to include in each selected object.

- A requestor that wants the provider to return *nothing* of the added object MUST specify "returnData='nothing'".
- A requestor that wants the provider to return *only the identifier* of the added object MUST specify "returnData='identifier'".
- A requestor that wants the provider to return the identifier of the added object *plus the XML representation of the object (as defined in the schema of the target)* MUST specify "returnData='data'".
- A requestor that wants the provider to return the identifier of the added object plus the XML representation of the object (as defined in the schema of the target) *plus any capability-specific data that is associated with the object* MAY specify "returnData='everything'" or MAY omit the "returnData" attribute (since "returnData='everything'" is the default).

maxSelect. A <searchRequest> MAY have a "maxSelect" attribute. The value of the "maxSelect" attribute specifies the maximum number of objects the provider should select.

IncludeDataForCapability. A <searchRequest> MAY contain any number of <includeDataForCapability> elements. Each <includeDataForCapability> element specifies a capability for which the provider should return capability-specific data (unless the "returnData" attribute specifies that the provider should return no capability-specific data at all).

- A requestor that wants the provider to return (as part of each object) capability-specific data *for only a certain set of capabilities* MUST enumerate that set of capabilities (by including an <includeDataForCapability> element that specifies each such capability) in the <searchRequest>.

- A requestor that wants the provider to return (as part of each object) capability-specific data *for all capabilities* MUST NOT include an <includeDataForCapability> element in the <searchRequest>.

- A requestor that wants the provider to return *no capability-specific data* MUST specify an appropriate value for the "returnData" attribute.
  See the topic named "ReturnData" immediately previous.

3.6.7.1.2 searchResponse (normative)

A provider that receives a <searchRequest> from a requestor that the provider trusts must examine the content of the <searchRequest>. If the request is valid, the provider MUST return (the XML that represents) every object that matches the specified <query> (if the provider can possibly do so). However, the number of objects selected (for immediate return or for eventual iteration) MUST NOT exceed any limit specified as "maxSelect" in the <searchRequest>.

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

A provider SHOULD execute a search operation synchronously if it is possible to do so. (The reason for this is that the result of a search should reflect the current state of each matching object. Other operations are more likely to intervene if a search operation is executed asynchronously.)

Response. The provider MUST return to the requestor a <searchResponse>.

Status. The <searchResponse> must contain a "status" attribute that indicates whether the provider successfully selected every object that matched the specified query.
See the section titled ""Status (normative)".

- If the provider successfully returned (the XML that represents) every object that matched the specified <query> up to any limit specified by the value of the "maxSelect" attribute, then the <searchResponse> MUST specify "status='success'".

- If the provider encountered an error in selecting any object that matched the specified <query> or (if the provider encountered an error) in returning (the XML that represents) any of the selected objects, then the <searchResponse> MUST specify "status='failure'".

PSO. The <searchResponse> MAY contain any number of <pso> elements.

- If the <searchResponse> specifies "status='success'" and *at least one object matched* the specified <query>, then the <searchResponse> MUST contain at least one <pso> element that contains (the XML representation of) a matching object.

- If the <searchResponse> specifies "status='success'" and *no object matched* the specified <query>, then the <searchResponse> MUST NOT contain a <pso> element.

- If the <searchResponse> specifies "status='failure'", then the <searchResponse> MUST NOT contain a <pso> element.

PSO and ReturnData. Each <pso> contains the subset of (the XML representation of) a requested object that the "returnData" attribute of the <searchRequest> specified. By default, each <pso> contains the entire (XML representation of an) object.

- A <pso> element MUST contain a <psoID> element.
  The <psoID> element MUST contain the identifier of the requested object.
  See the section titled "PSO Identifier (normative)".

- A <pso> element MAY contain a <data> element.
  - If the <searchRequest> specified "returnData='identifier'",
    then the <pso> MUST NOT contain a <data> element.
  - Otherwise, if the <searchRequest> specified "returnData='data'"
    or (if the <searchRequest> specified) "returnData='everything'"
    or (if the <searchRequest>) omitted the "returnData" attribute
    then the <data> element MUST contain the XML representation of the object.
    This XML must be valid according to the schema of the target for the schema entity of which the newly created object is an instance.

- A <pso> element MAY contain any number of <capabilityData> elements. Each <capabilityData> element contains a set of *capability-specific data* that is associated with the newly created object (for example, a *reference* to another object).
  - If the <searchRequest> specified "returnData='identifier'"
    or (if the <searchRequest> specified) "returnData='data'"
    then the <pso> MUST NOT contain a <capabilityData> element.
  - Otherwise, if the <searchRequest> specified "returnData='everything'"
    or (if the <searchRequest>) omitted the "returnData" attribute,
    then the <pso> MUST contain a <capabilityData> element for each set of capability-specific data that is associated with the requested object
    (and that is specific to a capability that the target supports for the schema entity of which the requested object is an instance).

PSO capabilityData and IncludeDataForCapability. A <searchResponse> MUST include (as <capabilityData> sub-elements of each <pso>) any set of capability-specific data that is associated with a matching object and for which *all* of the following are true:

- The <searchRequest> specifies "returnData='everything'" or (the <searchRequest>) omits the "returnData" attribute.

- The schema for the target declares that the *target supports the capability* (for the schema entity of which each matching object is an instance).

- The <searchRequest> contains an <includeDataForCapability> element that contains (as its string content) the URI of the capability to which the data are specific or the <searchRequest> contains no <includeDataForCapability> element.

A <searchResponse> SHOULD NOT include (as a <capabilityData> sub-element of each <pso>) any set of capability-specific data for which any of the above is not true.

iterator. A <searchResponse> MAY contain at most one <iterator> element.

- If the <searchResponse> specifies "status='success'" and the search response *contains all of the objects* that matched the specified <query>, then the <searchResponse> MUST NOT contain an <iterator>.

- If the <searchResponse> specifies "status='success'" and the search response *contains some but not all of the objects* that matched the specified <query>, then the <searchResponse> MUST contain exactly one <iterator>.

- If the <searchResponse> specifies "status='success'" and *no object matched* the specified <query>, then the <searchResponse> MUST NOT contain an <iterator>.

- If the <searchResponse> specifies "status='failure'", then the <searchResponse> MUST NOT contain an <iterator>.

iterator ID. An <iterator> MUST have an "ID" attribute.

The value of the "ID" attribute uniquely identifies the <iterator> within the namespace of the provider. The "ID" attribute allows the provider to map each <iterator> token to the result set of the requestor's <query> and (also allows the provider to map each <iterator> token) to any state that records the requestor's position within that result set.

The "ID" attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an <iterator>. An <iterator> is not a PSO.

Error. If the <searchResponse> specifies "status='failure'", then the <searchResponse> MUST have an "error" attribute that characterizes the failure.
See the general section titled ""Error (normative)".

The section titled "SearchQueryType Errors (normative)" describes errors specific to a request that contains a <query>. Also see the section titled "SelectionType Errors (normative)".
In addition, a <searchResponse> MUST specify an appropriate value of "error" if any of the following is true:

- If the *number of objects that matched* the <query> that was specified in a <searchRequest> *exceeds any limit on the part of the provider* (but does not exceed any value of "maxSelect" that the requestor specified as part of the <query>). In this case, the provider's <searchResponse> SHOULD specify "error='resultSetTooLarge'".

3.6.7.1.3 search Examples (non-normative)

In the following example, a requestor asks a provider to search for every Person with an email 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>
```

The provider returns a <searchResponse>. The "status" attribute of the <searchResponse> indicates that the provider successfully executed the search operation.

```
<searchResponse requestID="137" 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>
    <iterator ID="1826"/>
</searchResponse>
```

In the following example, a requestor asks a provider to search for every account that is currently owned by “joebob”. The requestor uses the “returnData” attribute to specify that the provider 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>
```

The provider returns a <searchResponse>. The “status” attribute of the <searchResponse> indicates that the provider successfully executed the search operation.

```
<searchResponse requestID=”138” status=“success" >
    <pso>
        <psoID ID=”1431” targetID=“target1”/>
    </pso>
</searchResponse>
```

3.6.7.2 iterate

The iterate operation obtains the next set of objects from the result set that the provider selected for a search operation. (See the description of the search operation above.)

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>

   <element name="iterateRequest" type="spmlsearch:IterateRequestType"/>
   <element name="iterateResponse" type="spmlsearch:SearchResponseType"/>
```

An iterateRequest receives an iterateResponse. A requestor supplies as input to an <iterateRequest> the <iterator> that was part of the original <searchResponse> or the <iterator> that was part of a subsequent <iterateResponse>, whichever is most recent. A provider returns an <iterateResponse> in response to each <iterateRequest>. An <iterateResponse> has the same structure as a <searchResponse>.

The <iterateResponse> will contain at least one <pso> element that represents a matching object. If more matching objects are available to return, then the <iterateResponse> will also contain an <iterator>. The requestor can use this <iterator> in another <iterateRequest> to retrieve more of the matching objects.

Iterate is not batchable. For reasons of scale, neither a search request nor an iterate request should be nested in a batch request. When a search query matches more objects than the provider can place directly in the response, the provider must temporarily store the remaining objects. Storing the remaining objects allows the requestor to iterate the remaining objects, but also requires the provider to commit resources.
See the topic named “Resource Considerations” earlier in this section.

Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the results of asynchronous batch operations imposes on providers a resource burden similar to that of search results. Allowing a requestor to nest a search request or an iterate request within a batch request would aggravate the resource problem, requiring a provider to store more information in larger chunks for a longer amount of time.

The iterate operation must be executed synchronously. The provider is already queuing the result set (every object beyond those returned in the first search response), so it is unreasonable for a requestor to ask the provider to queue the results of a request for the next item in the result set.

Furthermore, asynchronous iteration would complicate the provider’s maintenance of the result set. Since a provider could never know that the requestor had processed the results of an asynchronous iteration, the provider would not know when to increment its position in the result set. In order to support asynchronous iteration both correctly and generally, a provider would have to maintain a version of every result set *for each iteration* of that result set. This would impose an unreasonable burden on the provider.

3.6.7.2.1 iterateRequest (normative)

A requestor MUST send an <iterateRequest> to a provider in order to obtain any *additional* objects that matched a previous <searchRequest> but that the provider has not yet returned to the requestor. (That is, matching objects that were not contained in the response to that <searchRequest> and that have not yet been contained in any response to an <iterateRequest> associated with that <searchRequest>.)

Execution. An <iterateRequest> MUST NOT specify "executionMode='asynchronous'".
An <iterateRequest> MUST specify "executionMode='synchronous'"
or (an <iterateRequest> MUST) omit “executionMode”.
See the section titled "Determining execution mode".

iterator. An <iterateRequest> MUST contain exactly one <iterator> element. A requestor MUST supply as input to an <iterateRequest> the <iterator> from the original <searchResponse> or (the requestor MUST supply as input to the <iterateRequest>) the <iterator> from a subsequent <iterateResponse>. A requestor SHOULD supply as input to an <iterateRequest> the most recent <iterator> that represents the search result set.

3.6.7.2.2 iterateResponse (normative)

A provider that receives a <iterateRequest> from a requestor that the provider trusts must examine the content of the <iterateRequest>. If the request is valid, the provider MUST return (the XML that represents) the next set of objects from the result set that the <iterator> represents.

Execution. The provider MUST execute the iterate operation synchronously (if the provider executes the iterate operation at all). See the section titled "Determining execution mode".

Response. The provider MUST return to the requestor an <iterateResponse>.

Status. The <iterateResponse> must contain a "status" attribute that indicates whether the provider successfully returned the next set of objects from the result set that the <iterator> represents. See the section titled ""Status (normative)".

- If the provider successfully returned (the XML that represents) the next set of objects from the result set that the <iterator> represents, then the <iterateResponse> MUST specify "status='success'".
- If the provider encountered an error in returning (the XML that represents) the next set of objects from the result set that the <iterator> represents, then the <iterateResponse> MUST specify "status='failure'".

PSO. The <iterateResponse> MAY contain any number of <pso> elements.

- If the <iterateResponse> specifies "status='success'" and *at least one object remained to iterate* (in the result set that the <iterator> represents), then the <iterateResponse> MUST contain at least one <pso> element that contains the (XML representation of the) next matching object.
- If the <iterateResponse> specifies "status='success'" and *no object remained to iterate* (in the result set that the <iterator> represents), then the <iterateResponse> MUST NOT contain a <pso> element.
- If the <iterateResponse> specifies "status='failure'", then the <iterateResponse> MUST NOT contain a <pso> element.

PSO and ReturnData. Each <pso> contains the subset of (the XML representation of) a requested object that the "returnData" attribute of the original <searchRequest> specified. By default, each <pso> contains the entire (XML representation of an) object.

- A <pso> element MUST contain a <psoID> element.
  The <psoID> element MUST contain the identifier of the requested object.
  See the section titled "PSO Identifier (normative)".
- A <pso> element MAY contain a <data> element.
  - If the <searchRequest> specified "returnData='identifier'", then the <pso> MUST NOT contain a <data> element.

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

- A <pso> element MAY contain any number of <capabilityData> elements. Each <capabilityData> element contains a set of capability-*specific data* that is associated with the newly created object (for example, a *reference* to another object).
  - If the <searchRequest> specified "returnData='identifier'"
    or (if the <searchRequest> specified) "returnData='data'"
    then the <pso> MUST NOT contain a <capabilityData> element.
  - Otherwise, if the <searchRequest> specified "returnData='everything'"
    or (if the <searchRequest>) omitted the "returnData" attribute,
    then the <pso> MUST contain a <capabilityData> element for each set of capability-specific data that is associated with the requested object
    (and that is specific to a capability that the target supports for the schema entity of which the requested object is an instance).

PSO capabilityData and IncludeDataForCapability. An <iterateResponse> MUST include (as <capabilityData> sub-elements of each <pso>) any capability-specific data that is associated with each matching object and for which *all* of the following are true:

- The original <searchRequest> specified "returnData='everything'"
  or (the original <searchRequest>) omitted the "returnData" attribute.

- The schema for the target declares that the *target supports the capability*
  (for the schema entity of which each matching object is an instance).

- The original <searchRequest> contained an <includeDataForCapability> element that specified the capability to which the data are specific
  or the original <searchRequest> contained no <includeDataForCapability> element.

An <iterateResponse> SHOULD NOT include (as <capabilityData> sub-elements of each <pso>) any capability-specific data for which any of the above is not true.

iterator. A <iterateResponse> MAY contain at most one <iterator> element.

- If the <iterateResponse> specifies "status='success'" and the search response *contains the last of the objects* that matched the <query> that was specified in the original <searchRequest>, then the <iterateResponse> MUST NOT contain an <iterator>.

- If the <iterateResponse> specifies "status='success'" and the provider *still has more matching objects* that have not yet been returned to the requestor, then the <iterateResponse> MUST contain exactly one <iterator>.

- If the <iterateResponse> specifies "status='failure'", then the <iterateResponse> MUST NOT contain an <iterator>.

iterator ID. An <iterator> MUST have an "ID" attribute.

The value of the "ID" attribute uniquely identifies the <iterator> within the namespace of the provider. The "ID" attribute allows the provider to map each <iterator> token to the result set of the requestor's <query> and to any state that records the requestor's position within that result set.

The "ID" attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an <iterator>. An <iterator> is not a PSO.

Error. If the <iterateResponse> specifies "status='failure'", then the <iterateResponse> MUST have an "error" attribute that characterizes the failure. See the general section titled ""Error (normative)" .

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

- If the provider does not recognize the <iterator> in an <iterateRequest> as representing a result set.
- If the provider does not recognize the <iterator> in an <iterateRequest> as representing any result set that the provider currently maintains.

The <iterateResponse> MAY specify an appropriate value of "error" if any of the following is true:

- If an <iterateRequest> contains an <iterator> that is *not the most recent version* of the <iterator>. If the provider has returned to the requestor a more recent <iterator> that represents the same search result set, then the provider MAY reject the older <iterator>. (A provider that changes the ID—for example, to encode the state of iteration within a search result set—may be sensitive to this.)

3.6.7.2.3 iterate Examples (non-normative)

In order to illustrate the iterate operation, we first need a search operation that returns more than one object. In the following example, a requestor asks a provider to search for every Person with 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>
```

The provider returns a <searchResponse>. The "status" attribute of the <searchResponse> indicates that the provider successfully executed the search operation. The <searchResponse> contains two <pso> elements that represent the first matching objects.

```
<searchResponse requestID="147" 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>
```

The requestor asks the provider to return the next matching objects (in the result set for the search). The requestor supplies the <iterator> from the <searchResponse> as input to the <iterateRequest>.

```
<iterateRequest requestID="148" >
    <iterator ID="1900"/>
</iterateRequest>
```

The provider returns an <iterateResponse> in response to the <iterateRequest>. The "status" attribute of the <iterateResponse> indicates that the provider successfully executed the iterate operation. The <iterateResponse> contains two <pso> elements that represent the 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>
```

The <iterateResponse> also contains another <iterator> element. The "ID" of this <iterator> differs from the "ID" of the <iterator> in the original <searchResponse>. The "ID" could remain constant (for each iteration of the result set that the <iterator> represents) if the provider so chooses, but the "ID" value could change (e.g., if the provider uses "ID" to encode the state of the result set).

To get the final matching object, the requestor again supplies the <iterator> from the <iterateResponse> as input to the <iterateRequest>.

```
<iterateRequest requestID="149">
    <iterator ID="1901"/>
</iterateRequest>
```

The provider again returns an <iterateResponse> in response to the <iterateRequest>. The "status" attribute of the <iterateResponse> indicates that the provider successfully executed the iterate operation. The <iterateResponse> contains a <pso> element that represents the final matching object. Since all of the matching objects have now been returned to the requestor, 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>
        <psoID ID=”2244” targetID=“target2”/>
    </pso>
</iterateResponse>
```

3.6.7.3 closeIterator

The closeIterator operation tells the provider that the requestor has no further need for the search result that a specific <iterator> represents. (See the description of the search operation above.)

A requestor should send a <closeIteratorRequest> to the provider when the requestor no longer intends to iterate a search result. (A provider will eventually free an inactive search result -- even if the provider never receives a <closeIteratorRequest> from the requestor-- but this behavior is unspecified.) For more information, see the topic named "Resource Considerations" topic earlier within this section.

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"/>
```

A closeIteratorRequest receives a closeIteratorResponse. A requestor supplies as input to a <closeIteratorRequest> the <iterator> that was part of the original <searchResponse> or the <iterator> that was part of a subsequent <iterateResponse>, whichever is most recent. A provider returns a <closeIteratorResponse> in response to each <closeIteratorRequest>. A <closeIteratorResponse> has the same structure as an <spml:response>.

closeIterator is not batchable. For reasons of scale, neither of a search request nor an iterate request nor a closeIterator request should be nested in a batch request. When a search query

matches more objects than the provider can place directly in the response, the provider must temporarily store the remaining objects. Storing the remaining objects allows the requestor to iterate the remaining objects, but also requires the provider to commit resources. See the topic named “Resource Considerations” earlier in this section.

Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the results of asynchronous batch operations imposes on providers a resource burden similar to that of search results. Allowing a requestor to nest a search request or an iterate request or a closeIterator request within a batch request would aggravate the resource problem, requiring a provider to store more information in larger chunks for a longer amount of time.

The closeIterator operation must be executed synchronously. The provider is already queuing the result set (every object beyond those returned in the first search response), so a request to close the iterator (and thus to free the system resources associated with the result set) should be executed as soon as possible. It is unreasonable for a requestor to ask the provider to queue the results of a request to close an iterator (especially since the close iterator response contains little or no information beyond success or failure).

3.6.7.3.1 closeIteratorRequest (normative)

A requestor SHOULD send a `<closeIteratorRequest>` to a provider when the requestor no longer intends to iterate a search result. (This allows the provider to free any system resources associated with the search result.).

Execution. A `<closeIteratorRequest>` MUST NOT specify `"executionMode='asynchronous'"`.
A `<closeIteratorRequest>` MUST specify `"executionMode='synchronous'"` or (a `<closeIteratorRequest>` MUST) omit `"executionMode"`.
See the section titled "Determining execution mode".

iterator. A `<closeIteratorRequest>` MUST contain exactly one `<iterator>` element. A requestor MUST supply as input to a `<closeIteratorRequest>` the `<iterator>` from the original `<searchResponse>` or (a requestor MUST supply the `<iterator>` from a subsequent `<iterateResponse>`). A `requestor` SHOULD supply as input to a `<closeIteratorRequest>` the most recent `<iterator>` that represents the search result set.

iterator ID. An `<iterator>` that is part of a `<closeIteratorRequest>` MUST have an `"ID"` attribute. (The value of the `"ID"` attribute uniquely identifies the `<iterator>` within the namespace of the provider. The `"ID"` attribute allows the provider to map each `<iterator>` token to the result set of the requestor’s `<query>` and also (allows the provider to map each `<iterator>` token) to any state that records the requestor's iteration *within* that result set.)

3.6.7.3.2 closeIteratorResponse (normative)

A provider that receives a `<closeIteratorRequest>` from a requestor that the provider trusts must examine the content of the `<closeIteratorRequest>`. If the request is valid, the provider MUST release any search result set that the `<iterator>` represents. Any subsequent request to iterate that same search result set MUST fail.

Execution. The provider MUST execute the closeIterator operation synchronously (if the provider executes the closeIterator operation at all). See the section titled “Determining execution mode”.

Response. The provider MUST return to the requestor a `<closeIteratorResponse>`.

Status. The <closeIteratorResponse> must contain a "status" attribute that indicates whether the provider successfully released the search result set that the <iterator> represents. See the section titled "Status (normative)".

- If the provider successfully released the search result set that the <iterator> represents, then the <closeIteratorResponse> MUST specify "status='success'".
- If the provider encountered an error in releasing the search result set that the <iterator> represents, then the <closeIteratorResponse> MUST specify "status='failure'".

Error. If the <closeIteratorResponse> specifies "status='failure'", then the <closeIteratorResponse> MUST have an "error" attribute that characterizes the failure. See the general section titled "Error (normative)".

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

- If the provider does not recognize the <iterator> in a <closeIteratorRequest> as representing a search result set.
- If the provider does not recognize the <iterator> in a <closeIteratorRequest> as representing any search result set that the provider currently maintains.
- If the provider recognized the <iterator> in a <closeIteratorRequest> as representing a search result set that the provider currently maintains but *cannot release the resources associated with that search result set*.

The <closeIteratorResponse> MAY specify an appropriate value of "error" if any of the following is true:

- If a <closeIteratorRequest> contains an <iterator> that is *not the most recent version* of the <iterator>. If the provider has returned to the requestor a more recent <iterator> that represents the same search result set, then the provider MAY reject the older <iterator>.
  (A provider that changes the ID—for example, to encode the state of iteration within a search result set—may be sensitive to this.)

3.6.7.3.3 closeIterator Examples (non-normative)

In order to illustrate the closeIterator operation, we first need a search operation that returns more than one object. In the following example, a requestor asks a provider to search for every Person 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>
```

The provider returns a <searchResponse>. The "status" attribute of the <searchResponse> indicates that the provider successfully executed the search operation. The <searchResponse> 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>
```

The requestor decides that the two objects in the initial `<searchResponse>` will suffice, and does not intend to retrieve any more matching objects (in the result set for the search). The requestor supplies the `<iterator>` from the `<searchResponse>` as input to the `<closeIteratorRequest>`.

```
<closeIteratorRequest requestID=”151”>
    <iterator ID=”1900”/>
</closeIteratorRequest>
```

The provider returns a `<closeIteratorResponse>` in response to the `<closeIteratorRequest>`. The `"status"` attribute of the `<closeIteratorResponse>` indicates that the provider successfully released the result set.

```
<closeIteratorResponse requestID=”151” status=“success"/>
```

3.6.8 Suspend Capability

The Suspend Capability is defined in a schema associated with the following XML namespace: `urn:oasis:names:tc:SPML:2:0:suspend`. This document includes the Suspend Capability XSD as Appendix H.

The Suspend Capability defines three operations: suspend, resume and active.

- The suspend operation *disables an object* (immediately or on a specified date).
- The resume operation *re-enables an object* (immediately or on a specified date).
- The active operation *tests whether an object is currently suspended*.

The suspend operation disables an object *persistently* (rather than transiently). The suspend operation is intended to revoke the privileges of an account, for example, while the authorized user of the account is on vacation.

The resume operation re-enables an object persistently. One might use the resume operation to restore privileges for an account, for example, when the authorized user of the account returns from vacation.

A provider that supports the suspend, resume and active operations for a target SHOULD declare that the target supports the Suspend Capability. A provider that does not support all of suspend, resume and active MUST NOT declare that the target supports the Suspend Capability.

Idempotent. The suspend operation and the resume operation are both *idempotent*. Any requestor should be able to suspend (or to resume) the same object multiple times without error.

Search. A requestor can *search for objects based on enabled state* using the `<isActive>` query clause. The `{IsActiveType}` extends `{QueryClauseType}`, which indicates that an instance of `{IsActiveType}` can be used to select objects. An `<isActive>` clause matches an object if and only if the object is currently enabled. In order to select disabled objects, a requestor would combine this clause with the logical operator `<not>`. See the section titled "Selection".

3.6.8.1 suspend

The suspend operation enables a requestor to disable an object.

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="spmlsuspend:SuspendRequestType"/>
   <element name="suspendResponse" type="spml:ResponseType"/>
```

3.6.8.1.1 suspendRequest (normative)

A requestor MUST send a <suspendRequest> to a provider in order to (ask the provider to) disable an existing object.

Execution. A <suspendRequest> MAY specify "executionMode".
See the section titled "Determining execution mode.

psoID. A <suspendRequest> MUST contain exactly one <psoID> element. A <psoID> element MUST identify an object that exists on a target that is exposed by the provider.
See the section titled "PSO Identifier (normative)".

EffectiveDate. A <suspendRequest> MAY specify an "effectiveDate". Any "effectiveDate" value MUST be expressed in UTC form, with no time zone component.
A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.
A requestor MUST NOT generate time instants that specify leap seconds.

3.6.8.1.2 suspendResponse (normative)

A provider that receives a <suspendRequest> from a requestor that the provider trusts MUST examine the content of the <suspendRequest>. If the request is valid and if the specified object exists, then the provider MUST disable the object that the <psoID> specifies.

If the <suspendRequest> specifies an "effectiveDate", the provider MUST enable the specified object as of that date.

- If the "effectiveDate" of the <suspendRequest> is in the past, then the provider MUST do one of the following:
  - The provider MAY disable the specified object *immediately*.
  - The provider MAY return an error. (The provider's response SHOULD indicate that the request failed because the effective date is past.)
- If the "effectiveDate" of the <suspendRequest> is in the future, then
  - The provider MUST NOT disable the specified object until that future date and time.
  - The provider MUST disable the specified object at that future date and time (unless a subsequent request countermands this request).

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

Response. The provider must return to the requestor a <suspendResponse>. The <suspendResponse> must have a "status" attribute that indicates whether the provider successfully disabled the specified object. See the section titled "Status (normative)".

Error. If the provider cannot create the requested object, the <suspendResponse> must contain an error attribute that characterizes the failure. See the general section titled "Error (normative)".

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

- The <suspendRequest> contains a <psoID> for an object that does not exist.
- The <suspendRequest> specifies an "effectiveDate" that is not valid.

The provider MAY return an error if any of the following is true:

- The <suspendRequest> specifies an "effectiveDate" that is in the past.

The provider MUST NOT return an error when (the operation would otherwise succeed and) the object is already disabled. In this case, the <suspendResponse> MUST specify "status='success'".

3.6.8.1.3 suspend Examples (non-normative)

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

```
<suspendRequest requestID="139">
    <psoID ID="2244" targetID="target2"/>
</suspendRequest>
```

The provider returns an <suspendResponse> element. The "status" attribute of the <suspendResponse> indicates that the provider successfully disabled the specified object.

```
<suspendResponse requestID="139" status="success" />
```

In the following example, a requestor asks a provider to suspend an existing account.

```
<suspendRequest requestID="140" >
    <psoID ID="1431" targetID="target1"/>
</suspendRequest>
```

The provider returns a <suspendResponse>. The "status" attribute of the <suspendResponse> indicates that the provider successfully disabled the specified account.

```
<suspendResponse requestID="140" status="success"/>
```

3.6.8.2 resume

The resume operation enables a requestor to re-enable an object that has been suspended. (See the description of the suspend operation above.)

The subset of the Suspend Capability XSD that is most relevant to the resume operation follows.

```
   <complexType name="ResumeRequestType">
      <complexContent>
         <extension base="spml:RequestType">
            <sequence>
               <element name="psoID" type="spml:PSOIdentifierType"/>
            </sequence>
            <attribute name="effectiveDate" type="dateTime"
use="optional"/>
         </extension>
      </complexContent>
   </complexType>

   <element name="ResumeRequest" type="spmlsuspend:ResumeRequestType"/>
   <element name="ResumeResponse" type="spml:ResponseType"/>
```

3.6.8.2.1 resumeRequest (normative)

A requestor MUST send a <resumeRequest> to a provider in order to (ask the provider to) re-enable an existing object.

Execution. A <resumeRequest> MAY specify "executionMode".
See the section titled "Determining execution mode".

psoID. A <resumeRequest> MUST contain exactly one <psoID> element. A <psoID> element MUST identify an object that exists on a target (that is supported by the provider).
See the section titled "PSO Identifier (normative)".

EffectiveDate. A <resumeRequest> MAY specify an "effectiveDate". Any "effectiveDate" value MUST be expressed in UTC form, with no time zone component.
A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.
A requestor MUST NOT generate time instants that specify leap seconds.

3.6.8.2.2 resumeResponse (normative)

A provider that receives a <resumeRequest> from a requestor that the provider trusts MUST examine the content of the <resumeRequest>. If the request is valid and if the specified object exists, then the provider MUST enable the object that is specified by the <psoID>.

If the <resumeRequest> specifies an "effectiveDate", the provider MUST enable the specified object as of that date.

- If the "effectiveDate" of the <resumeRequest> is in the past, then the provider MUST do one of the following:
  - The provider MAY enable the specified object *immediately*.
  - The provider MAY return an error. (The provider's response SHOULD indicate that the request failed because the effective date is past.)
- If the "effectiveDate" of the <resumeRequest> is in the future, then
  - The provider MUST NOT enable the specified object until that future date and time.
  - The provider MUST enable the specified object at that future date and time (unless a subsequent request countermands this request).

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

Response. The provider must return to the requestor a <resumeResponse>. The <resumeResponse> must have a "status" attribute that indicates whether the provider successfully enabled the specified object. See the section titled "Status (normative)".

Error. If the provider cannot enable the requested object, the <resumeResponse> must contain an error attribute that characterizes the failure. See the general section titled "Error (normative)".

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

- The <resumeRequest> contains a <psoID> for an object that does not exist.
- The <resumeRequest> specifies an "effectiveDate" that is not valid.

The provider MAY return an error if any of the following is true:

- The <resumeRequest> specifies an "effectiveDate" that is in the past.

The provider MUST NOT return an error when (the operation would otherwise succeed and) the object is already enabled. In this case, the response should specify "status='success'".

3.6.8.2.3 *resume Examples (non-normative)*

In the following example, a requestor asks a provider to resume an existing `Person` object.

```
<resumeRequest requestID="141">
    <psoID ID="2244" targetID="target2"/>
</resumeRequest>
```

The provider returns a `<resumeResponse>` element. The `"status"` attribute of the `<resumeResponse>` element indicates that the provider successfully disabled the specified object.

```
<resumeResponse requestID="141" status="success"/>
```

In the following example, a requestor asks a provider to resume an existing account.

```
<resumeRequest requestID="142">
    <psoID ID="1431" targetID="target1"/>
</resumeRequest>
```

The provider returns a `<resumeResponse>`. The `"status"` attribute of the `<resumeResponse>` indicates that the provider successfully enabled the specified account.

```
<resumeResponse requestID="142" status="success"/>
```

3.6.8.3 active

The active operation enables a requestor to determine whether a specified object has been suspended. (See the description of the suspend operation above.)

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"/>
```

3.6.8.3.1 *activeRequest (normative)*

A requestor MUST send an `<activeRequest>` to a provider in order to (ask the provider to) determine whether the specified object is enabled (active) or disabled.

Execution. An <activeRequest> MAY specify "executionMode".
See the section titled "Determining execution mode.

psoID. A <activeRequest> MUST contain exactly one <psoID> element. A <psoID> element MUST identify an object that exists on a target that is exposed by the provider.
See the section titled "PSO Identifier (normative)".

3.6.8.3.2 activeResponse (normative)

A provider that receives a <activeRequest> from a requestor that the provider trusts MUST examine the content of the <activeRequest>. If the request is valid and if the specified object exists, then the provider MUST disable the object that is specified by the <psoID>.

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

Response. The provider must return to the requestor an <activeResponse>. The <activeResponse> must have a "status" attribute that indicates whether the provider successfully determined whether the specified object is enabled (i.e. active).
See the section titled "Status (normative)".

active. An <activeResponse> MAY have an "active" attribute that indicates whether the specified object is suspended. An <activeResponse> that specifies "status='success'" MUST have an "active" attribute.

- If the specified object is suspended, the <activeResponse> MUST specify "active='false'".
- If the specified object is not suspended, the <activeResponse> MUST specify "active='true'".

Error. If the provider cannot determine whether the requested object is suspended, the <activeResponse> must contain an "error" attribute that characterizes the failure.
See the general section titled "Error (normative)".

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

- The <activeRequest> contains a <psoID> that specifies an object that does not exist.

3.6.8.3.3 active Examples (non-normative)

In the following example, a requestor asks a provider whether a Person object is active.

```
<activeRequest requestID="143">
    <psoID ID="2244" targetID="target2"/>
</activeRequest>
```

The provider returns an <activeResponse> element. The "status" attribute of the <activeResponse> element indicates that the provider successfully completed the requested operation. The "active" attribute of the <activeResponse> indicates that the specified object is active.

```
<activeResponse requestID="143" status="success" active="true"/>
```

In the following example, a requestor asks a provider whether an account is active.

```
<activeRequest requestID="144">
    <psoID ID="1431" targetID="target1"/>
</activeRequest>
```

The provider returns an `<activeResponse>`. The `"status"` attribute of the `<activeResponse>` indicates that the provider successfully completed the requested operation. The "`active`" attribute of the `<activeResponse>` indicates that the specified object is active.

```
<activeResponse requestID="144" status="success" active="true"/>
```

3.6.9 Updates Capability

The Updates Capability is defined in a schema associated with the following XML namespace: `urn:oasis:names:tc:SPML:2:0:updates`. This document includes the Updates Capability XSD as Appendix I.

The Updates Capability defines three operations: updates, iterate and closeIterator. The updates and iterate operations together allow a requestor to obtain *in a scalable manner* every recorded *update* (i.e., modification to an object) that matches specified selection criteria. The updates operation returns in its response a first set of matching updates. Each subsequent iterate operation returns more matching updates. The closeIterator operation allows a requestor to tell a provider that it does not intend to finish iterating a result set and that the provider may therefore release the associated resources).

A provider that supports the updates and iterate operations for a target SHOULD declare that the target supports the Updates Capability. A provider that does not support both updates and iterate MUST NOT declare that the target supports the Updates Capability.

Resource considerations. A provider must limit the size and duration of its updates result sets (or that provider will exhaust available resources). A provider must decide:

- How large of an updates result set the provider will *select* on behalf of a requestor.
- How large of an updates result set the provider will *queue* on behalf of a requestor (so that the requestor may iterate the updates result set).
- For *how long a time* the provider will queue an updates result set on behalf of a requestor.

These decisions may be governed by the provider's implementation, by its configuration, or by runtime computation.

A provider that wishes to *never to queue updates result sets* may return every matching object (up to the provider's limit and up to any limit that the request specifies) in the updates response. Such a provider would never return an iterator, and would not need to support the iterate operation. The disadvantage is that, without an iterate operation, a provider's updates capability either is limited to small results or produces large updates responses.

A provider that wishes to support the iterate operation must store (or somehow queue) the updates selected by an updates operation until the requestor has a chance to iterate those results. (That is, a provider must somehow queue the updates that matched the criteria of an updates operation and that were not returned in the updates response.)

If all goes well, the requestor will continue to iterate the updates result set until the provider has sent all of the updates to the requestor. The requestor may also use the closeIterator operation to tell the provider that the requestor is no longer interested in the search result. Once all of the updates have been sent to the requestor, the provider may free any resource that is still associated with the updates result set. However, it is possible that the requestor may not iterate the updates result set in a timely manner--or that the requestor may *never* iterate the updates result set completely. Such a requestor may also neglect to close the iterator.

A provider cannot queue updates result sets indefinitely. The provider must eventually release the resources associated with an updates result set. (Put differently, any iterator that a provider returns to a requestor must eventually expire.) Otherwise, the provider may run out of resources.

Providers should carefully manage the resources associated with updates result sets. For example:

- A provider may define a *timeout interval* that specifies the maximum time between iterate requests. If a requestor does not request an iterate operation within this interval, the provider will release the resources associated with the result set. This invalidates any iterator that represents this result set.

- A provider may also define an overall *result lifetime* that specifies the maximum length of time to retain a result set. After this amount of time has passed, the provider will release the result set.

- A provider may also wish to enforce an *overall limit* on the resources available to queue result sets, and may wish to adjust its behavior (or even to refuse updates requests) accordingly.

- To prevent denial of service attacks, the provider should not allocate any resource on behalf of a requestor until that requestor is properly authenticated.
  See the section titled "Security and Privacy Considerations".

3.6.9.1 updates

The updates operation obtains *records of changes to objects*. A requestor may select change records based on changed-related criteria and (may also select change records) based on the set of objects.

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>
```

```
            <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"/>
```

The <query> is the same type of element that is specified as part of a <bulkModifyRequest> or a <bulkDeleteRequest> or a <searchRequest>. This <query> selects the objects for which the provider will return recorded updates. See the section titled "SearchQueryType".

The "updatedSince" attribute allows the requestor to select only updates that occurred since a specific date and time.

If the updates operation is successful *but selects no matching update*, the <updatesResponse> will not contain an <update>.

If the updates operation is successful *and selects at least one matching update*, the <updatesResponse> will contain any number of <update> elements, each of which represents a matching update. If the updates operation selects more matching updates than the <updatesResponse> contains, the <updatesResponse> will also contain an <iterator> that the requestor can use to retrieve more matching updates. (See the description of the iterate operation below.)

If an updates operation would select more updates than the provider can queue for subsequent iteration by the requestor, the provider's <updatesResponse> will specify "error='resultSetTooLarge'".

Updates is not batchable. For reasons of scale, neither an updates request nor an iterate request should be nested in a batch request. When an updates query matches more updates than the provider can place directly in the response, the provider must temporarily store the remaining updates. Storing the remaining updates allows the requestor to iterate the remaining updates, but also requires the provider to commit resources.
See the topic named "Resource Considerations" earlier in this section.

Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the results of asynchronous batch operations imposes on providers a resource burden similar to that of updates result sets. Allowing a requestor to nest an updates request within a batch request would aggravate the resource problem, requiring a provider to store more information in larger chunks for a longer amount of time.

3.6.9.1.1 updatesRequest (normative)

A requestor MUST send an `<updatesRequest>` to a provider in order to (ask the provider to) obtain every update that matches specified selection criteria.

Execution. An `<updatesRequest>` MAY specify "`executionMode`".
See the section titled "Determining execution mode".

query. A `<query>` describes criteria that (the provider must use to) select objects on a target. The provider will return only updates that affect objects that match these criteria.
An `<updatesRequest>` MAY contain at most one `<query>` element.

- If the provider's `<listTargetsResponse>` contains only a single `<target>`, then an `<updatesRequest>` may omit the `<query>` element.
- If the provider's `<listTargetsResponse>` contains more than one `<target>`, then an `<updatesRequest>` MUST contain exactly one `<query>` element and that `<query>` must specify "`targetID`".

See the section titled "SearchQueryType in a Request (normative)".

updatedByCapability. An `<updatesRequest>` MAY contain any number of `<updatedByCapability>` elements. Each `<updatedByCapability>` element contains the URN of an XML namespace that uniquely identifies a capability. Each `<updatedByCapability>` element must identify a capability that the target supports.

- A requestor that wants the provider to return no update that reflects a change to capability-specific data associated with an object MUST NOT place an `<updatedByCapability>` element in its `<updatesRequest>`.
- A requestor that wants the provider to return updates that reflect changes to capability-specific data associated with one or more objects MUST specify each capability (for which the provider should return updates) as an `<updatedByCapability>` element in its `<updatesRequest>`.

updatedSince. A `<updatesRequest>` MAY have an "`updatedSince`" attribute. (The provider will return only updates with a timestamp greater than this value.)

Any "`updatedSince`" value MUST be expressed in UTC form, with no time zone component.
A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.
A requestor MUST NOT generate time instants that specify leap seconds.

maxSelect. An `<updatesRequest>` MAY have a "`maxSelect`" attribute. The value of the "`maxSelect`" attribute specifies the maximum number of updates the provider should select.

token. An <updatesRequest> MAY have a "token" attribute. Any "token" value MUST match a value that the provider returned to the requestor as the value of the "token" attribute in a previous <updatesResponse> for the same target. Any "token" value SHOULD match the (value of the "token" attribute in the) provider's *most recent* <updatesResponse> for the same target.

3.6.9.1.2 updatesResponse (normative)

A provider that receives an <updatesRequest> from a requestor that the provider trusts must examine the content of the <updatesRequest>. If the request is valid, the provider MUST return updates that represent every change (that occurred since any time specified as "updatedSince") to every object that matches the specified <query> (if the provider can possibly do so). However, the number of updates selected (for immediate return or for eventual iteration) MUST NOT exceed any limit specified as "maxSelect" in the <updatesRequest>.

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

A provider SHOULD execute an updates operation synchronously if it is possible to do so. (The reason for this is that the result of an updates should reflect the set of changes currently recorded for each matching object. Other operations are more likely to intervene if an updates operation is executed asynchronously.)

Response. The provider MUST return to the requestor a <updatesResponse>.

Status. The <updatesResponse> must contain a "status" attribute that indicates whether the provider successfully selected every object that matched the specified query. See the section titled "Status (normative)" for values of this attribute.

- If the provider successfully returned every update that occurred (since any time specified by "updatedSince") to every object that matched the specified <query> up to any limit specified by the value of the "maxSelect" attribute, then the <updatesResponse> MUST specify "status='success'".

- If the provider encountered an error in selecting any object that matched the specified <query> or (if the provider encountered an error) in returning any of the selected updates, then the <updatesResponse> MUST specify "status='failure'".

Update. The <updatesResponse> MAY contain any number of <update> elements.

- If the <updatesResponse> specifies "status='success'" and *at least one update matched* the specified criteria, then the <updatesResponse> MUST contain at least one <update> element that describes a change to a matching object.

- If the <updatesResponse> specifies "status='success'" and *no object matched* the specified criteria, then the <updatesResponse> MUST NOT contain an <update> element.

- If the <updatesResponse> specifies "status='failure'", then the <updatesResponse> MUST NOT contain an <update> element.

Update PsoID. Each <update> MUST contain exactly one <psoID> element. Each <psoID> element uniquely identifies the object that was changed.

Update timestamp. Each <update> must have a "timestamp" attribute that specifies when the object was changed.

Any "timestamp" value MUST be expressed in UTC form, with no time zone component. A requestor or a provider SHOULD NOT rely on time resolution finer than milliseconds.

Update updateKind. Each <update> must have an "updateKind" attribute that describes how the object was changed.

- If the <update> specifies "updateKind='add'", then the object was added.
- If the <update> specifies "updateKind='modify'", then the (schema-defined XML data that represents the) object was modified.
- If the <update> specifies "updateKind='delete'", then the object was deleted.
- If the <update> specifies "updateKind='capability'", then a set of capability-specific data that is (or was) associated with the object was modified.

Update wasUpdatedByCapability. Each <update> MAY have a "wasUpdatedByCapability" attribute that identifies the capability for which data (specific to that capability and associated with the object) was changed.

- An <update> that specifies "updateKind='capability'" MUST have a "wasUpdatedByCapability" attribute.
- An <update> that specifies "updateKind='add'" or (that specifies) "updateKind='modify'" or (that specifies) "updateKind='delete'" MUST NOT have a "wasUpdatedByCapability" attribute.
- The value of each "wasUpdatedByCapability" MUST be the URN of an XML namespace that uniquely identifies a capability. Each "wasUpdatedByCapability" attribute MUST identify a capability that the target supports.

iterator. A <updatesResponse> MAY contain at most one <iterator> element.

- If the <updatesResponse> specifies "status='success'" and the updates response *contains all of the objects* that matched the specified <query>, then the <updatesResponse> MUST NOT contain an <iterator>.
- If the <updatesResponse> specifies "status='success'" and the updates response *contains some but not all of the objects* that matched the specified <query>, then the <updatesResponse> MUST contain exactly one <iterator>.
- If the <updatesResponse> specifies "status='success'" and *no object matched* the specified <query>, then the <updatesResponse> MUST NOT contain an <iterator>.
- If the <updatesResponse> specifies "status='failure'", then the <updatesResponse> MUST NOT contain an <iterator>.

iterator ID. An <iterator> MUST have an "ID" attribute.

The value of the "ID" attribute uniquely identifies the <iterator> within the namespace of the provider. The "ID" attribute allows the provider to map each <iterator> token to the result set of the requestor's <query> and to any state that records the requestor's position within that result set.

The "ID" attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an <iterator>. An <iterator> is not a PSO.

token. An <updatesResponse> MAY have a "token" attribute. (The requestor may pass this "token" value in the next <updatesRequest> for the same target. See the topic named "token" within the section titled "UpdatesRequest" above.)

Error. If the <updatesResponse> specifies "status='failure'", then the <updatesResponse> MUST have an "error" attribute that characterizes the failure. See the general section titled "Error (normative)".

The section titled "SearchQueryType Errors (normative)" describes errors specific to a request that contains a <query>. Also see the section titled "SelectionType Errors (normative)". In addition, the <updatesResponse> MUST specify an appropriate value of "error" if any of the following is true:

- If the *number of updates that matched* the criteria that were specified in an <updatesRequest> *exceeds any limit on the part of the provider*. (but does not exceed any value of "maxSelect" that the requestor specified as part of the <query>). In this case, the provider's <updatesResponse> SHOULD specify "error='resultSetTooLarge'".

3.6.9.1.3 updates Examples (non-normative)

In the following example, a requestor asks a provider to updates for every Person with an email address matching "joebob@example.com". The requestor includes no <updatedByCapability> element, which indicates that only updates to the schema-defined data for each matching object 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>
```

The provider returns a <updatesResponse>. The "status" attribute of the <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>
```

The requestor next asks the provider to include capability-specific updates (i.e., recorded changes to capability-specific data items that are associated with each matching object). The requestor indicates interest in updates specific to the reference capability and (indicates interest in updates 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>
```

The provider returns a <updatesResponse>. The "status" attribute of the <updatesResponse> indicates that the provider successfully executed the updates operation.

```
<updatesResponse requestID=”146” status=“success”>
    <update timestamp="20050704115911" updateKind="modify">
        <psoID ID=”2244” targetID=“target2”/>
    </update>
    <update timestamp="20050704115923" updateKind="capability"
wasUpdatedByCapability="urn:oasis:names:tc:SPML:2.0:reference">
        <psoID 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="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>

   <element name="iterateRequest" type="spmlupdates:IterateRequestType"/>
   <element name="iterateResponse"
type="spmlupdates:UpdatesResponseType"/>
```

An iterateRequest receives an iterateResponse. A requestor supplies as input to an <iterateRequest> the <iterator> that was part of the original <updatesResponse> or the <iterator> that was part of a subsequent <iterateResponse>, whichever is most recent. A provider returns an <iterateResponse> in response to each <iterateRequest>. An <iterateResponse> has the same structure as a <updatesResponse>.

The <iterateResponse> will contain at least one <update> element that records a change to an object. If more matching updates are available to return, then the <iterateResponse> will also contain an <iterator>. The requestor can use this <iterator> in another <iterateRequest> to retrieve more of the matching objects.

Iterate is not batchable. For reasons of scale, neither an updates request nor an iterate request should be nested in a batch request. When an updates query matches more updates than the provider can place directly in the response, the provider must temporarily store the remaining updates. Storing the remaining updates allows the requestor to iterate the remaining updates, but also requires the provider to commit resources.
See the topic named “Resource Considerations” earlier in this Updates Capability section.

Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the results of asynchronous batch operations imposes on providers a resource burden similar to that of updates result sets. Allowing a requestor to nest a updates request or an iterate request within a batch request would aggravate the resource problem, requiring a provider to store more information in larger chunks for a longer amount of time.

The iterate operation must be executed synchronously. The provider is already queuing the result set (every update beyond those returned in the first updates response), so it is unreasonable for a requestor to ask the provider to queue the results of a request for the next item in the result set.

Furthermore, asynchronous iteration would complicate the provider's maintenance of the result set. Since a provider could never know that the requestor had processed the results of an asynchronous iteration, the provider would not know when to increment its position in the result set. In order to support asynchronous iteration both correctly and generally, a provider would have to maintain a version of every result set for each iteration of that result set. This would impose an unreasonable burden on the provider.

3.6.9.2.1 iterateRequest (normative)

A requestor MUST send an `<iterateRequest>` to a provider in order to obtain any *additional* objects that matched a previous `<updatesRequest>` but that the provider has not yet returned to the requestor. (That is, matching objects that were not contained in the response to that `<updatesRequest>` and that have not yet been contained in any response to an `<iterateRequest>` associated with that `<updatesRequest>`.)

Execution. An `<iterateRequest>` MUST NOT specify `"executionMode='asynchronous'"`. An `<iterateRequest>` MUST specify `"executionMode='synchronous'"` or (an `<iterateRequest>` MUST) omit `"executionMode"`.
See the section titled "Determining execution mode".

iterator. An `<iterateRequest>` MUST contain exactly one `<iterator>` element. A requestor MUST supply as input to an `<iterateRequest>` the `<iterator>` from the original `<searchResponse>` or (the requestor MUST supply as input to the `<iterateRequest>`) the `<iterator>` from a subsequent `<iterateResponse>`. A `requestor` SHOULD supply as input to an `<iterateRequest>` the most recent `<iterator>` that represents the updates result set.

3.6.9.2.2 iterateResponse (normative)

A provider that receives a `<iterateRequest>` from a requestor that the provider trusts must examine the content of the `<iterateRequest>`. If the request is valid, the provider MUST return (the XML that represents) the next object in the result set that the `<iterator>` represents.

Execution. The provider MUST execute the iterate operation synchronously (if the provider executes the iterate operation at all). See the section titled "Determining execution mode".

Response. The provider MUST return to the requestor an `<iterateResponse>`.

Status. The `<iterateResponse>` must contain a `"status"` attribute that indicates whether the provider successfully returned the next update from the result set that the `<iterator>` represents. See the section titled "Status (normative)".

- If the provider successfully returned (the XML that represents) the next update from the result set that the `<iterator>` represents, then the `<iterateResponse>` MUST specify "`status='success'`".

- If the provider encountered an error in returning (the XML that represents) the next update from the result set that the `<iterator>` represents, then the `<iterateResponse>` MUST specify "`status='failure'`".

Update. The `<iterateResponse>` MAY contain any number of `<update>` elements.

- If the `<iterateResponse>` specifies "`status='success'`" and *at least one update remained to iterate* (in the updates result set that the `<iterator>` represents), then the `<iterateResponse>` MUST contain at least one `<update>` element that records a change to an object.

- If the <iterateResponse> specifies "status='success'" and *no update remained to iterate* (in the updates result set that the <iterator> represents), then the <iterateResponse> MUST NOT contain an <update> element.

- If the <iterateResponse> specifies "status='failure'", then the <iterateResponse> MUST NOT contain an <update> element.

iterator. A <iterateResponse> to an <iterateRequest> MAY contain at most one <iterator> element.

- If the <iterateResponse> specifies "status='success'" and the <iterateResponse> *contains the last of the updates* that matched the criteria that the original <updatesRequest> specified, then the <updatesResponse> MUST NOT contain an <iterator>.

- If the <iterateResponse> specifies "status='success'" and the provider *still has more matching updates* that have not yet been returned to the requestor, then the <iterateResponse> MUST contain exactly one <iterator>.

- If the <iterateResponse> specifies "status='failure'", then the <iterateResponse> MUST NOT contain an <iterator>.

iterator ID. An <iterator> MUST have an "ID" attribute.

The value of the "ID" attribute uniquely identifies the <iterator> within the namespace of the provider. The "ID" attribute allows the provider to map each <iterator> token to the result set of the requestor's <query> and to any state that records the requestor's position within that result set.

The "ID" attribute is (intended to be) *opaque to the requestor*. A requestor cannot lookup an <iterator>. An <iterator> is not a PSO.

Error. If the <iterateResponse> specifies "status='failure'", then the <iterateResponse> MUST have an "error" attribute that characterizes the failure. See the general section titled ""Error (normative)".

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

- The provider does not recognize the <iterator> in an <iterateRequest> as representing an updates result set.

- The provider does not recognize the <iterator> in an <iterateRequest> as representing any updates result set that the provider currently maintains.

The <iterateResponse> MAY specify an appropriate value of "error" if any of the following is true:

- An <iterateRequest> contains an <iterator> that is *not the most recent version* of the <iterator>. If the provider has returned to the requestor a more recent <iterator> that represents the same updates result set, then the provider MAY reject the older <iterator>. (A provider that changes the ID—for example, to encode the state of iteration within an updates result set—may be sensitive to this.)

3.6.9.2.3 iterate Examples (non-normative)

In order to illustrate the iterate operation, we first need an updates operation that returns more than one update. In the following example, a requestor asks a provider to return updates for every `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>
```

The provider returns a `<updatesResponse>`. The `"status"` attribute of the `<updatesResponse>` indicates that the provider successfully executed the updates operation. The `<updatesResponse>` contains two `<update>` elements that represent the first matching updates.

```
<updatesResponse requestID="152" status="success">
    <update timestamp="1944062400000000" updateKind="add">
        <psoID ID="0001" targetID="target2"/>
    </update>
    <update timestamp="1942092700000000" updateKind="add">
        <psoID ID="0002" targetID="target2"/>
    </update>
    <update timestamp="1970091800000000" updateKind="delete">
        <psoID ID="0002" targetID="target2"/>
    </update>
    <iterator ID="1970"/>
</updatesResponse>
```

The requestor asks the provider to return the next set of matching updates (from the original result set). The requestor supplies the `<iterator>` from the `<updatesResponse>` as input to the `<iterateRequest>`.

```
<iterateRequest requestID="153">
    <iterator ID="1970"/>
</iterateRequest>
```

The provider returns an `<iterateResponse>` in response to the `<iterateRequest>`. The `"status"` attribute of the `<iterateResponse>` indicates that the provider successfully executed the iterate operation. The `<iterateResponse>` contains two `<update>` elements that represent the next matching updates.

```
<iterateResponse requestID="153" status="success">
    <update timestamp="1948031200000000" updateKind="add">
        <psoID ID="0003" targetID="target2"/>
    </update>
    <update timestamp="1969120900000000" updateKind="add">
        <psoID ID="0004" targetID="target2"/>
    </update>
    <iterator ID="1971"/>
</iterateResponse>
```

The `<iterateResponse>` also contains another `<iterator>` element. The `"ID"` of this `<iterator>` differs from the `"ID"` of the `<iterator>` in the original `<updatesResponse>`. The `"ID"` could remain constant (for each iteration of the result set that the `<iterator>` represents) if the provider so chooses, but the `"ID"` value could change (e.g., if the provider uses `"ID"` to encode the state of the result set).

To get the next set of matching updates, the requestor again supplies the <iterator> from the <iterateResponse> as input to an <iterateRequest>.

```
<iterateRequest requestID="154">
    <iterator ID="1971"/>
</iterateRequest>
```

The provider again returns an <iterateResponse> in response to the <iterateRequest>. The "status" attribute of the <iterateResponse> indicates that the provider successfully executed the iterate operation. The <iterateResponse> contains an <update> element that represents the final matching object. Since all of the matching objects have now been returned to the requestor, this <iterateResponse> contains no <iterator>.

```
<iterateResponse requestID="154" status="success">
    <update timestamp="20050704115900" updateKind="modify">
        <psoID ID="2244" targetID="target2"/>
    </update>
</iterateResponse>
```

3.6.9.3 closeIterator

The closeIterator operation tells the provider that the requestor has no further need for the updates result set that a specific <iterator> represents. (See the description of the updates operation above.)

A requestor should send a <closeIteratorRequest> to the provider when the requestor no longer intends to iterate an updates result set. (A provider will eventually free an inactive updates result set--even if the provider never receives a <closeIteratorRequest> from the requestor--but this behavior is unspecified.) For more information, see the topic named "Resource Considerations" topic earlier within this section.

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"/>
```

A closeIteratorRequest receives a closeIteratorResponse. A requestor supplies as input to a <closeIteratorRequest> the <iterator> that was part of the original <updatesResponse> or the <iterator> that was part of a subsequent <iterateResponse>, whichever is most recent. A provider returns a <closeIteratorResponse> in response to each <closeIteratorRequest>. A <closeIteratorResponse> has the same structure as an <spml:response>.

closeIterator is not batchable. For reasons of scale, neither an updates request nor an iterate request nor a closeIterator request should be nested in a batch request. When an updates query matches more updates than the provider can place directly in the response, the provider must temporarily store the remaining updates. Storing the remaining updates allows the requestor to iterate the remaining updates, but also requires the provider to commit resources. See the topic named “Resource Considerations” earlier in this section.

Batch responses also tend to be large. Batch operations are typically asynchronous, so storing the results of asynchronous batch operations imposes on providers a resource burden similar to that of search results. Allowing a requestor to nest an updates request or an iterate request or a closeIterator request within a batch request would aggravate the resource problem, requiring a provider to store more information in larger chunks for a longer amount of time.

The closeIterator operation must be executed synchronously. The provider is already queuing the result set (every update beyond those returned in the first updates response), so a request to close the iterator (and thus to free the system resources associated with the result set) should be executed as soon as possible. It is unreasonable for a requestor to ask the provider to queue the results of a request to close an iterator (especially since the close iterator response contains little or no information beyond success or failure).

3.6.9.3.1 closeIteratorRequest (normative)

A requestor SHOULD send a <closeIteratorRequest> to a provider when the requestor no longer intends to iterate an updates result set. (This allows the provider to free any system resources associated with the updates result set.).

Execution. A <closeIteratorRequest> MUST NOT specify "executionMode='asynchronous'".
A <closeIteratorRequest> MUST specify "executionMode='synchronous'" or (a <closeIteratorRequest> MUST) omit "executionMode". See the section titled "Determining execution mode".

iterator. A <closeIteratorRequest> MUST contain exactly one <iterator> element. A requestor MUST supply as input to a <closeIteratorRequest> the <iterator> from the original <updatesResponse> or (a requestor MUST supply the <iterator>) from a subsequent <iterateResponse>. A requestor SHOULD supply as input to a <closeIteratorRequest> the most recent <iterator> that represents the updates result set.

iterator ID. An <iterator> that is part of a <closeIteratorRequest> MUST have an "ID" attribute. (The value of the "ID" attribute uniquely identifies the <iterator> within the namespace of the provider. The "ID" attribute allows the provider to map each <iterator> token to the result set of the requestor's <query> and also (allows the provider to map each <iterator> token) to any state that records the requestor's iteration *within* that result set.)

3.6.9.3.2 closeIteratorResponse (normative)

A provider that receives a <closeIteratorRequest> from a requestor that the provider trusts must examine the content of the <closeIteratorRequest>. If the request is valid, the provider MUST release any updates result set that the <iterator> represents. Any subsequent request to iterate that same updates result set MUST fail.

Execution. The provider MUST execute the closeIterator operation synchronously (if the provider executes the closeIterator operation at all). See the section titled "Determining execution mode".

Response. The provider MUST return to the requestor a <closeIteratorResponse>.

Status. The <closeIteratorResponse> must contain a "status" attribute that indicates whether the provider successfully released the updates result set that the <iterator> represents. See the section titled "Status (normative)".

- If the provider successfully released the updates result set that the <iterator> represents, then the <closeIteratorResponse> MUST specify "status='success'".
- If the provider encountered an error in releasing the updates result set that the <iterator> represents, then the <closeIteratorResponse> MUST specify "status='failure'".

Error. If the <closeIteratorResponse> specifies "status='failure'", then the <closeIteratorResponse> MUST have an "error" attribute that characterizes the failure. See the general section titled "Error (normative)".

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

- If the provider does not recognize the <iterator> in a <closeIteratorRequest> as representing an updates result set.

- If the provider does not recognize the `<iterator>` in a `<closeIteratorRequest>` as representing any updates result set that the provider currently maintains.

- If the provider recognized the `<iterator>` in a `<closeIteratorRequest>` as representing a updates result set that the provider currently maintains but *cannot release the resources associated with that updates result set*.

The `<closeIteratorResponse>` MAY specify an appropriate value of `"error"` if any of the following is true:

- If a `<closeIteratorRequest>` contains an `<iterator>` that is *not the most recent version* of the `<iterator>`. If the provider has returned to the requestor a more recent `<iterator>` that represents the same updates result set, then the provider MAY reject the older `<iterator>`.
  (A provider that changes the ID—for example, to encode the state of iteration within a updates result set—may be sensitive to this.)

3.6.9.3.3 closeIterator Examples (non-normative)

In order to illustrate the iterate operation, we first need an updates operation that returns more than one update. In the following example, a requestor asks a provider to return updates for every `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>
```

The provider returns a `<updatesResponse>`. The `"status"` attribute of the `<updatesResponse>` indicates that the provider successfully executed the updates operation. The `<updatesResponse>` contains two `<update>` elements that represent the first matching updates.

```
<updatesResponse requestID="152" status="success">
    <update timestamp="1944062400000000" updateKind="add">
        <psoID ID="0001" targetID="target2"/>
    </update>
    <update timestamp="1942092700000000" updateKind="add">
        <psoID ID="0002" targetID="target2"/>
    </update>
    <update timestamp="1970091800000000" updateKind="delete">
        <psoID ID="0002" targetID="target2"/>
    </update>
    <iterator ID="1970"/>
</updatesResponse>
```

The requestor decides that the two objects in the initial `<searchResponse>` will suffice, and does not intend to retrieve any more matching objects (in the result set for the search). The requestor supplies the `<iterator>` from the `<updatesResponse>` as input to the `<closeIteratorRequest>`.

```
<closeIteratorRequest requestID=”153”>
    <iterator ID=”1900”/>
</closeIteratorRequest>
```

The provider returns a `<closeIteratorResponse>` in response to the `<closeIteratorRequest>`. The `“status”` attribute of the `<closeIteratorResponse>` indicates that the provider successfully released the result set.

```
<closeIteratorResponse requestID=”153” status=“success"/>
```

3.7 Custom Capabilities

The features of SPMLv2 that allow the PSTC to define optional operations as part of standard capabilities are *open mechanisms* that will work for anyone. An individual provider (or any third party) can define a custom capability that integrates with SPMLv2. Whoever controls the namespace of the capability controls the extent to which it can be shared. Each provider determines which capabilities are supported for which types of objects on which types of targets.

The SPMLv2 capability mechanism is extensible. Any party may define additional capabilities. A provider declares its support for a custom capability in exactly the same way that it declares support for a standard capability: as a target `<capability>` element.

The standard capabilities that SPMLv2 defines will not address all needs. Contributors may define additional custom capabilities.

Since the schema for each capability is defined in a separate namespace, a custom capability will not ordinarily conflict with a standard capability that is defined as part of SPMLv2, nor will a custom capability ordinarily conflict with another custom capability. In order for a custom capability B to conflict with another capability A, capability B would have to import the namespace of capability A and re-declare a schema element from capability A. Such a conflict is clearly intentional and a provider can easily avoid such a conflict by not declaring support for capability B.

Also see the section titled “Conformance”.

4 Conformance (normative)

4.1 Core operations and schema are mandatory

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.

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.

4.2 Standard capabilities are optional

A conformant provider SHOULD support the XML schema and operations defined by each standard capability of SPMLv2.

4.3 Custom capabilities must not conflict

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.

A conformant provider MAY support any custom capability that conforms to SPMLv2.

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:

- The request type (directly or indirectly) extends `{RequestType}`
- The response type is `{ResponseType}` or (the response type directly or indirectly) extends `{ResponseType}`.

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:

- 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.

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:

- 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*.

4.4 Capability Support is all-or-nothing

A provider that claims to support a particular capability for (a set of schema entities on) a target MUST support (for every instance of those schema entities on the target) every operation that the capability defines.

4.5 Capability-specific data

A capability MAY imply capability-specific data. That is, a capability MAY specify that data specific to that capability may be associated with one or more objects. (For example, the Reference Capability implies that each object may contain a set of references to other objects.)

Any capability that implies capability-specific data MAY rely on the default processing that SPMLv2 specifies for capability-specific data (see the section titled "CapabilityData Processing (normative)"). However, any capability that implies capability-specific data SHOULD specify the structure of that data. (For example, the Reference Capability specifies that its capability-specific data must contain at least one `<reference>` and should contain only `<reference>` elements.)

Furthermore, any capability that implies capability-specific data and *for which the default processing of capability-specific data is inappropriate* (i.e., any capability for which an instance of `{CapabilityDataType}` that refers to the capability would specify "mustUnderstand='true'")

- MUST specify the structure of that capability-specific data.
- MUST specify how core operations should handle that capabilityData. (For example, the Reference Capability specifies how each reference must be validated and processed. See the section titled "Reference CapabilityData Processing (normative).)

5 Security Considerations

5.1 Use of SSL 3.0 or TLS 1.0

When using Simple Object Access Protocol (SOAP) **[SOAP]** as the protocol for the requestor (client) to make SPMLv2 requests to a provider (server), Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** SHOULD be used.

The TLS implementation SHOULD implement the TLS_RSA_WITH_3DES_EDE_CBC_SHA or the TLS_RSA_WITH_AES_128_CBC_SHA **[AES]** cipher suite.

5.2 Authentication

When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as the SOAP **[SOAP]** transport protocol, the provider (server) SHOULD be authenticated to the requestor (client) using X.509 v3 **[X509]** service certificates. The requestor (client) SHOULD be authenticated to the provider (server) using X.509 v3 service certificates.

For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require intermediaries, Web Services Security **[WSS]** SHOULD be used for authentication.

5.3 Message Integrity

When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as the SOAP **[SOAP]** transport protocol, message integrity is reasonably assured for point-to-point message exchanges.

For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require intermediaries, Web Services Security **[WSS]** SHOULD be used to ensure message integrity.

5.4 Message Confidentiality

When using Secure Sockets Layer (SSL 3.0) or Transport Layer Security (TLS 1.0) **[RFC 2246]** as the SOAP **[SOAP]** transport protocol, message confidentiality is reasonably assured for point-to-point message exchanges, and for the entire message.

For SOAP requests that are not made over SSL 3.0 or TLS 1.0, or for SOAP requests that require intermediaries, Web Services Security **[WSS]** SHOULD be used to ensure confidentiality for the sensitive portions of the message.

4515 # 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>

   <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>
```

4516

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"/>
            </sequence>
            <attribute name="recursive" type="boolean" use="optional"/>
         </extension>
      </complexContent>
   </complexType>

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

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

   <element name="bulkDeleteRequest"
type="spmlbulk:BulkDeleteRequestType"/>
   <element name="bulkDeleteResponse" type="spml:ResponseType"/>

</schema>
```

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>
```

4524

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 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

4527 # 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>

   <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>
```

4529

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>

   <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

[AES] National Institute of Standards and Technology (NIST), FIPS-197: Advanced Encryption Standard, **http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf**, National Institute of Standards and Technology (NIST)

[ARCHIVE-1] OASIS Provisioning Services Technical Committee, email archive, **http://www.oasis-open.org/apps/org/workgroup/provision/email/archives/index.html**, OASIS PS-TC

[DS] IETF/W3C, *W3C XML Signatures*, **http://www.w3.org/Signature/**, W3C/IETF

[DSML] OASIS Directory Services Markup Standard, *DSML V2.0 Specification*, **http://www.oasis-open.org/specs/index.php#dsmlv2**, OASIS DSML Standard

[GLOSSARY] OASIS Provisioning Services TC, *Glossary of Terms*, **http://www.oasis-open.org/apps/org/workgroup/provision/download.php**, OASIS PS-TC

[RFC 2119] S. Bradner., *Key words for use in RFCs to Indicate Requirement Levels*, **http://www.ietf.org/rfc/rfc2119.txt**, IETF

[RFC 2246] T. Dierks and C. Allen, *The TLS Protocol*, **http://www.ietf.org/rfc/rfc2246.txt**, IETF

[SAML] OASIS Security Services TC, **http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security**, OASIS SS-TC

[SOAP] W3C XML Protocol Working Group, **http://www.w3.org/2000/xp/Group/**

[SPML-Bind] OASIS Provisioning Services TC, SPML V1.0 Protocol Bindings, **http://www.oasis-open.org/apps/org/workgroup/provision/download.php/1816/draft-pstc-bindings-03.doc**, OASIS PS-TC

[SPML-REQ] OASIS Provisioning Services Technical Committee, *Requirements*, **http://www.oasis-open.org/apps/org/workgroup/provision/download.php/2277/draft-pstc-requirements-01.doc**, OASIS PS-TC

[SPML-UC] OASIS Provisioning Services Technical Committee, *SPML V1.0 Use Cases*, **http://www.oasis-open.org/apps/org/workgroup/provision/download.php/988/draft-spml-use-cases-05.doc**, OASIS PS-TC

[SPMLv2-Profile-DSML] OASIS Provisioning Services Technical Committee, SPMLv2 DSMLv2 Profile, OASIS PS-TC

[SPMLv2-Profile-XSD] OASIS Provisioning Services Technical Committee, SPML V2 XSD Profile, OASIS PS-TC

[SPMLv2-REQ] OASIS Provisioning Services Technical Committee, Requirements, OASIS PS-TC

[SPMLv2-ASYNC] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Async Capability of SPMLv2, OASIS PS-TC

[SPMLv2-BATCH] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Batch Capability of SPMLv2, OASIS PS-TC

[SPMLv2-BULK] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Bulk Capability of SPMLv2, OASIS PS-TC

[SPMLv2-CORE] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Core Operations of SPMLv2, OASIS PS-TC

[SPMLv2-PASS] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Password Capability of SPMLv2, OASIS PS-TC

[SPMLv2-REF] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Reference Capability of SPMLv2, OASIS PS-TC

[SPMLv2-SEARCH] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Search Capability of SPMLv2, OASIS PS-TC

[SPMLv2-SUSPEND] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Suspend Capability of SPMLv2, OASIS PS-TC

[SPMLv2-UPDATES] OASIS Provisioning Services Technical Committee, XML Schema Definitions for Updates Capability of SPMLv2, OASIS PS-TC

[SPMLv2-UC] OASIS Provisioning Services Technical Committee., SPML V2.0 Use Cases, OASIS PS-TC

[WSS] OASIS Web Services Security (WSS) TC, **http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss**, OASIS SS-TC

[X509] RFC 2459 - Internet X.509 Public Key Infrastructure Certificate and CRL Profile, **http://www.ietf.org/rfc/rfc2459.txt**

[XSD] W3C Schema WG ., *W3C XML Schema*, **http://www.w3.org/TR/xmlschema-1/** W3C

Appendix K. Acknowledgments

The following individuals were voting members of the Provisioning Services committee at the time that this version of the specification was issued:

Jeff Bohren, BMC
Robert Boucher, CA
Gary Cole, Sun Microsystems
Rami Elron, BMC
Marco Fanti, Thor Technologies
James Hu, HP
Martin Raepple, SAP
Gavenraj Sodhi, CA
Kent Spaulding, Sun Microsystems

Appendix L. Notices

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

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